Exploración multi-robot basada en grillas de ocupación probabilística y diagramas de Voronoi

Estudiante: Federico Ciuffardi

Supervisor: Facundo Benavides

Informe de Proyecto de Grado presentado al Tribunal Evaluador
como requisito de graduación de la carrera Ingeniería
en Computación

Montevideo, Uruguay

20 de abril de 2022

Resumen

La exploración es un problema fundamental de la robótica móvil autónoma que consiste en utilizar un robot para obtener información de un entorno desconocido, a través de sus sensores, con el objetivo de generar un mapa que lo represente. Por motivos de eficiencia y robustez la exploración suele llevarse a cabo con más de un robot, en este caso el problema se conoce como el de exploración multi-robot.

Uno de los principales aspectos de la exploración es determinar a que lugares los robots deben moverse para obtener información del entorno. Para esto se suele primero identificar dichos lugares, conocidos como objetivos de exploración, para luego asignar los objetivos identificados a los robots. Cuando se utiliza más de un robot es deseable que la asignación de objetivos se lleve a cabo siguiendo una estrategia de coordinación para evitar que los robots exploren los mismos lugares o que se obstaculicen entre sí.

Los entornos estructurados como edificios, hogares y otras construcciones humanas son entornos que pueden dividirse en segmentos, como habitaciones y corredores. En este tipo de entornos una posible estrategia de coordinación es la de llevar a cabo la exploración maximizando la distribución de los robots sobre los segmentos.

El presente proyecto de grado se planteó como objetivo mejorar la propuesta original de la estrategia de coordinación antes mencionada, resultando en cinco aportes sobre aspectos específicos de la estrategia y del problema de exploración multi-robot en general.

Dos de los aportes se concentran en acelerar la construcción de una estructura llamada Diagrama Generalizado de Voronoi (GVD por sus siglas en inglés), que es esencial mantener actualizada para reconocer los segmentos sobre los cuales distribuir a los robots durante la exploración. El primer aporte optimiza la actualización solo modificando las partes desactualizadas de la última versión del GVD. El segundo reduce la construcción del GVD a las partes conocidas del entorno, ahorrando el tiempo de construir sobre las partes desconocidas.

El tercer aporte consiste en una forma novedosa de identificar los objetivos de exploración que se basa en las capacidades sensoriales de los robots para evitar identificar objetivos redundantes.

El cuarto aporte introduce un algoritmo de asignación de objetivos de exploración que logra maximizar la distribución de los robots sobre los segmentos evitando asignar más robots a un segmento que los objetivos que contenidos en él.

El quinto y último aporte consiste en un método para planificar caminos hacia los objetivos de exploración que hace uso del GVD (ya disponible por ser usado en la identificación de segmentos) para reducir los tiempos de planificación, pero a su vez permite que los caminos no sean completamente sobre el GVD evitando planificar caminos innecesariamente largos.

Se implementó una solución al problema de exploración multi-robot que incluye los cinco aportes antes mencionados. Esta solución se puso a prueba dentro de un simulador. Los resultados de las pruebas en general confirman que los aportes logran su objetivo de mejorar la propuesta original, siendo los aportes asociados a la aceleración de la construcción del GVD los de mayor impacto.

Índice general

Índice de figuras IV

Índice de cuadros V

Índice de algoritmos VI

Glosario VII

1. Introducción 1
- 1.1. Contribuciones . . . 2
- 1.2. Organización del documento . . . 3

2. Marco de trabajo 4
- 2.1. Exploración . . . 5
  - 2.1.1. Exploración multi-robot . . . 6
- 2.2. Mapas . . . 6
  - 2.2.1. Mapa de carreteras . . . 8
  - 2.2.2. Diagrama de Voronoi . . . 9
  - 2.2.3. Diagrama generalizado de Voronoi . . . 10
  - 2.2.4. Construcción de un GVD . . . 12
  - 2.2.5. Construcción incremental de un GVD . . . 14
  - 2.2.6. Generación de mapas topológicos basados en regiones a partir de un GVD . . . 18
  - 2.2.7. Alternativas para la generación de mapas topológicos basados en regiones . . . 20
  - 2.2.8. Criterios de calidad de un mapa topológico basado en regiones . . . 21
- 2.3. Coordinación en la exploración multi-robot . . . 22
  - 2.3.1. Coordinación a partir de segmentaciones topológicas . . . 22
  - 2.3.2. Coordinación a partir de segmentaciones no topológicas . 23
- 2.4. Simuladores . . . 25
  - 2.4.1. Características . . . 26
  - 2.4.2. Experimentación . . . 27
  - 2.4.3. Conclusiones . . . 28

3. Solución propuesta 29
- 3.1. Hipótesis de trabajo . . . 30
- 3.2. Arquitectura . . . 31
  - 3.2.1. Move Base . . . 31
  - 3.2.2. Combinador de mapas . . . 32
  - 3.2.3. Controlador central . . . 33
  - 3.2.4. Controlador del robot . . . 33
  - 3.2.5. Controlador de movimiento . . . 33

3.3. Definiciones . . . 33
3.3.1. Grillas de ocupación . . . 33
3.3.2. Componentes conexas . . . 34
3.4. Identificación de objetivos . . . 36
3.4.1. Fronteras . . . 36
3.4.2. Fronteras significativas basadas en K-Means . . . 36
3.4.3. Fronteras significativas basadas en cubrimiento . . . 38
3.5. Asignación de objetivos . . . 41
3.5.1. Obtención de información . . . 41
3.5.2. Valuación . . . 42
3.5.3. Resolución . . . 43
3.6. Segmentación . . . 46
3.7. Diagrama generalizado de Voronoi . . . 50
3.7.1. Criterio de pertenencia . . . 50
3.7.2. Consideraciones sobre el espacio desconocido . . . 51
3.8. Planificación . . . 53
3.8.1. Criterios de compleción . . . 54
3.8.2. Planificación jerárquica . . . 55
3.9. Construcción cooperativa del mapa . . . 56

4. Experimentación **58**
4.1. Especificación de las pruebas . . . 59
4.1.1. Simulador . . . 59
4.1.2. Tipos de pruebas . . . 59
4.1.3. Entorno . . . 60
4.1.4. Robots . . . 60
4.1.5. Software . . . 61
4.1.6. Hardware . . . 61
4.2. Métricas . . . 61
4.3. Resultados . . . 62
4.3.1. Completitud y calidad de los mapas . . . 62
4.3.2. Incrementalidad . . . 65
4.3.3. Identificación de objetivos . . . 67
4.3.4. Consideración del espacio desconocido . . . 70

5. Conclusiones y trabajo futuro **73**
5.1. Conclusiones . . . 74
5.2. Trabajo a futuro . . . 75
5.2.1. Experimentación con robots reales . . . 75
5.2.2. Identificación de objetivos . . . 75
5.2.3. Mapa topológico . . . 76
5.2.4. Algoritmo de asignación de objetivos y Planificación . . . 76
5.2.5. Potenciales mejoras de rendimiento . . . 76
5.2.6. Reusabilidad . . . 76

Appendices **77**

A. Algoritmos completos **78**
A.1. Obtención de celdas base . . . 78

B. Ejemplos completos **80**
B.1. Obtención de fronteras significativas basada en cubrimiento . . . 80

Índice de figuras

2.1. Tipos de mapas. 7
2.2. Mapa del sistema de transporte subterráneo de Buenos Aires, Argentina. 7
2.3. Red de carreteras de Uruguay. 9
2.4. Diagrama de Voronoi. 10
2.5. Distribución de generadores. 11
2.6. Diagrama generalizado de Voronoi. 12
2.7. Método de construcción de un GVD. 13
2.8. Algoritmo brushfire. 14
2.9. Algoritmo brushfire dinámico. 16
2.10. Ondas de brushfire dinámico. 17
2.11. Resultados de brushfire dinámico original y la variante propuesta por Lau *et al.* 18
2.12. Puntos críticos y líneas críticas. 19
2.13. Método de construcción de un mapa topológico basado en regiones a partir de un GVD. 19
2.14. Método de construcción de un mapa topológico basado en regiones a partir de aprendizaje automático. 20
2.15. Segmentación propuesta en [Solanas and Garcia, 2004]. 24
2.16. Segmentación propuesta en [Lopez-Perez et al., 2018] 25
2.17. Implementaciones del escenario de simulación. 27

3.1. Arquitectura de la solución propuesta. 31
3.2. Arquitectura del stack de navegación de ROS. 32
3.3. Vecinos de una celda en una grilla de ocupación. 34
3.4. Descomposición en componentes conexas. 35
3.5. Proceso de obtención de fronteras significativas basado en K-Means. 37
3.6. Resultado subóptimo del método de obtención de fronteras significativas basado en K-Means. 38
3.7. Proceso de obtención de fronteras significativas basado en cubrimiento. 41
3.8. Camino. 43
3.9. Caso en el cual se requiere de una tolerancia para detectar correctmente a las celdas base. 48
3.10. Proceso de segmentación de un entorno representado con una grilla. 49
3.11. Línea crítica discretizada diagonal, y sus efectos en la segmentacion al usar ady_4 o ady_8. 49
3.12. Resultado de considerar las celdas de estado desconocido iguales a las de estado libre. 52
3.13. Formas de considerar el espacio desconocido. 53
3.14. Conectividad del GVD en diferentes formas de considerar el espacio desconocido. 53
3.15. Planificación jerárquica. 56

3.16. Aplicación de la regla de actualización propuesta en [Stachniss, 2009] en un corredor. 57

4.1. Mapa del entorno utilizado en las pruebas. 60
4.2. Robot diferencial Pioneer 3-DX. 61
4.3. Mapas de referencia utilizados. 62
4.4. Completitud del mapa en función de celdas por metro cuadrado. 63
4.5. Celda explorable que no se reconoce como tal. 64
4.6. Calidad del mapa en función de celdas por metro cuadrado. . . . 64
4.7. Tiempo promedio en construcción de GVD en función de celdas por metro cuadrado. 66
4.8. Tiempo de exploración en función de celdas por metro cuadrado. 66
4.9. Distancia total recorrida por la flota en función de celdas por metro cuadrado. 67
4.10. Tiempo de exploración en función de celdas por metro cuadrado. 68
4.11. Distancia total recorrida por la flota en función de celdas por metro cuadrado. 69
4.12. Tiempo promedio en obtención de información función de celdas por metro cuadrado. 70
4.13. Tiempo promedio en construcción de GVD en función de celdas por metro cuadrado. 71
4.14. Tiempo de exploración en función de celdas por metro cuadrado. 72
4.15. Distancia total recorrida por la flota en función de celdas por metro cuadrado. 72

B.1. Proceso de obtención de fronteras significativas basado en cubrimiento. 82

Índice de cuadros

2.1. Comparación entre simuladores. 26

4.1. Completitud y calidad de los mapas obtenidos en todas las pruebas realizadas. 63
4.2. Construcción del GVD: incremental vs. no incremental. 65
4.3. Identificación de objetivos: comparación de métodos. 68
4.4. Construcción del GVD: incidencia del espacio desconocido. . . . 70

Índice de algoritmos

1. Estrategia Coordinación [Wurm et al., 2008] 23

2. Descomposición en componentes conexas de C 35
3. Obtención de fronteras significativas basada en cubrimiento 39
4. Asignación de objetivos a robots 45
5. Obtención de las celdas base CB_c de una celda c (simplificada) . . 51

6. Obtención de las celdas base CB_c de una celda c 78

Glosario

ROS Framework para el desarrollo de aplicaciones robóticas.

LiDAR Dispositivo que permite determinar la distancia desde emisores láser a superficies midiendo el tiempo que tarda la luz reflejada en regresar a los receptores.

NUE Sistema de coordenadas donde el Norte se asocia al eje X positivo, Arriba al eje Y-positivo y el Este al eje Z-positivo.

ENU Sistema de coordenadas donde el Este se asocia al eje X positivo, el Norte al eje Y positivo y Arriba al eje Z positivo.

Capítulo 1

Introducción

Contenidos

1.1. Contribuciones 2
1.2. Organización del documento 3

La exploración es un problema clásico de la robótica móvil autónoma que consiste en utilizar un robot para obtener información de un entorno desconocido, a través de sus sensores, con el objetivo de generar un mapa que lo represente. La exploración es una parte fundamental en tareas de limpieza [Luo and Yang, 2002], operaciones de búsqueda y rescate [Liu et al., 2015], misiones extra planetarias [Schuster et al., 2019], entre otras tareas en las cuales se desconozca el entorno y sea ineficiente, o directamente inviable teleoperar a un robot.

La exploración suele ser una tarea altamente paralelizable, al ser usual que en un mismo instante de tiempo existan varios lugares del entorno de los cuales un robot puede extraer información novedosa. Esto causa que sea atractivo el uso de varios robots para acelerar la exploración, y en este caso el problema pasa a conocerse como el de exploración multi-robot.

Sin embargo, al tener varios robots dedicados a la tarea de exploración es necesario algún mecanismo de coordinación que logre explotar el paralelismo. Para esto se debe evitar situaciones subóptimas como que varios robots realicen trabajo redundante al explorar los mismos lugares, o que estos interfieran entre sí causando perdidas de tiempo en desvíos para evitar colisiones.

Los entornos estructurados como edificios, hogares y otras construcciones humanas son entornos que pueden dividirse en segmentos, como habitaciones y corredores. En [Wurm et al., 2008] se introduce una estrategia de coordinación para la exploración multi-robot en entornos estructurados que consiste en llevar a cabo la exploración maximizando la distribución de los robots sobre los segmentos. La estrategia se fundamenta en que las exploraciones de segmentos diferentes suelen ser independientes entre sí, por lo cual asignar robots a diferentes segmentos ayuda a evitar redundancia en la exploración. También en que los segmentos pueden ser demasiado pequeños para que un segundo robot acelere su exploración. Y adicionalmente en que de haber más de un robot explorando un mismo segmento estos pueden bloquearse entre sí mientras intentan abandonarlo a la vez al finalizar su exploración.

El proyecto de grado se propone mejorar la propuesta de [Wurm et al., 2008], para esto se evaluaron varios aspectos de la propuesta en cuestión y del problema de exploración multi-robot en general, relevando una serie de potenciales mejoras que resultan en las seis contribuciones que se describen a continuación.

1.1. Contribuciones

La estrategia de coordinación requiere identificar los segmentos del entorno, lo cual se conoce como segmentación. Uno de los métodos más populares y eficientes para segmentar, que es también utilizado en la propuesta original, se basa en utilizar una estructura llamada Diagrama Generalizado de Voronoi (GVD por sus siglas en inglés). Dado que para aplicar la estrategia de coordinación los segmentos deben mantenerse actualizados durante la exploración, el GVD también debe mantenerse actualizado. En la propuesta original el GVD se mantiene actualizado aplicando un algoritmo no incremental que reconstruye completamente el GVD en cada actualización. Dado que gran parte del GVD se mantiene igual con respecto a su última actualización, la reconstrucción completa se considera ineficiente. Visto esto la **primera contribución** consiste en adaptar e integrar a la propuesta original un algoritmo incremental que optimice la actualización del GVD solo cambiando las partes necesarias para que la última versión disponible del GVD quede actualizada.

Por otro lado la definición de GVD no establece cómo tratar las porciones desconocidas de un entorno parcialmente explorado. La **segunda contribución** consiste en una forma novedosa de tratar con dichas porciones durante

la construcción del GVD, que logra optimizar la construcción reduciéndola a las porciones conocidas en lugar de construir sobre todo el entorno (conocido y desconocido) como se hace usualmente.

Uno de los principales aspectos de la exploración multi-robot es el de determinar a qué lugares se debe enviar a los robots a explorar. Para esto se suele primero identificar dichos lugares, conocidos como objetivos de exploración, para luego asignar los objetivos identificados a los robots.

Respecto a la identificación de objetivos de exploración, identificar grandes cantidades de objetivos no es deseable en tanto complejiza la posterior tarea de asignarlos a los robots. La **tercera contribución** es un método novedoso que busca reducir la cantidad de objetivos identificados eliminando objetivos redundantes basándose en las capacidades sensoriales de los robots.

Respecto a la asignación de objetivos de exploración, en el contexto de la estrategia de coordinación de [Wurm et al., 2008], se debe maximizar la distribución de los robots sobre los segmentos. Para lograr esto la propuesta original primero asigna robots a los segmentos, y luego distribuye los objetivos de cada segmento sobre sus robots asignados. El problema es que durante la asignación de los robots sobre los segmentos no se considera el número de objetivos que hay en cada segmento, y por lo tanto es posible asignar más robots a un segmento que los objetivos que hay disponibles en este, dejando robots ociosos que podrían ser necesarios en otro segmento. La **cuarta contribución** consiste en un algoritmo de asignación de objetivos que logra maximizar la distribución de los robots sobre los segmentos considerando los objetivos que existen en cada segmento para evitar robots ociosos.

Luego de que un robot recibe un objetivo de exploración, este debe moverse hacia él, esto introduce la necesidad de planificar caminos hacia los objetivos. En el contexto del trabajo desarrollado hay dos métodos de planificar que son de especial interés: planificar sobre el GVD, y planificar sobre todo el espacio libre. Planificar sobre el GVD es más rápido, y genera caminos más seguros, mientras que planificar sobre todo el espacio libre lleva a caminos más cortos. La **quinta contribución** es la propuesta de un método de planificación que combina los métodos antes descritos de forma jerárquica para obtener la velocidad de la planificación sobre el GVD y el largo de los caminos planificados sobre todo el espacio libre.

Las contribuciones antes mencionadas se implementaron como parte de una solución al problema de exploración multi-robot. Esta solución fue utilizada para realizar pruebas dentro de un simulador con el propósito de validar y estudiar la utilidad las contribuciones. La solución implementada es la **sexta contribución** del proyecto y se encuentra disponible en línea[1].

1.2. Organización del documento

El resto del documento se organiza de la siguiente manera. El capítulo 2 introduce problemas, artículos, herramientas y conceptos que conforman la base del trabajo desarrollado. En el capítulo 3 se describe la solución al problema multi-robot implementada, incluyendo las potenciales mejoras que constituyen las contribuciones del proyecto de grado. El capítulo 4 se dedica especificar las pruebas realizadas, presentar sus resultados y analizarlos. Por último en el capítulo 5 se plantean las conclusiones del proyecto de grado y se mencionan líneas de trabajo futuro que surgen a partir del trabajo realizado.

[1] https://gitlab.fing.edu.uy/federico.ciuffardi/pgmappingcooperativo
Accedido por última vez: 25/02/2022.

Capítulo 2

Marco de trabajo

Contenidos

2.1. Exploración 5
2.1.1. Exploración multi-robot 6
2.2. Mapas 6
2.2.1. Mapa de carreteras 8
2.2.2. Diagrama de Voronoi 9
2.2.3. Diagrama generalizado de Voronoi 10
2.2.4. Construcción de un GVD 12
2.2.5. Construcción incremental de un GVD 14
2.2.6. Generación de mapas topológicos basados en regiones a partir de un GVD 18
2.2.7. Alternativas para la generación de mapas topológicos basados en regiones 20
2.2.8. Criterios de calidad de un mapa topológico basado en regiones 21
2.3. Coordinación en la exploración multi-robot 22
2.3.1. Coordinación a partir de segmentaciones topológicas 22
2.3.2. Coordinación a partir de segmentaciones no topológicas 23
2.4. Simuladores 25
2.4.1. Características 26
2.4.2. Experimentación 27
2.4.3. Conclusiones 28

Este capítulo tiene como objetivo introducir aspectos que conforman la base del trabajo desarrollado. A lo largo de este se tratan tanto problemas relevantes, como algunas de sus soluciones presentes en el estado del arte, finalizando con una discusión sobre una herramienta fundamental para el proyecto, la simulación.

2.1. Exploración

La exploración es un problema clásico de la robótica móvil autónoma que consiste en utilizar un robot con el objetivo de obtener información de un entorno desconocido generando un mapa que lo represente. La exploración es fundamental en tareas de limpieza [Luo and Yang, 2002], operaciones de búsqueda y rescate [Liu et al., 2015], misiones extra planetarias [Schuster et al., 2019], entre otras tareas en las cuales se desconozca el entorno y sea ineficiente o directamente inviable teleoperar a un robot.

La información del entorno es recopilada por los sensores del robot. Dado que dichos sensores tienen un rango máximo y adicionalmente suelen existir obstáculos que limitan aún más ese rango, lo normal es que sea necesario que el robot se mueva para que sus sensores puedan obtener información de las distintas partes del entorno a explorar.

Determinar el siguiente lugar al que es conveniente enviar a un robot para obtener información del entorno es uno de los principales problemas de la exploración. Este problema suele separarse en dos partes, (i) identificar dichos lugares, conocidos como objetivos de exploración, y (ii) asignar uno de los objetivos identificados al robot. Para identificar objetivos de exploración, un método popular es el propuesto en [Yamauchi, 1997] en el que se identifican como objetivos a las fronteras entre el espacio conocido y desconocido. Por otro lado para establecer qué objetivo asignar al robot se suele decidir cual de los objetivos identificados es el más conveniente, donde la conveniencia de un objetivo puede calcularse como un balance entre el costo y el beneficio de completarlo. El costo normalmente involucra aspectos como la distancia del recorrido que el robot debe hacer para llegar al objetivo, el tiempo que lleva recorrerlo y el costo energético que implica. Por otro lado el beneficio usualmente se asocia al tamaño de la porción explorada al completar el objetivo.

Otro aspecto fundamental es el denominado criterio de parada según el cual se determina cuando la exploración se considera finalizada. Uno de los criterios de parada tomados usualmente, es detener la exploración cuando el mapa construido alcanza cierto porcentaje de cubrimiento del entorno [Yan et al., 2015a]. Aunque este es utilizado usualmente para evaluaciones de rendimiento no es útil en la práctica cuando se desconocen las dimensiones del entorno a explorar, ya que esta información es necesaria para el cálculo del porcentaje de cubrimiento. Otro criterio de parada que sí es aplicable en la práctica, cuando el entorno es cerrado, es el de explorar el entorno hasta que no quede ninguna posibilidad de obtener nueva información de él, es decir que no exista espacio desconocido que sea posible explorar. La desventaja de este criterio es que se fuerza a una exploración exhaustiva del entorno, lo cual puede ser indeseable por cuestiones de costos, tanto temporales como de recursos (por ejemplo energía de los robots o su desgaste). En [Amorin et al., 2019] se propone un criterio de parada aplicable en la exploración de entornos cerrados, que logra evitar una exploración exhaustiva. La idea de este criterio es que es posible determinar cuándo los robots recopilan suficiente información de un entorno como para estimar el resto de forma precisa, y es en ese momento cuando la exploración se puede detener.

2.1.1. Exploración multi-robot

La exploración multi-robot consiste en utilizar una flota robótica para resolver el problema de exploración.

Una de las ventajas del uso de varios robots es la introducción de redundancia y aumento de tolerancia a fallos que esto implica. A su vez la exploración suele ser una tarea altamente paralelizable, al ser usual que en un mismo instante de tiempo existan varios lugares de los cuales un robot puede obtener información, por lo tanto es esperable que al aumentar la cantidad de recursos robóticos se experimente una reducción de los tiempos de exploración [Cao et al., 1997, Dudek et al., 1996, Guzzoni et al., 1997].

Sin embargo la reducción del tiempo de exploración no es proporcional al aumento de robots en la flota. En primer lugar siempre que el número de robots sea mayor que el número de objetivos de exploración, se tendrán recursos desaprovechados. De forma similar si varios robots son enviados a objetivos de exploración cercanos entre sí existe la posibilidad de que los sensores que estos utilizan para explorar recolecten información sobre una misma porción del entorno y por lo tanto realicen trabajo redundante. Adicionalmente al existir un mayor número de robots aumenta el riesgo de que estos se interfieran entre sí, causando perdidas de tiempo en desvíos para evitar colisiones [Guzzoni et al., 1997, Goldberg and Mataric, 1997].

Las problemáticas mencionadas pueden ser mitigadas asignando objetivos de forma coordinada buscando explotar el paralelismo en la exploración y reducir el tiempo perdido por interferencias [Nieto-Granda et al., 2014]. Más adelante en la sección 2.3 se describen algunas estrategias de coordinación para la exploración multi-robot.

2.2. Mapas

Un mapa es un modelo de un entorno. En el contexto de la robótica los mapas se pueden dividir en dos categorías, *mapas métricos* y *mapas topológicos* [Thrun, 1998, Choset et al., 2005].

Los *mapas métricos* son mapas principalmente caracterizados por representar la geometría de todo el entorno, incluyendo a su vez una escala que permite asociar las dimensiones representadas dentro del mapa a las reales.

Un tipo de mapa métrico es el denominado *mapa geométrico*, los mapas de este tipo utilizan primitivas geométricas para representar al entorno. Aunque es usual el uso de segmentos de recta como primitiva, es posible utilizar otras primitivas de ser conveniente. Un ejemplo de este tipo de mapa se puede ver en la figura 2.1a, siendo las primitivas utilizadas rectángulos.

Una *grilla de ocupación* es otro tipo de mapa métrico que consiste en representar el entorno mediante una grilla regular compuesta de celdas cuadradas que almacenan una estimación probabilística de su estado de ocupación que puede ser libre u ocupado. En la figura 2.1b se muestra una grilla de ocupación en la que cada celda se encuentra coloreada según la estimación probabilística que almacena, los colores oscuros indican una mayor probabilidad del estado ocupado y los claros una mayor probabilidad del estado libre.

Los *mapas topológicos* por otro lado son mapas simplificados al punto de solo incluir lugares topológicamente distintivos y conexiones entre estos. Normalmente se representan mediante grafos donde los lugares se modelan como vértices y la adyacencia entre lugares como aristas. Dependiendo del contexto es posible que los mapas topológicos contengan información adicional en los vértices y aristas, por ejemplo información métrica como lo puede ser el tamaño de la región para un vértice o la distancia entre regiones para las aristas.

Los tipos de mapas topológicos se determinan según lo representado por un vértice. En los *mapas topológicos basados en regiones* los vértices se corresponden a regiones de interés, como por ejemplo habitaciones y corredores. La figura 2.1c es un ejemplo de un mapa topológico basado en regiones. Notar que en este ejemplo, para que sea más ilustrativo, los vértices del grafo utilizado como representación están dispuestos para que coincidan con su correspondiente ubicación en el entorno, pero el grafo no tiene porque contener la información necesaria para esto.

(a) Mapa geométrico (b) Grilla de ocupación (c) Mapa topológico basado en regiones

Figura 2.1: Tipos de mapas. Un mismo entorno representado con distintos tipos de mapas. Extraído de [Choset et al., 2005].

La simplicidad de los mapas topológicos es útil para nosotros los humanos al facilitar la comprensión un entorno y como navegar en él, por lo que es usual ver mapas topológicos de sistemas de transporte (ver figura 2.2).

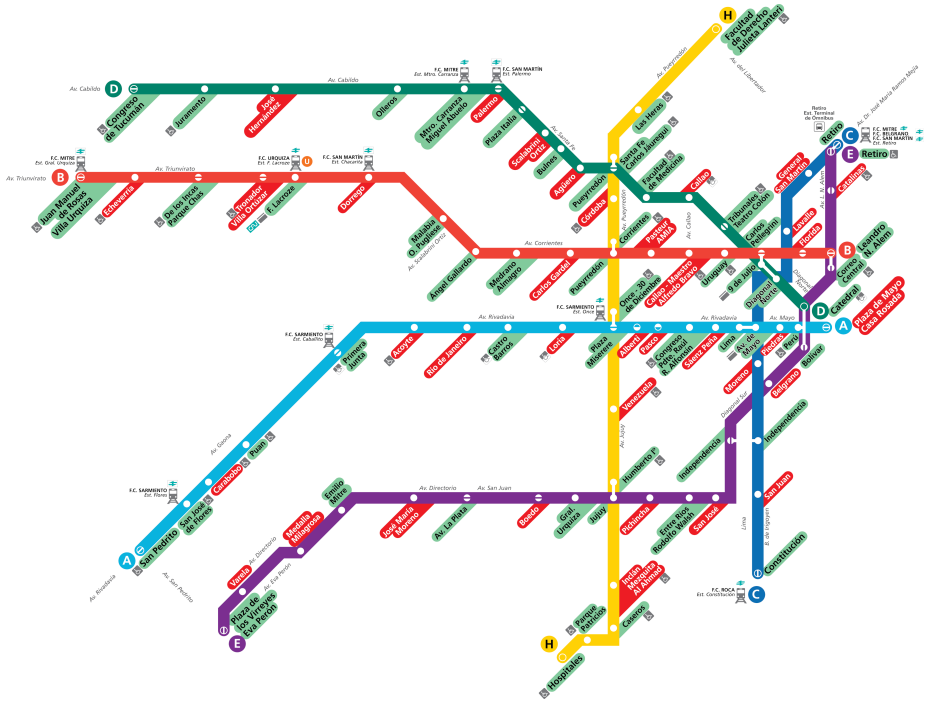


Figura 2.2: Mapa del sistema de transporte subterráneo de Buenos Aires, Argentina. Extraído de [www.buenosaires.gob.ar, 2021].

La simplicidad mencionada también tiene ventajas computacionales, ya que para un mismo entorno los mapas topológicos suelen ser ordenes de magnitud más pequeños que los mapas métricos y por lo tanto, más allá de la memoria

ahorrada, también se tiene una planificación de caminos más eficiente (menor espacio de búsqueda). Sin embargo la falta de detalle en la representación es también una desventaja en tanto hace que los caminos planificados con un mapa topológico suelan ser más largos que los obtenidos en mapas métricos [Thrun, 1998].

Dadas las ventajas y desventajas mencionadas existen trabajos [Thrun, 1998, Wurm et al., 2008, Liu et al., 2015] que proponen el uso simultáneo de mapas topológicos y métricos para "ganar lo mejor de los dos mundos" al tener disponible abstracción y detalle a la vez. Un ejemplo de cómo combinar las ventajas de ambos tipos de mapa es la planificación jerárquica descrita en [Thrun, 1998] donde se utiliza un mapa topológico y uno métrico de un mismo entorno con el propósito de generar planes para navegar en él. El mapa topológico es utilizado para generar un plan de navegación topológico que consiste en una secuencia de corredores y habitaciones que se deben atravesar para llegar a un objetivo, este plan no posee suficiente detalle como para traducirse directamente en acciones que el robot pueda realizar para llegar al objetivo. Por lo tanto se introduce una segunda etapa de planificación que consiste en utilizar la representación métrica del entono (mapa métrico) para generar planes métricos que permitan al robot moverse entre los segmentos que componen al plan topológico. En este acercamiento se hace un único plan topológico sobre todo el entorno y varios planes métricos sobre porciones reducidas del entorno, lo cual resulta en una reducción del costo computacional en comparación a hacer un único plan métrico sobre el entorno entero.

Los tres trabajos antes mencionados utilizan una misma estrategia para lograr construir un mapa topológico y uno métrico, asegurando que estos son congruentes entre sí. La estrategia consiste en construir una grilla de ocupación a partir de los datos obtenidos por los sensores de los robots. Dicha grilla de ocupación es utilizada tanto como mapa métrico, como para generar una estructura intermedia llamada *grafo generalizado de Voronoi*, con el cual es posible detectar los segmentos que constituyen un mapa topológico basado en regiones. Este proceso es una parte fundamental del trabajo desarrollado en este proyecto y se detallará a lo largo de las siguientes secciones.

2.2.1. Mapa de carreteras

Un *mapa de carreteras* (del inglés *roadmap*) [Choset et al., 2005] es una clase particular de mapas topológicos. Que un mapa topológico cumpla con las condiciones necesarias para ser un *roadmap* asegura que a partir de este es posible planificar un camino entre dos puntos cualquiera del espacio libre. La idea de *roadmap* está inspirada en cómo los humanos utilizamos las redes de carreteras. Al planificar un camino entre dos puntos distantes, por ejemplo ubicados en dos departamentos distintos de Uruguay: Montevideo y Paysandú, en lugar de considerar cada camino posible, normalmente planificamos primero un camino hacia una red de carreteras (Ruta 1), luego a lo largo de la red de carreteras (a través de ruta 1 y ruta 3), para finalmente planificar un camino desde la red de carreteras (Ruta 3) hacia el destino (ver figura 2.3).

Figura 2.3: Red de carreteras de Uruguay. Con rojo se resalta un camino entre ubicaciones de Montevideo y Paysandú. Extraído de [www.google.com/maps, 2021].

Para ser un *roadmap*, un mapa topológico debe estar representado con un grafo $RM = (V_{RM}, E_{RM})$ en el cual sus vértices V_{RM} corresponden a puntos libres del entorno y sus aristas E_{RM} implican adyacencia entre los vértices. Adicionalmente, dicho grafo RM debe cumplir con 3 propiedades fundamentales. Sea W el entorno y $W_{libre} \subseteq W$ el espacio libre del entorno, las tres propiedades son:

1. Accesibilidad: Para todo $p_{inicio} \in W_{libre}$ existe un $p'_{inicio} \in V_{RM}$ tal que existe un camino de p_{inicio} a p'_{inicio}.
2. Conectividad: Para todo par p'_{inicio}, $p'_{fin} \in V_{RM}$ existe un camino en RM entre p'_{inicio} y p'_{fin}.
3. Capacidad de salida: Para todo $p_{fin} \in W_{libre}$ existe un $p'_{fin} \in V_{RM}$ tal que existe un camino de p'_{fin} a p_{fin}.

Estas propiedades aseguran que es posible planificar un camino entre dos puntos cualquiera p_{inicio}, $p_{fin} \in W_{libre}$ a partir del *roadmap*. Primero por *accesibilidad* se tiene que existe un camino entre p_{inicio} y un $p'_{inicio} \in V_{RM}$. Luego por *capacidad de salida* se asegura la existencia de un $p'_{fin} \in V_{RM}$ desde el cual hay un camino a p_{fin}. Finalmente por *conectividad* se sabe que existe un camino entre p'_{inicio} y p'_{fin}. Entonces uniendo los caminos $p_{inicio} \to p'_{inicio}$ (por accesibilidad), $p'_{inicio} \to p'_{fin}$ (por conectividad) y $p'_{fin} \to p_{fin}$(por capacidad de salida) se obtiene un camino completo entre p_{inicio} y p_{fin}.

2.2.2. Diagrama de Voronoi

Dado un espacio euclídeo $W \subseteq \mathbb{R}^2$, un conjunto de puntos $G \subseteq W$ llamados *generadores* y la función $d_p : W \to \mathbb{R}$ que computa la distancia desde $p \in W$ a otro punto en W.

Los *puntos base* de p son el conjunto $BP_p \subseteq G$ de los generadores más cercanos a un punto $p \in W$:

$$BP_p = \{g_i \in G : d_p(g_i) \leq d_p(g_j), \forall g_j \in G, g_j \neq g_i\} \tag{2.1}$$

Un *diagrama de Voronoi* se define como el conjunto $VD \subseteq W$ de los puntos de W que tienen dos o más generadores a una misma mínima distancia, o lo que es equivalente, que tienen dos o más puntos base:

$$VD = \{p \in W : |BP_p| \geq 2\} \tag{2.2}$$

En la figura 2.4 se muestra un ejemplo de un diagrama de Voronoi, con $G = \{A_i | 1 \leq i \leq 10\}$ como generadores.

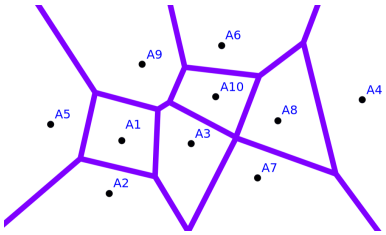


Figura 2.4: Diagrama de Voronoi. Con negro se indican los generadores, mientras que con violeta se indican los puntos pertenecientes al diagrama de Voronoi. Generado con [Jiménez, 2021].

Una propiedad de los diagramas de Voronoi es que a lo largo de los caminos generados sobre estos se mantiene la mayor distancia posible a los generadores más cercanos. Esta propiedad es útil en el contexto de la robótica porque dado un diagrama de Voronoi generado con el conjunto de obstáculos como generadores, los caminos sobre este se consideran caminos de máxima seguridad en tanto mantienen al robot lo más alejado posible de los obstáculos durante el recorrido.

2.2.3. Diagrama generalizado de Voronoi

Recordando que los generadores en un diagrama de Voronoi son puntos, se tiene un problema al querer utilizar obstáculos como generadores. El problema radica en que normalmente los obstáculos ocupan un área y por lo tanto no pueden representarse como puntos en el espacio, para solucionar esto se generaliza la definición de diagrama de Voronoi para poder definirlo a partir de generadores de mayor orden, como lo son líneas o polígonos. A esta generalización se la denomina como *diagrama generalizado de Voronoi* $GVD \subseteq W$, y los generadores de mayor orden o generadores generalizados son representados como conjuntos de puntos del espacio de trabajo, siendo $GG \subseteq P(W)$ el conjunto de todos los generadores generalizados.

Las definiciones relacionadas a los GVD consisten en generalizaciones de las planteadas para los VD en la sección 2.2.2 que consideran generadores generalizados en lugar de generadores puntuales.

En primer lugar se define la función de distancia generalizada $gd_p : P(W) \to \mathbb{R}$ que basándose en distancia euclídea d_p permite obtener la distancia entre un punto $p \in W$ y un generador generalizado g_i. La definición de gd_p se encuentra en (2.3).

$$gd_p(g_i) = \min_{p' \in g_i} \{d_p(p')\} \tag{2.3}$$

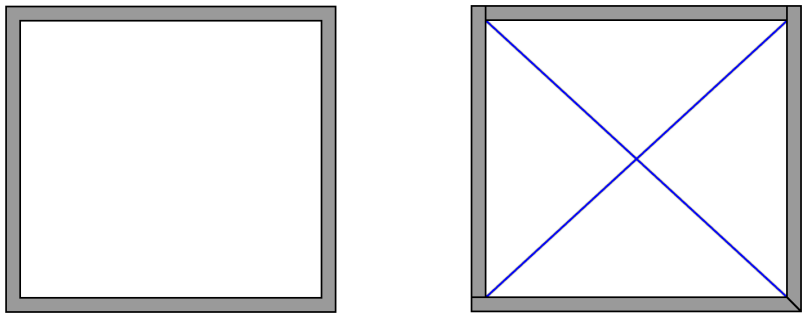
A partir de la distancia generalizada se define a los generadores base generalizados como el conjunto $GBG_p \subseteq GG$ de los generadores generalizados más cercanos a un punto $p \in W$, como se muestra en (2.4). Notar que GBG_p es la generalización de los puntos base BP_p definidos para los VD.

$$GBG_p = \{g_i \in GG : dg_p(g_i) \leq dg_p(g_j), \forall g_j \in GG, g_j \neq g_i\} \tag{2.4}$$

Finalmente la definición de GVD es análoga a la de su contra parte no generalizada y se encuentra en (2.5).

$$GVD = \{p \in W : |GBG_p| \geq 2\} \tag{2.5}$$

Algo a destacar sobre los GVD es que la distribución de los puntos sobre los generadores generalizados no es un tema trivial. Volviendo al caso en el se desea utilizar los obstáculos como generadores, el problema en especifico es el de como distribuir los puntos obstaculizados en generadores generalizados. Una distribución razonable es la resultante de utilizar los conjuntos conexos de puntos obstaculizados como generadores. Esta decisión lleva a resultados que pueden no ser deseados, por ejemplo dentro de una habitación vacía no va a existir ningún punto perteneciente al GVD (figura 2.5a). Esto se debe a que las cuatro paredes conforman un único obstáculo y por lo tanto para cualquier punto p dentro de la habitación se tiene que $|GBG_p| = 1$. El resultado se repite para cualquier generador cóncavo que no contenga otros generadores en su interior, como es el caso de las habitaciones de una sola entrada, y de los corredores sin salida. De esta última observación se deriva el criterio presentado en [Choset et al., 2005] que consiste en utilizar como generadores a conjuntos que además de ser conexos sean convexos, evitando así generadores cóncavos. En la figura 2.5b se muestra un ejemplo de distribución de puntos obstaculizados en generadores conexos y convexos junto al GVD generado en el interior de la habitación.



(a) Generadores conexos

(b) Generadores conexos y convexos.

Figura 2.5: Distribución de generadores. Los generadores se indican con polígonos de interior gris delimitados con aristas negras mientras que el GVD se indica con azul.

En la figura 2.6 se muestra un ejemplo de un GVD que es posible obtener utilizando conjuntos conexos y convexos de puntos obstaculizados como generadores.

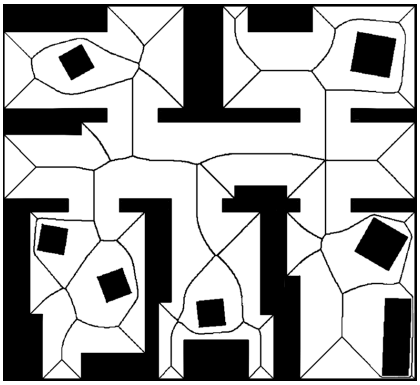
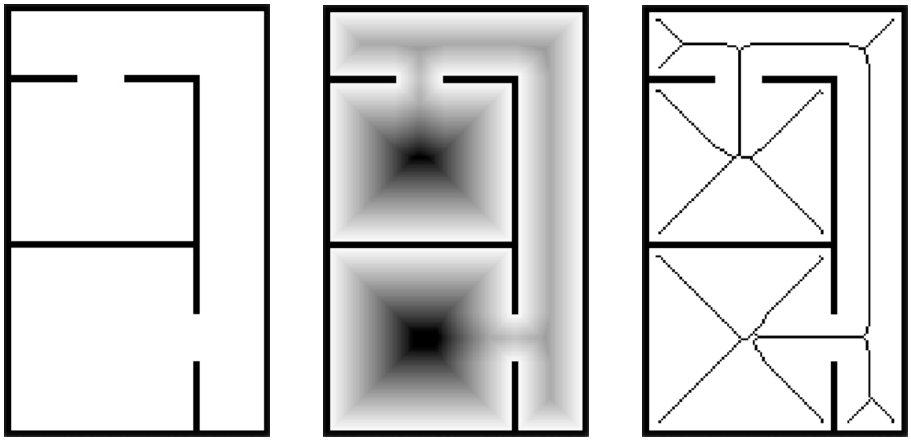


Figura 2.6: Diagrama generalizado de Voronoi. El espacio libre se marca con blanco, las líneas negras identifican al GVD y los polígonos negros representan obstáculos. Extraído de [Wallgrün, 2005]

En [Choset et al., 2005] se demuestra que los GVD que utilizan conjuntos conexos y convexos de puntos obstaculizados como generadores cumplen con las 3 propiedades necesarias para ser *roadmaps* (sección 2.2.1), lo cual justifica su uso para la planificación de caminos. Esto junto a su capacidad de generar caminos seguros ya mencionada y a otras características como su utilidad para generar mapas topológicos basados en regiones que se tratará más adelante, hacen que estos GVD sean estructuras de útilidad en el contexto de la robótica. De ahora en adelante al hablar de GVD se hace referencia a GVD generado usando los conjuntos conexos y convexos de puntos obstaculizados como generadores.

2.2.4. Construcción de un GVD

En [Wurm et al., 2008] se propone un método para construir un GVD a partir de una grilla donde cada celda está ocupada o libre, que es posible obtener a partir de una grilla de ocupación. El método comienza aplicando una transformación de distancia Euclidiana [Meijster et al., 2002] a dicha grilla, resultando en una grilla que contiene para cada celda la distancia al obstáculo más cercano, este tipo de mapa se denomina *mapa de distancia* a los obstáculos. Finalmente el GVD se obtiene aplicando un algoritmo de esqueletización [Zhang and Suen, 1984] al mapa de distancia construido. Un ejemplo de este proceso se puede ver en la figura 2.7. En esta propuesta es clave el uso del algoritmo de esqueletización, que retrae el mapa de distancias a un esqueleto discreto que se aproxima al GVD que se generaría en el espacio continuo.



(a) Mapa original

(b) Mapa original y mapa de distancia (oscuridad proporcional a la distancia al obstáculo más cercano)

(c) Mapa original y GVD (resultante de la esqueletización)

Figura 2.7: Método de construcción de un GVD. Extraído de [Wurm et al., 2008].

Otro método es el presentado en [Choset et al., 2005] llamado *brushfire* que permite construir un GVD mientras se determina un mapa de distancia a los obstáculos. Este algoritmo se puede ver como una variante con multiples fuentes y sin destino del conocido algoritmo de Dijkstra, en el que las fuentes son los obstáculos y la distancia calculada es la mínima distancia hacia un obstáculo. Se comienza inicializando las distancias de las celdas libres con ∞ y las ocupadas con 0. Adicionalmente, las celdas ocupadas se insertan en una cola de prioridad que prioriza las celdas de menor distancia. Luego hasta que las cola de prioridad esté vacía se remueve la celda más prioritaria c y para cada celda c' adyacente a c se computa una distancia tentativa $dt_{c'}$ según (2.6) que remplaza la distancia actual $d_{c'}$ de la celda c' si $dt_{c'} < d_{c'}$. De reemplazarse $d_{c'}$ se inserta a c' en la cola de prioridad.

$$dt_{c'} = d_c + ||c' - c||_2 \tag{2.6}$$

Una vez que la cola de prioridad queda vacía, el valor de distancia de las celdas converge al valor de su mínima distancia hacia un obstáculo, obteniéndose así el mapa de distancia a los obstáculos.

El nombre de este algoritmo proviene de como este determina las distancias comenzando desde cada obstáculo y extendiéndose desde las celdas más cercanas a las más lejanas asemejándose a un incendio de maleza (traducido al inglés como brushfire). Adicionalmente el algoritmo también puede interpretarse como ondas que parten desde cada obstáculo, y a medida de que avanzan le asignan la distancia recorrida a las celdas que atraviesan. Un ejemplo de ejecución se muestra en la figura 2.8.

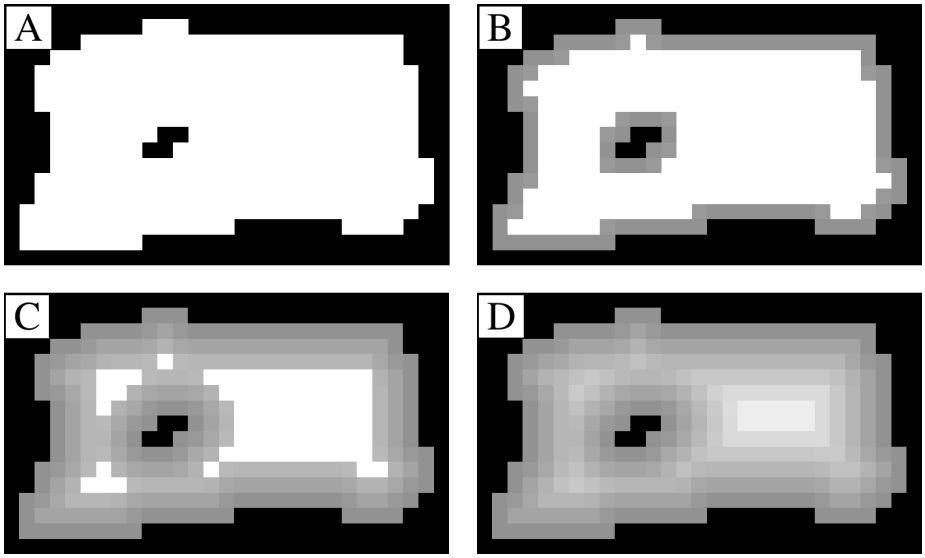


Figura 2.8: Algoritmo brushfire. (A) Los valores de distancia se inicializan con 0 (negro) para celdas ocupadas e infinito (blanco) para celdas vacías. (B) y (C) Las celdas ocupadas generan ondas que asignan distancias crecientes a medida que se propagan, indicadas por el aumento de brillo de las celdas. (D) Las ondas se detienen al no poderse reducir más valores de distancia. Una vez que se han detenido todas las ondas, se obtiene el mapa de distancia completo. Extraído de [Lau et al., 2013].

La ventaja sobre el método propuesto en [Wurm et al., 2008] es que brushfire logra construir el GVD evitando el paso intermedio de esqueletización. La idea se basa en asignar identificadores a cada conjunto conexo de celdas ocupadas, y luego según la interpretación de brushfire como ondas, las ondas provenientes de celdas ocupadas con identificadores distintos colisionan en las celdas que deben pertenecer al GVD. Una colisión de ondas involucra dos celdas c y c', y ocurre cuando una celda c recién extraída de la cola de prioridad no reemplaza la distancia $d_{c'}$ de una celda c' adyacente a c, con la distancia tentativa $dt_{c'}$ calculada según (2.6), por ser $dt_{c'} \geq d_{c'}$. Notar que dos ondas pueden generar multiples choques de ondas.

Esta asignación de identificadores causa los problemas asociados a generadores cóncavos comentados en la sección 2.2.3. En la sección 2.2.5 se describe una solución.

2.2.5. Construcción incremental de un GVD

A lo largo de la exploración el mapa construido recibe actualizaciones de forma frecuente que solo modifican una pequeña porción del mismo. Esto causa que los datos que tienen que ser consistentes con el mapa también deban actualizarse de forma frecuente. Dado esto, por motivos de eficiencia, se busca que los algoritmos que deben ejecutarse para mantener la consistencia de los datos con el mapa, eviten descartar el resultado anterior y recomputarse completamente en cada actualización, sino que en su lugar modifiquen el último resultado para que este sea consistente con la última versión del mapa.

La idea se basa en que, como se mencionó anteriormente, las actualizaciones son frecuentes pero también pequeñas y por lo tanto luego de una actualización gran parte del mapa se mantiene igual. Esto usualmente implica que gran parte del resultado anterior continúa siendo valido luego de la actualización, solo siendo necesario modificar una pequeña porción del mismo para que este vuelva

a ser consistente, lo cual suele ser órdenes de magnitud menos costoso que la alternativa de calcular todo desde cero.

Los algoritmos que actualizan el resultado anterior a partir de los cambios experimentados son llamados *algoritmos incrementales*, por otro lado los algoritmos que al cambiar parte de la entrada descartan el último resultado y recomputan el nuevo resultado desde cero se denominan como *algoritmos no incrementales*.

El uso de algoritmos no incrementales como los presentados en la sección 2.2.4 no es adecuado cuando se desea mantener un GVD consistente con un entorno que es actualizado en tiempo real mientras es explorado, para esto lo ideal es utilizar algoritmos incrementales. A continuación se describe el algoritmo incremental introducido en [Kalra et al., 2009] denominado *brushfire dinámico*, que como su nombre lo indica se basa en el algoritmo no incremental brushfire comentado en la sección 2.2.4.

Como se comento anteriormente, los algoritmos incrementales se basan en mantener un resultado consistente con una entrada que sufre modificaciones a lo largo del tiempo, solo actualizando las porciones del resultado que quedan inconsistentes debido a dichas modificaciones. En el contexto de brushfire dinámico los cambios en la entrada corresponden a celdas cuyo estado pasa de ocupado a libre (se quita una celda ocupada) o viceversa (se agrega una celda ocupada), mientras que las porciones a actualizar son las celdas en las que la distancia mínima a un obstáculo deja de ser consistente debido a los cambios. Específicamente, cuando se quita una celda ocupada, es necesario actualizar aquellas celdas que definen su mínima distancia a un obstáculo según la celda que ahora se encuentra libre. Por otro lado cuando se agrega una celda ocupada, se deben actualizar las celdas cuya distancia a la nueva celda ocupada es menor que la distancia mínima a un obstáculo que tenían antes de la modificación.

Brushfire dinámico lleva a cabo las actualizaciones propagando los cambios ocurridos en forma de ondas (sección 2.2.4). Las celdas o que pasan a estar ocupadas emiten una onda "consistente" que establece la distancia dt_c recorrida por la onda a la distancia d_c asociada a la celda c si $dt_c < d_c$, si la distancia se modifica, entonces se establece a o como el obstáculo a mínima distancia de c. En el caso de las celdas l que pasan a estar libres, se emana una onda "inconsistente" que reinicia a ∞ las distancias d_c de las celdas c que tienen a l como obstáculo a mínima distancia, esta onda se expande hasta llegar a alguna celda c' que tiene un obstáculo valido a mínima distancia. Al suceder esto desde c' se resume la onda "consciente" que causo su valor de distancia actual, y esta onda es la que establece la distancia de las celdas previamente reiniciadas. En la figura 2.9 se muestra gráficamente el funcionamiento de brushfire dinámico.

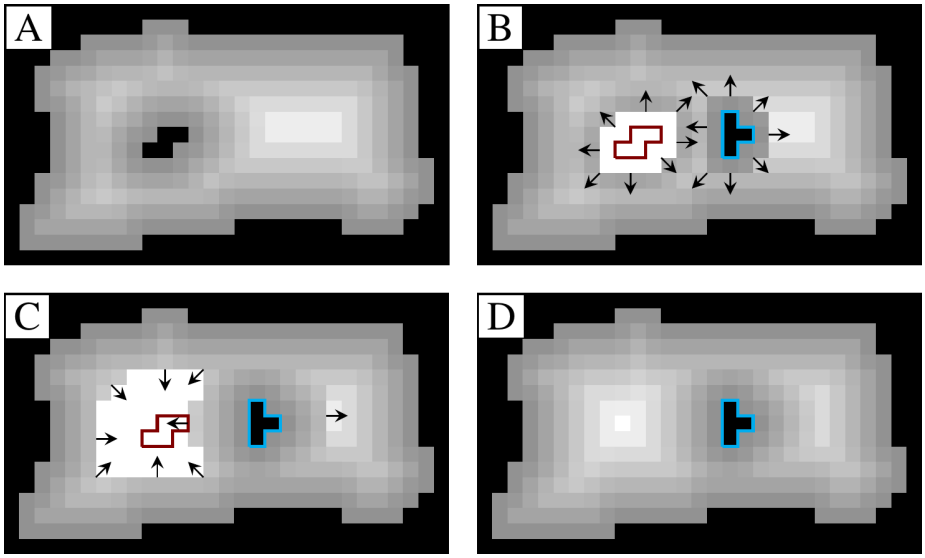


Figura 2.9: Algoritmo brushfire dinámico. (A) Último mapa de distancias calculado. (B) Un obstáculo es eliminado (contorno rojo) lo que causa ondas inconsistentes que comienzan a reiniciar los valores de distancia inválidos, a su vez un obstáculo es insertado (contorno azul) generando ondas consistentes que propagan las nuevas distancias mínimas. (C) Cuando las ondas inconsistentes chocan contra celdas que tienen un obstáculo valido a mínima distancia estas se detienen y en ese mismo lugar inicia una onda consistente para restaurar los valores de distancia reiniciados. (D) Después que todas las ondas se detienen, la actualización se completa. Extraído de [Lau et al., 2013].

Al igual que en el algoritmo brushfire original, en brushfire dinámico la construcción de un GVD se hace a partir de identificadores asignados a conjuntos conexos de celdas ocupadas. Las ondas consistentes en su expansión remueven las celdas del GVD, y al chocar agregan al GVD tanto las celdas en donde se detienen, como también las celdas que causaron el choque, siempre y cuando dichas celdas tengan identificadores distintos entre sí. Por otro lado las ondas inconsistentes remueven del GVD a todas las celdas cuya pertenencia al GVD involucra a celdas previamente ocupadas que pasaron a ser libres. Un ejemplo de como agregar y remover un obstáculo altera al GVD se encuentra en la figura 2.10.

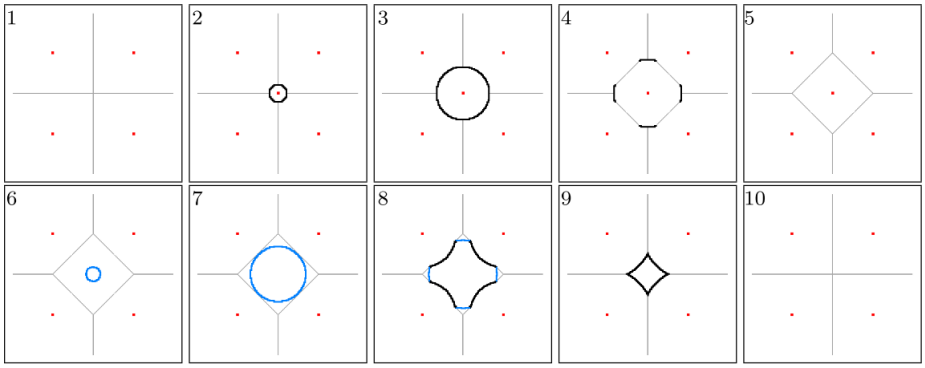


Figura 2.10: Una secuencia que ejemplifica las ondas generadas cuando se agregan obstáculos (1-5) y se eliminan obstáculos (6-10). (1-5) Se agrega nueva celda ocupada en el centro del entorno (en rojo) lo cual produce una onda consistente (en negro) que al chocar genera celdas pertenecientes al GVD (en gris). (6-10) La celda central deja de ser un obstáculo produciéndose una onda inconsistente (en azul) que remueve las celdas del GVD que involucran a la celda ocupada que paso a ser libre y al llegar a celdas con un obstáculo valido a mínima distancia generan ondas consistentes (en negro) que reparan el GVD. Extraído de [Kalra et al., 2009].

De esta forma se logra la construcción incremental de un GVD. Sin embargo en [Lau et al., 2013] se destacan algunas problemáticas de brushfire dinámico, principalmente que el GVD generado tiene líneas de dos celdas de ancho y que no se genera GVD en el interior de obstáculos cóncavos como lo son habitaciones con una sola entrada, lo cual es crítico en entornos estructurados. Los problemas mencionados se pueden ver a la izquierda de la figura 2.11.

Para solucionar las problemáticas planteadas, Lau *et al.* proponen cambios al algoritmo de brushfire dinámico. El problema de generación de GVD dentro de obstáculos cóncavos se relaciona con el uso de conjuntos conexos de obstáculos para determinar la pertenencia de celdas al GVD (sección 2.2.3). Dado esto la solución propuesta consiste en dejar de utilizar identificadores para los conjuntos conexos de obstáculos y en su lugar utilizar un identificador para cada celda ocupada. Al chocar dos ondas, las celdas en donde se da el choque pertenecen al GVD si sus identificadores de obstáculo a mínima distancia son distintos y las celdas ocupadas correspondientes a dichos identificadores no son adyacentes en la grilla. Este cambio soluciona la generación del GVD en obstáculos cóncavos. Con respecto a las líneas de dos celdas de ancho la propuesta consiste en aplicar técnicas que permitan eliminar las celdas redundantes, estas son las celdas del GVD que son adyacentes a más de una celda perteneciente al GVD (no son hojas) y que su remoción no desconecta al GVD. A la derecha de la figura 2.11 se muestra el resultado de aplicar estos cambios.

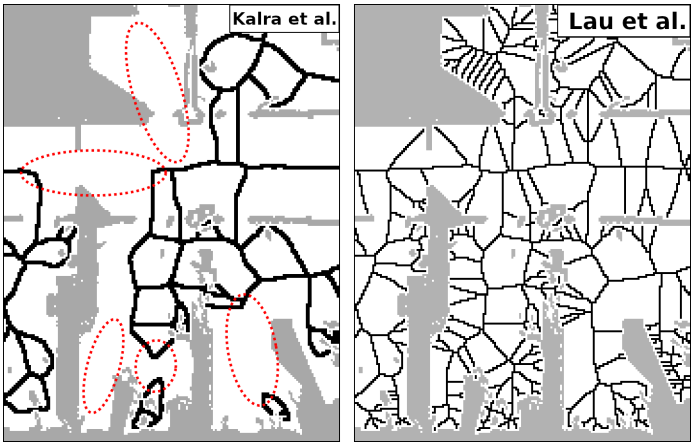


Figura 2.11: Resultados de brushfire dinámico original (a la izquierda) y la variante propuesta por Lau *et al.* (a la derecha). Las elipses marcan los lugares en donde el acercamiento de Kalra *et al.* falla al no generar celdas pertenecientes al GVD. Extraído de [Lau et al., 2013].

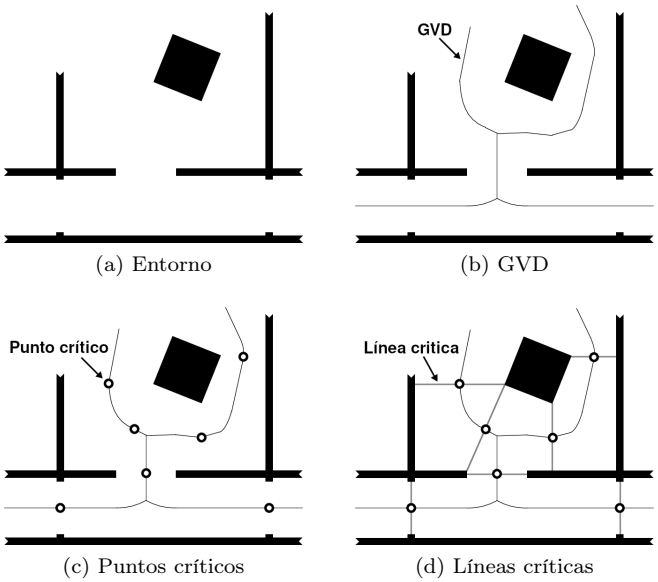
2.2.6. Generación de mapas topológicos basados en regiones a partir de un GVD

En [Thrun, 1998] se propone un método que a partir de un GVD permite segmentar el entorno en habitaciones y corredores, que se corresponden con las regiones de interés de un mapa topológico basado en regiones. La idea principal del algoritmo de segmentación es que las transiciones entre los segmentos son pasajes estrechos, y por lo tanto encontrar estos pasajes estrechos permite delimitar dichos segmentos.

Para describir el método de segmentación es necesario introducir tres nuevos conceptos, el primero es el concepto de *despeje* (del inglés clearance) $D : W \to \mathbb{R}$ de un punto $p \in W$, que corresponde a la distancia del punto p a su punto ocupado más cercano. El segundo es el concepto de *puntos críticos* $Cr \subseteq GVD$, que son los puntos pertenecientes al GVD que son mínimos locales estrictos según el despeje en el GVD, esta definición se formaliza en (2.7), donde $d_p(p')$ es la distancia entre p y p'.

$$Cr = \{p \ \in GVD : \exists \varepsilon > 0, \forall p' \in GVD, d_p(p') \leq \varepsilon, p' \neq p, D(p) < D(p')\} \quad (2.7)$$

El tercer y último concepto es el de *líneas críticas*, que son las líneas que se encuentran entre un punto crítico $p \in Cr$ y sus puntos ocupados más cercanos. La figura 2.12 ilustra estos conceptos.

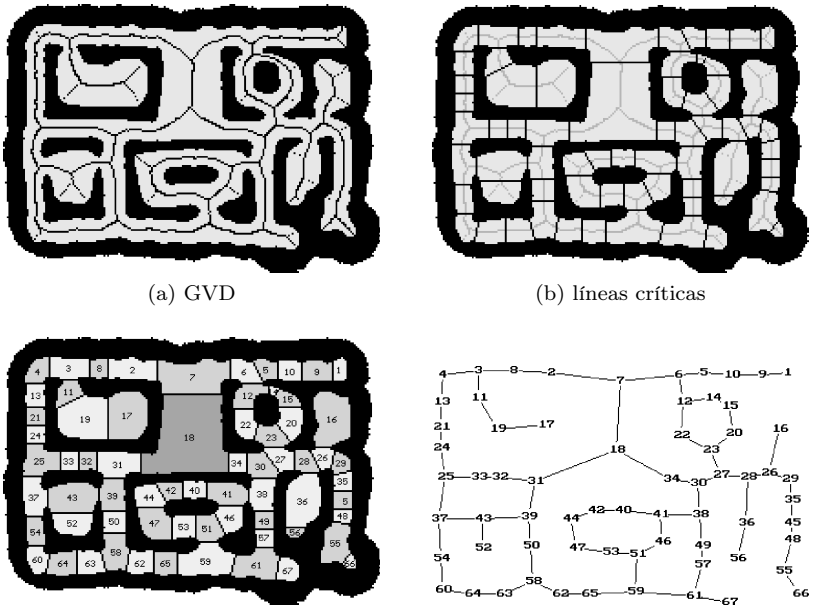


(a) Entorno (b) GVD

(c) Puntos críticos (d) Líneas críticas

Figura 2.12: Puntos críticos y líneas críticas. Notar que el despeje en puntos críticos se corresponde con el largo de sus líneas críticas asociadas. Extraído de [Thrun, 1998].

El algoritmo consiste en determinar los puntos críticos y sus líneas críticas asociadas. Para luego identificar los segmentos como las regiones de espacio libre delimitadas por las líneas críticas y los obstáculos. En la figura 2.13 se muestra un ejemplo de algunas etapas del algoritmo aplicado a un mismo entorno y el mapa topológico construido a partir de los segmentos identificados.



(a) GVD (b) líneas críticas

(c) Segmentos (d) Mapa topologico

Figura 2.13: Método de construcción de un mapa topológico basado en regiones a partir de un GVD. Los números de las figuras (c) y (d) son identificadores arbitrarios de regiones. Extraído de [Thrun, 1998].

La idea subyacente de este algoritmo es que por definición los puntos críticos se encuentran presentes en el centro de todo pasaje estrecho, siendo sus puntos obstaculizados más cercanos los extremos de dicho pasaje. Por lo tanto las líneas críticas estarán ubicadas a lo largo de cada pasaje estrecho y dado que las transiciones entre los segmentos se dan en los pasajes estrechos, las líneas críticas actúan naturalmente como limites entre cada segmento.

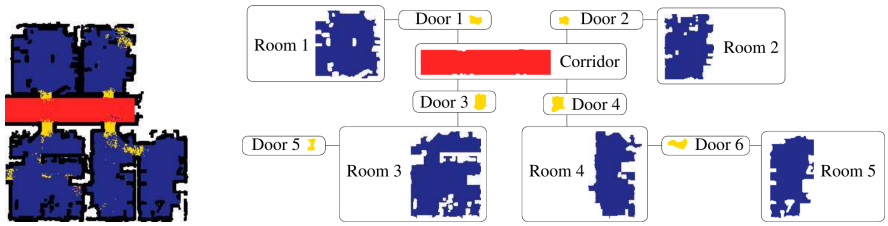
2.2.7. Alternativas para la generación de mapas topológicos basados en regiones

Existen alternativas para generar mapas topológicos basados en regiones, que no involucran el uso de un GVD.

En [Fermin-Leon et al., 2017] la generación se logra interpretando la grilla de ocupación como una imagen binaria, para luego extraer sus contornos los cuales se representan como polígonos. Finalmente se aplica un algoritmo que segmenta el espacio según la convexidad de dichos contornos, obteniendo así las regiones de un mapa topológico basado en regiones.

Por otro lado en [Zivkovic et al., 2006] se interpreta la grilla de ocupación como un grafo donde cada celda libre es un nodo y las aristas se encuentran entre nodos cuyas celdas asociadas son adyacentes en la grilla. Usando un método de particionamiento de grafos se obtienen agrupaciones de nodos que corresponden a las regiones de un mapa topológico.

En [Mart et al., 2006] se presenta un método de generación basado en aprendizaje automático. En cada celda libre de la grilla de ocupación se determina el valor de atributos geométricos asociados a la distancia de la celda a los obstáculos que tiene a su alrededor. Estos atributos se utilizan para entrenar un modelo de clasificación denominado *AdaBoost* [Schapire and Singer, 1999] que es usado para clasificar celdas libres en categorías semánticas como por ejemplo puerta, habitación o corredor. Luego en la práctica a partir de los valores de los atributos y el modelo entrenado se clasifica cada celda libre. Las regiones que conforman el mapa topológico se obtienen agrupando las celdas con una misma categoría. La figura 2.14 muestra un ejemplo de la clasificación y su posterior segmentación.



(a) Clasificación (b) Mapa topológico

Figura 2.14: Método de construcción de un mapa topológico basado en regiones a partir de aprendizaje automático. Las celdas tienen un color correspondiente a su clasificación, amarillo corresponde a puerta, azul a habitación y rojo a corredores. Extraído de [Mart et al., 2006].

2.2.8. Criterios de calidad de un mapa topológico basado en regiones

Criterio de calidad basado en intuiciones geométricas

En [Liu et al., 2015] se plantea un criterio para cuantificar la calidad de un mapa topológico basado en regiones mediante indicadores que surgen de intuiciones geométricas. El criterio considera tanto la calidad de cada región de forma independiente, como la calidad de la segmentación formada por todas las regiones en su conjunto.

La **calidad de una región** se calcula tomando en cuenta dos indicadores la convexidad y la compacidad.

La *convexidad* indica la similitud entre la región y su envolvente convexa. Se calcula como el cociente entre el área de la región y el área de su envolvente convexa.

La *compacidad* mide la distribución de una región con respecto a su centro de masa. A menor valor de compacidad mayor es la concentración de una región sobre su centro de masa.

La **calidad de segmentación** se compone de varios indicadores de calidad sobre ciertas características globales de la segmentación. Estos indicadores son la cobertura, validez y simplicidad.

La *cobertura* de una segmentación se define como el cociente entre la suma de las áreas de cada región y el área total del mapa.

La *validez* indica la proporción de regiones validas en la segmentación, siendo valida una región cuando esta es accesible por los robots utilizados.

La *simplicidad* es un valor que busca penalizar a las segmentaciones que sean demasiado complejas o demasiado simples, esto se logra midiendo la diferencia entre la cantidad de regiones de la segmentación generada y la cantidad de regiones esperada para el entorno actual.

Finalmente la **calidad de un mapa topológico basado en regiones** se reduce a combinar aritméticamente la calidad de cada región y los indicadores de calidad de la segmentación.

Criterio de calidad basado en la segmentación humana

En [Bormann et al., 2016] se propone que la calidad de una segmentación corresponde a su semejanza a la segmentación humana y se proporciona un método para medirla.

En el contexto del artículo la segmentación humana se aproxima como la segmentación que un humano genera para dicho entorno. Luego para cuantificar la semejanza se definen dos indicadores, exhaustividad y precision.

La *exhaustividad* se define como el área correspondiente a la mayor superposición entre una región de la segmentación humana y una región de la segmentación evaluada, dividida por el área de la región de la segmentación humana. La exhaustividad es alta si las regiones de la segmentación humana están completamente contenidas en las regiones de la segmentación evaluada.

La *precisión* de manera similar se define como el área correspondiente a la mayor superposición entre una región de la segmentación humana y una región de la segmentación evaluada, dividida por el área de la región de la segmentación evaluada. La precisión es alta si las regiones de la segmentación evaluada están completamente contenidas en las regiones de la segmentación humana.

En la subsegmentación, la exhaustividad es alta y la precisión es baja, mientras que en la sobresegmentación, la precisión es alta y la exhaustividad es baja. Solo si ambas medidas son altas, la segmentación evaluada se ajusta bien a la segmentación humana.

Este criterio es utilizado en [Bormann et al., 2016, Fermin-Leon et al., 2017] para comparar las segmentaciones resultantes de distintos acercamientos para la construcción de mapas topológicos basados en regiones, incluyendo el método descrito en 2.2.6 y algunos de los mencionados en 2.2.7, siendo la segmentación a partir de un GVD la que obtiene mejores resultados en general.

2.3. Coordinación en la exploración multi-robot

Como fue mencionado en la sección 2.1.1, la coordinación es un aspecto crítico en el problema de exploración multi-robot. A continuación se describen estrategias de coordinación que consisten en segmentar el espacio, asignar segmentos a los robots y finalmente distribuir los objetivos de exploración de cada segmento a los robots asignados a él. Las propuestas se dividen en las que segmentan el espacio considerando la topología del mismo y las que segmentan según criterios no topológicos.

2.3.1. Coordinación a partir de segmentaciones topológicas

En [Wurm et al., 2008] se describe y analiza una estrategia de coordinación que tiene en cuenta la estructura del entorno a partir del uso de una segmentación topológica. La estrategia de coordinación consiste en explorar maximizando la distribución de los robots sobre los segmentos (habitaciones y corredores), que componen a un entorno estructurado. Esto se basa en que asignar más de un robot a un mismo segmento en muchos casos puede ser una desventaja ya que este podría ser demasiado pequeño para que un segundo robot acelere su exploración, aunque exista más de un objetivo de exploración en el mismo. Adicionalmente al terminar la exploración de un segmento los robots pueden bloquearse entre sí mientras intentan abandonarlo. Por otro lado los objetivos de exploración de segmentos diferentes suelen ser independientes entre sí. Considerando esto, distribuir los robots sobre los segmentos a explorar debería lograr una reducción del trabajo redundante y de las interferencias entre los mismos, llevando a una reducción del tiempo de exploración.

Implementar esta estrategia de coordinación requiere una segmentación topológica del entorno para determinar los segmentos sobre los cuales distribuir a los robots. Para lograr esto los autores utilizan el método descrito en la sección 2.2.6 que requiere de un GVD y para construir el GVD se hace uso del método explicado en la sección 2.2.4 que aplica una esqueletización de un mapa de distancia a los obstáculos.

Para maximizar la distribución de los robots sobre los segmentos la idea es asignar los objetivos de exploración a los robots asegurando maximizar la distribución de los objetivos asignados sobre los segmentos. Siendo los objetivos, como es usual, las fronteras (sección 2.1) entre las partes desconocidas y conocidas del mapa.

Para asignar los objetivos a los robots, primero los robots son asignados a los segmentos a explorar maximizando la distribución de los robots sobre los segmentos, por ejemplo evitando asignar más de un robot por segmento si la cantidad de segmentos a explorar es mayor que la cantidad de robots. Específicamente, dicha maximización se obtiene cuando los robots están distribuidos de forma uniforme sobre los segmentos, lo que se obtiene si se cumple lo planteado en (2.8) donde S es el conjunto de todos los segmentos que contienen objetivos, y $\#Robots : S \to \mathbb{N}$ es una función que dado un segmento devuelve el número de robots asignados al mismo.

$$\forall s, s' \in S, |\#Robots(s) - \#Robots(s')| <= 1 \tag{2.8}$$

Para resolver la asignación de robots a segmentos se propone el uso del método húngaro [Kuhn, 1955] de una forma particular que permite cumplir con la restricción (2.8). Para ejecutar el método húngaro es necesario calcular los costos C_s^i que se definen como el costo de llegar a la celda frontera más cercana del segmento $s \in S$ con el robot i. A dichos costos se le descuenta un valor constante en caso de que el robot i ya se encuentre en el segmento s. Luego de haber asignado los robots a los segmentos, se ejecuta nuevamente el método húngaro para cada segmento s con el propósito de asignar los objetivos que se encuentran dentro de s a los robots asignados a s.

La estrategia de coordinación explicada en esta sección se resume en el algoritmo 1.

Algoritmo 1: Estrategia Coordinación [Wurm et al., 2008]

```
Determinar segmentos con objetivos S = {s_1, ..., s_n}
Determinar el conjunto de objetivos para cada segmento
para cada robot i hacer
    para cada segmento s ∈ S hacer
        Computar el costo C_s^i
        Descontar un valor constante al costo C_s^i si el robot i ya se
         encontraba en s
    fin
fin
Usando el método húngaro asignar robots a los segmentos
para cada segmento s ∈ S hacer
    Usando el método húngaro asignar los objetivos de s a los robots
     asignados a s
fin
```

2.3.2. Coordinación a partir de segmentaciones no topológicas

Existen estrategias de coordinación que se basan en segmentar el espacio sin considerar su topología, una de estas se propone en [Solanas and Garcia, 2004]. En esta estrategia la segmentación consiste en agrupar el espacio desconocido en tantos grupos como robots se utilizan para explorar. Las agrupaciones se determinan utilizando el algoritmo K-Means [Hartigan and Wong, 1979]. En la figura 2.15 se muestra un ejemplo de esta segmentación con K = 8.

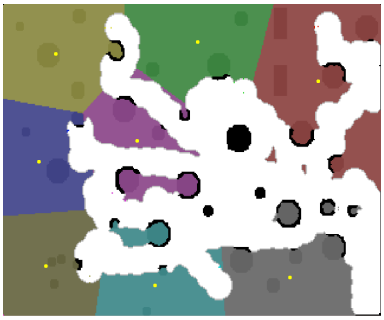


Figura 2.15: Segmentación propuesta en [Solanas and Garcia, 2004]. En blanco se indica el espacio libre, con negro el espacio ocupado, y con colores se indica el espacio desconocido, donde cada color indica un segmento diferente dentro de los cuales se indica con tonalidades más oscuras los obstáculos todavía no explorados y con amarillo su centroide. Extraída de [Wu et al., 2007].

La estrategia de coordinación comienza asignando a cada robot el segmento con el centroide más cercano según la distancia euclidiana. El segmento asignado al robot R_i se identifica como ζ_i. Los objetivos de exploración nuevamente serán las fronteras (sección 2.1), la frontera F_j que se asigna al robot R_i es la que minimiza el $costo_{ij}$ definido en (2.9) donde C_i es el centroide de ζ_i, Δ es una constante que representa la distancia máxima entre dos puntos del mapa (longitud de la diagonal del mapa), $||\ ||_2$ es la norma euclídea, $d(c_1, c_2)$ es la distancia del camino más corto sobre el espacio libre entre dos celdas c_1, c_2. Y o_{ij} es una penalización acumulada que aumenta cuando F_j está en el rango de sensado de otro robot.

$$costo_{ij} = \begin{cases} \Delta + ||F_j - C_i||_2 + o_{ij} & F_j \notin \zeta_i \\ d(F_j, R_i) + o_{ij} & F_j \in \zeta_i \end{cases} \tag{2.9}$$

Dado un robot R_i la función de $costo_{ij}$, a partir del valor Δ penaliza fuertemente a los objetivos F_j que no pertenecen al segmento ζ_i asignado a R_i. Por lo tanto los objetivos pertenecientes al segmento asignado al robot tendrán prioridad frente a los que pertenecen a otro segmento, incentivando que los robots se mantengan explorando sus segmentos asignados. Los objetivos pertenecientes a otros segmentos son considerados para permitir que un robot pueda continuar tomando objetivos aunque su segmento no tenga ningún objetivo disponible. Por otro lado la penalización o_{ij} evita el solapamiento de los rangos de sensado de los robots, evitando así el trabajo redundante.

Otro método es el que se describe en [Lopez-Perez et al., 2018], en este caso la segmentación consiste en asignar cada punto del espacio desconocido a el robot más cercano de la flota, por lo tanto el entorno se particiona en tantos segmentos S_i como robots R_i componen la flota de exploración. La definición de los segmentos se encuentra en (2.10) donde U son los puntos desconocidos del espacio, y la distancia d se define de la misma manera que la distancia homónima del método anterior.

$$S_i = \{c_j \in U : d(R_i, c_j) \leq d(R_k, c_j) \forall k \neq i\} \tag{2.10}$$

Un ejemplo de esta segmentación se muestra en la figura 2.16 para un entorno completamente desconocido, donde se puede visualizar las posiciones de los robots y con colores se indican los distintos segmentos. Notar que por la definición de segmento (2.10) cada robot R_i se encuentra dentro del segmento S_i que fue definido a partir de su ubicación.

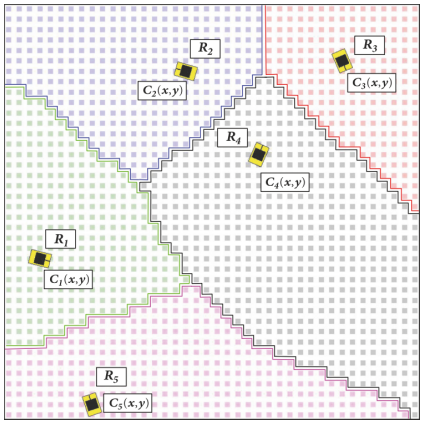


Figura 2.16: Segmentación propuesta en [Lopez-Perez et al., 2018], extraído de la misma fuente.

En este método los objetivos no son fronteras sino que son todo el espacio desconocido. Cada robot R_i solo puede ser asignado a objetivos pertenecientes a su segmento S_i, siendo el objetivo asignado el que alcanza el menor valor en la función de peso f_p que se encuentra definida en (2.11).

$$f_p(c_j) = k_d.d(R_i, c_j) + k_a.\phi_i(c_j) \tag{2.11}$$

Donde k_d y k_a son dos constantes positivas que determinan el peso de cada sumando de la ecuación, $\phi_i(c_j)$ es el ángulo que queda definido entre la orientación del robot y el vector que se origina en R_i y apunta hacia la celda c_j.

2.4. Simuladores

La simulación es una herramienta de gran utilidad en el contexto de la robótica y fundamental en el desarrollo de este proyecto.

Los simuladores robóticos modelan la realidad con el propósito de reproducir computacionalmente los aspectos de esta que son relevantes para la robótica. Específicamente un simulador robótico debe proporcionar un modelo de las interacciones físicas (gravedad, colisiones, iluminación, viento, entre otros) como permitir representaciones realistas de robots incluyendo sus sensores, actuadores y transductores.

La utilidad principal de los simuladores es que permiten realizar pruebas sin necesidad de robots, ni de entornos reales. Esto permite validar de forma rápida y económica soluciones a problemas complejos. Pero no es un reemplazo para pruebas en la realidad, ya que los modelos utilizados realizan simplificaciones que pueden causar cambios con respecto al comportamiento real.

Actualmente existe una gran variedad de simuladores robóticos, de los cuales se consideraron cuatro como candidatos para utilizar en el proyecto, Gazebo[1], CoppeliaSim[2] (antes conocido como V-Rep), ARGoS[3] y Webots[4].

En las siguientes secciones se realiza un análisis comparativo de los simuladores candidatos que concluye con la elección de uno de ellos para su uso en el proyecto.

[1] http://gazebosim.org. Accedido por última vez: 25/02/2022.
[2] http://www.coppeliarobotics.com. Accedido por última vez: 25/02/2022.
[3] http://www.argos-sim.info. Accedido por última vez: 25/02/2022.
[4] http://cyberbotics.com. Accedido por última vez: 25/02/2022.

2.4.1. Características

Existen varios estudios comparativos de simuladores robóticos que incluyen alguno de los candidatos [Nogueira, 2014, Santos Pessoa de Melo et al., 2019, Ramli et al., 2015, Pitonakova et al., 2018]. Se analizó dichos artículos y se estudió la información que se encuentra pública en las páginas oficiales en busca de características consideradas de utilidad, los resultados se resumen en la tabla 2.1.

	Gazebo	CoppeliaSim	Webots	ARGoS
Multiples robots	Si	Si	Si	Si
Simuladores físicos disponibles	Bullet, ODE, Sim-body y DART	Bullet, ODE, Vortex y Newton	ODE	Propios
Integración con ROS	Si	Si	Si	Inactiva
Aceleración de tiempo	Si	Si	Si	Si
Modo headless	Si	Si	No	Si
Licencia	Apache 2.0	Propietaria	Apache 2.0	MIT

Cuadro 2.1: Comparación entre simuladores.

Todos los simuladores candidatos permiten simular varios robots simultáneamente lo cual es esencial en el marco del proyecto.

Webots solo dispone de ODE[5] para la simulación física, mientras que Gazebo y CoppeliaSim disponen de varias alternativas entre las cuales se incluyen ODE y Bullet[6]. Tanto ODE como Bullet son conocidos motores de físicas cuyas capacidades han sido estudiadas en diversos artículos [Bzhikhatlov and Perepelkina, 2017, Erez et al., 2015, Rönnau et al., 2013]. ARGoS por su parte dispone de simuladores 2D y 3D personalizados que según [Pitonakova et al., 2018] son mucho más simples que los utilizados por Gazebo y CoppeliaSim. Esta simplicidad se comenta que está asociada con capacidades limitadas, pero también es posible que sea un factor que influye positivamente en la eficiencia.

ROS [www.ros.org, 2021] es un framework que facilita el desarrollo de aplicaciones robóticas, utilizarlo evita "reinventar la rueda" y permite dedicarse a los aspectos principales de un proyecto. De los simuladores candidatos todos menos ARGoS proveen soporte oficial para ROS. Para ARGoS existe una extension [Vardy, 2021] que permite conectar a ROS con ARGoS que se encuentra inactiva.

La posibilidad de acelerar el tiempo se encuentra presente en todos los simuladores, lo cual es deseable para reducir el tiempo real que lleva ejecutar una simulación, siempre y cuando el hardware que corra las simulaciones sea lo suficientemente potente como para permitirlo.

Ejecutar en modo *headless* es ejecutar sin una interfaz gráfica, esto es útil porque permite dedicar más recursos computacionales a la simulación. Esto se permite en todos los simuladores estudiados, menos en Webots.

Con respecto a las licencias Gazebo, Webots y ARGoS tienen licencias libres [www.fsf.org, 2021], mientras que CoppeliaSim tiene una licencia propietaria.

[5] http://www.ode.org. Accedido por última vez: 25/02/2022.
[6] http://pybullet.org. Accedido por última vez: 25/02/2022.

Aunque la licencia de CoppeliaSim permite su uso no comercial de forma gratuita con propósitos educacionales, es deseable el uso de alternativas libres.

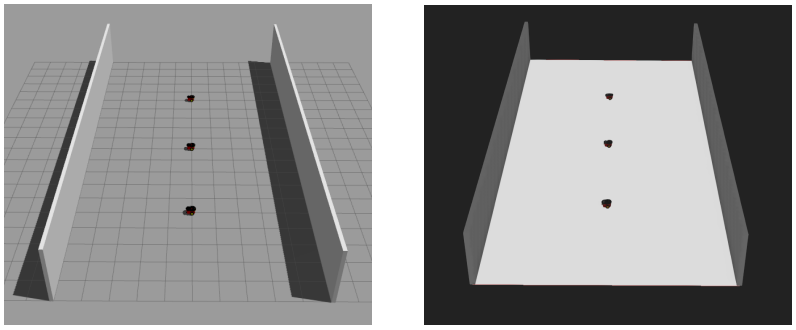
En [Pitonakova et al., 2018] se presenta un estudio comparativo de eficiencia entre CoppeliaSim, Gazebo y ARGoS. El estudio se basa en experimentos que consisten en distintas cantidades de robots (1, 5, 10 y 50) que se mueven en línea recta por un minuto evitando obstáculos en tiempo real. Esto se repitió en dos entornos distintos uno simple y otro complejo. Los experimentos evaluaron el rendimiento (a través de la aceleración de tiempo) y el uso de recursos (RAM y CPU). CoppeliaSim consistentemente presenta los peores resultados en uso de recursos y rendimiento, considerándose inviable la ejecución de las pruebas que utilizan 50 robots. Entre ARGoS y Gazebo en términos de uso de recursos ARGoS resulta superior en todos los casos. Con respecto al rendimiento, en casos simples (pocos robots o entorno simple) ARGoS es mejor, mientras que Gazebo es mejor en casos complejos (muchos robots y entorno complejo).

Dada la información presentada hasta el momento es posible descartar a ARGoS como opción dado que su simulación se considera poco realista y por su falta de integración con ROS. Por su licenciamiento propietario, bajo rendimiento y alta utilización de recursos, se descarta a CoppeliaSim como posibilidad. Entre los restantes Gazebo y Webots la información disponible dificulta tomar una decisión. Se busco sin éxito estudios de rendimiento y/o uso de recursos sobre Webots. Dado esto se decidió realizar un estudio comparativo propio entre Webots y Gazebo, cuyo objetivo es evaluar el rendimiento y usabilidad de ambos.

2.4.2. Experimentación

Método

El experimento se basa en replicar un mismo escenario en Gazebo y Webots que consiste en dos paredes de $20m$ paralelas que están a $10m$ una de la otra. Entremedio de estas se ubican 3 robots diferenciales Pioneer 3-DX [MobileRobots, 2021] uno alineado con el centro de las paredes, uno a la izquierda y otro a la derecha ambos a $5m$ del que se encuentra en el centro. En la figura 2.17 se muestra el escenario de simulación implementado en los simuladores a evaluar.



(a) Gazebo. (b) Webots.

Figura 2.17: Implementaciones del escenario de simulación.

La simulación consiste en que los robots avancen hacia una de las paredes hasta que se encuentran a un metro de esta (información obtenida usando un LiDAR), momento en el que estos cambian de dirección. El comportamiento se repite por cinco minutos.

El proceso de replicar el escenario en cada simulador utilizando ROS será el insumo para evaluar su usabilidad, mientras que para evaluar el rendimiento y

comportamiento se ejecutará una vez ambas simulaciones acelerando el tiempo tanto como sea posible. El valor R_{sim} (2.12) es usado como indicador de rendimiento, este es el resultado de dividir el tiempo ts dentro de la simulación entre el tiempo tr real que toma ejecutarla. A mayor R_{sim} mayor rendimiento.

$$R_{sim} = \frac{ts}{tr} \tag{2.12}$$

El software utilizado fue Ubuntu *20.04*, ROS *Noetic*, Gazebo *11.4.0* y Webots *R2021a*.

El hardware utilizado consiste de un procesador Intel Core i3-9100F, un procesador gráfico GeForce GTX 660 y 16GB de memoria RAM.

Resultados

Las implementaciones del escenario de prueba en Gazebo y Webots se encuentran disponible en línea[7] [8].

La construcción del entorno fue similar en ambos simuladores siendo más cómoda en Webots ya que este dispone por defecto del modelo de robot a utilizar, por otro lado en Gazebo fue necesario usar un modelo creado por un tercero.

En términos de integración con ROS, en Gazebo se logró sin inconvenientes, mientras que Webots requirió más trabajo en tanto no integra facilidades para el control de robots. Por ejemplo, Webots no soporta el uso del paquete de ROS *diff_ drive_ controller* [Magyar, 2021] que simplifica el control de movimiento de robots diferenciales, mientras que Gazebo sí.

Otro aspecto que complejizó la integración de ROS en Webots es que su sistema de coordenadas por defecto (NUE) difiere del utilizado por ROS y Gazebo (ENU). Fue posible configurar Webots para utilizar ENU pero se requirió hacer algunos cambios como rotar los elementos del entorno manualmente.

Al ejecutar las simulaciones en Gazebo el comportamiento fue el esperado, en Webots los robots se balanceaban notoriamente de adelante hacia atrás al moverse y a su vez las medidas provistas por el LiDAR no se corresponden con las ubicaciones de la pared.

Los resultados de rendimiento fueron, para Webots $R_{sim} = 2{,}20$, para Gazebo $R_{sim} = 3{,}58$ en su modo normal y $R_{sim} = 4{,}70$ haciendo uso del modo *headless* que Webots no dispone.

2.4.3. Conclusiones

Con respecto a la usabilidad, Webots presenta la ventaja de tener disponible el modelo del robot a utilizar, pero esto queda opacado por los inconvenientes experimentados al integrar con ROS[9]. Gazebo resultó tener una mejor usabilidad al no presentar dichos inconvenientes.

La simulación en Gazebo no presenta diferencias visibles al comportamiento esperado mientras que en Webots se detecto una diferencia a nivel de actuación y otra a nivel de sensado.

En términos de rendimiento, aunque los resultados no tienen relevancia estadística, dan indicios de que el rendimiento de Gazebo es mejor que el de Webots, principalmente al hacer uso del modo *headless*.

Por estas razones se decidió que el simulador más adecuado para utilizar en el proyecto es Gazebo.

---

[7] http://gitlab.fing.edu.uy/federico.ciuffardi/gazebo_benchmark. Accedido por última vez: 25/02/2022.

[8] http://gitlab.fing.edu.uy/federico.ciuffardi/webots_benchmark. Accedido por última vez: 25/02/2022.

[9] La versión R2021b de Webots agrega soporte para *diff_ drive_ controller*

Capítulo 3

Solución propuesta

Contenidos

3.1. Hipótesis de trabajo 30
3.2. Arquitectura 31
3.2.1. Move Base 31
3.2.2. Combinador de mapas 32
3.2.3. Controlador central 33
3.2.4. Controlador del robot 33
3.2.5. Controlador de movimiento 33
3.3. Definiciones 33
3.3.1. Grillas de ocupación 33
3.3.2. Componentes conexas 34
3.4. Identificación de objetivos 36
3.4.1. Fronteras 36
3.4.2. Fronteras significativas basadas en K-Means 36
3.4.3. Fronteras significativas basadas en cubrimiento . . . 38
3.5. Asignación de objetivos 41
3.5.1. Obtención de información 41
3.5.2. Valuación 42
3.5.3. Resolución 43
3.6. Segmentación 46
3.7. Diagrama generalizado de Voronoi 50
3.7.1. Criterio de pertenencia 50
3.7.2. Consideraciones sobre el espacio desconocido 51
3.8. Planificación 53
3.8.1. Criterios de compleción 54
3.8.2. Planificación jerárquica 55
3.9. Construcción cooperativa del mapa 56

En este capítulo se describen los principales aspectos de la solución al problema de exploración multi-robot desarrollada[1], incluyendo las potenciales mejoras que constituyen las contribuciones del proyecto de grado (sección 1.1).

El capítulo comienza con la sección 3.1 donde se enumeran las hipótesis de trabajo. Luego en la sección 3.2 se describe la arquitectura de la solución. Seguido de esto, en la sección 3.3 se detallan y formalizan conceptos que son fundamentales en la solución propuesta.

Lo que resta del capítulo se dedica a profundizar en diversos aspectos de la solución. A continuación se destacan las secciones donde se tratan las contribuciones del proyecto. En la sección 3.4.3 se describe el método de identificación de objetivos propuesto. En la sección 3.5.3 se introduce el algoritmo de asignación de objetivos. La sección 3.7 se comenta el algoritmo de construcción incremental del GVD y la forma novedosa de tratar las porciones desconocidas del espacio durante la construcción del GVD. Finalmente la sección 3.8 se describe la planificación jerárquica propuesta.

3.1. Hipótesis de trabajo

La solución se desarrolla con las siguientes hipótesis de trabajo.

(I) Cada robot puede obtener en todo momento su ubicación (posición y orientación) sin errores.

(II) Los robots son iguales entre sí.

(III) Las comunicaciones son ideales: sin perdida, con un ancho de banda y rango infinito.

(IV) Los robots tienen la capacidad de detectar las superficies más cercanas que se encuentran dentro de un circulo de radio *rango* centrado en el robot.

(V) El entorno a explorar es cerrado.

(VI) Se conocen las dimensiones del entorno a explorar.

Estas hipótesis tienen el propósito de simplificar algunos aspectos del problema de exploración permitiendo que el trabajo se pueda enfocar en otros.

La hipótesis (I) resuelve trivialmente el problema de localización [Khairuddin et al., 2015].

Las hipótesis (II) y (III) simplifican la coordinación. La (II) por no tener que hacer consideraciones especiales para cada robot, y la (III) por no tener que considerarse los posibles problemas de comunicación que pueden haber de asignar robots a objetivos muy distantes o con obstrucciones entre sí.

La hipótesis (IV) establece la capacidad sensorial del robot. Que sea posible sensar un circulo centrado en el robot, evita considerar su orientación.

La hipótesis (V) es útil para poder aplicar el criterio de parada que consiste en detener la exploración al no tener más objetivos restantes (sección 2.1).

Finalmente la hipótesis (VI) permite establecer una grilla de ocupación con las dimensiones suficientes para representar todo el entorno a explorar.

[1] Disponible en línea:
https://gitlab.fing.edu.uy/federico.ciuffardi/pgmappingcooperativo.
Accedido por última vez: 25/02/2022.

3.2. Arquitectura

La arquitectura se divide en dos tipos de componentes: robots y unidad central. Existe una única unidad central que se encarga de centralizar la información del entorno explorado y de coordinar a los multiples robots que pueden existir. Cada componente está asociado a un hardware específico, siendo los robots móviles y la estación central estática. Los componentes a su vez están compuestos por módulos los cuales cuales se encargan de tareas específicas.

En la figura 3.1 se resume la arquitectura en un diagrama en el cual es posible ver los componentes, sus módulos y cómo estos se distribuyen sobre los componentes. Adicionalmente es posible apreciar una simplificación de las comunicaciones que ocurren entre dichos módulos.

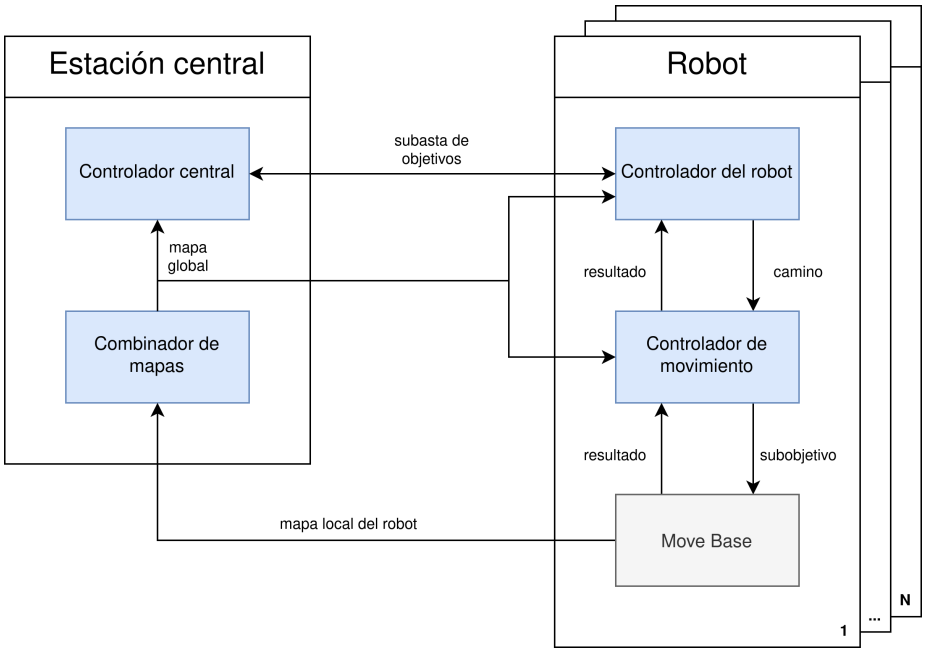


Figura 3.1: Arquitectura de la solución propuesta. Los módulos coloreados en azul fueron implementados en este proyecto.

En lo que resta de esta sección se comentará de cada módulo tanto sus tareas, como sus interacciones con el resto de los módulos.

3.2.1. Move Base

El módulo *Move Base* provee una interfaz para configurar, ejecutar e interactuar con el *stack de navegación* de ROS [Marder-Eppstein, 2021b]. Específicamente el módulo corresponde a un nodo[2] conocido como *move_base* [Marder-Eppstein, 2021a].

El stack de navegación de ROS es un conjunto de nodos que tienen como propósito que un robot pueda navegar el entorno hacia objetivos dentro del mismo. En la figura 3.2 se muestra un diagrama que representa a los nodos que componen al stack de navegación de ROS y sus interacciones.

[2]En ROS el concepto de nodo refiere básicamente a un proceso que utiliza ROS.

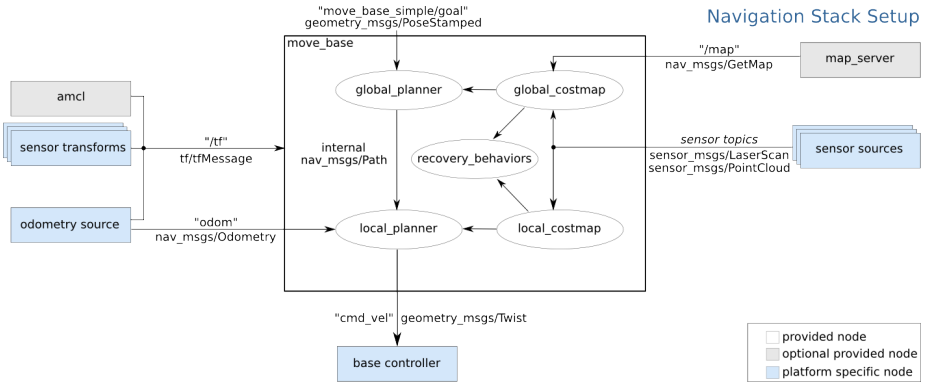


Figura 3.2: Arquitectura del stack de navegación de ROS. Extraída de [Marder-Eppstein, 2021a].

Cuando se establece un objetivo de navegación este se trasmite al nodo *global_planner* que se encarga de generar un plan de alto nivel conocido como plan global, que consiste en un número de subobjetivos que al seguirse en secuencia llevan al objetivo.

Los subobjetivos generados por el *global_planner* son enviados al nodo *local_planner* que se encarga de generar secuencias de velocidades lineales y angulares que un robot debe tener a lo largo del tiempo para llegar sin colisiones a los subobjetivos recibidos. A dicha secuencia de velocidades se la conoce como plan local.

El stack de navegación permite utilizar distintas implementaciones de *global_planner* y *local_planner*. En el trabajo desarrollado se hace uso del *global_planner* que valga la redundancia es llamado *global_planner* [Lu, 2021] y el *local_planner* llamado *teb_local_planner* [Rösmann, 2021].

Para generar sus planes tanto *local_planner* como *global_planner* requieren de un mapa, de esto se encargan los nodos *local_costmap* (mapa local) y *global_costmap* (mapa global) respectivamente. Ambos son nodos de una misma clase llamada *costmap_2d* [Eitan Marder-Eppstein, 2021], que dentro de sus funcionalidades está la de construir una grilla de ocupación a partir de los datos sensoriales provistos por los robots. Las principales diferencias entre el mapa global y el local son su tamaño de celda (pequeño en el local y grande en el global), sus dimensiones del mapa (el mapa global es el mapa completo mientras que el local es solo una porción), y sus marcos de referencia (el mapa global suele estar fijo, el local se centra en el robot).

El nodo *recovery_behaviors* permite ejecutar comportamientos de recuperación de detectarse que el robot no está avanzando de forma correcta al objetivo. Para la solución desarrollada solo se hace uso de un comportamiento de recuperación que consiste en que el robot rote en el lugar.

3.2.2. Combinador de mapas

El módulo *Combinador de mapas* es el encargado de construir el mapa completo del entorno explorado. Este recibe actualizaciones de los mapas globales, que son generadas por el nodo *global_costmap* del stack de navegación de cada robot y las combina en un único mapa. Luego de combinar una actualización, de existir cambios en el mapa completo estos se distribuyen a los diversos componentes del sistema que requieren del mapa completo del entorno explorado para llevar a cabo alguna de sus tareas (ver figura 3.1).

3.2.3. Controlador central

El módulo *Controlador central* es el responsable de orquestar la asignación de objetivos. Específicamente la asignación de objetivos consiste en una subasta y este módulo actúa como el subastador.

La subasta se puede resumir de la siguiente manera, los objetivos de exploración identificados son transmitidos desde la central hacia los robots los cuales valúan a dichos objetivos según que tan conveniente les es llegar a ellos. Luego los robots envían sus valuaciones a la central y esta aplica un algoritmo para determinar qué objetivo le corresponde a cada robot. Finalmente la central le informa a cada robot el objetivo que le corresponde.

3.2.4. Controlador del robot

El *Controlador del robot* es el módulo que se encarga de valuar los objetivos cuando ocurre una subasta. También se encarga de procesar los objetivos asignados, determinando un camino (secuencia de subobjetivos) que lleva al objetivo y enviándolo al módulo *Controlador de movimiento*. Adicionalmente es el responsable de solicitar el inicio de una subasta al *Controlador central* cuando el *Controlador de movimiento* indica el éxito o el fracaso en completar un camino asignado.

3.2.5. Controlador de movimiento

El módulo *Controlador de movimiento* recibe caminos desde el módulo *Controlador del robot* y se encarga de determinar cuales subobjetivos debe enviar al módulo *Move Base* para seguir el camino de forma rápida, evitando maniobras innecesarias. También lleva a cabo una capa superior de comportamientos de recuperación extendiendo los provistos por el módulo *Move Base*. Y finalmente se encarga de indicar al *Controlador del robot* si se completó el camino con éxito, o si existe algún problema.

3.3. Definiciones

Esta sección está dedicada a formalizar conceptos que serán utilizados en lo que resta del capítulo.

3.3.1. Grillas de ocupación

La grillas de ocupación fueron introducidas en la sección 2.2. En esta sección se profundizan conceptos a dichas estructuras que son útiles en el contexto del la solución desarrollada.

El conjunto $CG \subseteq \mathbb{R}^2$ está compuesto por los centros de cada celda de la grilla de ocupación. Una celda se asocia a un único centro y viceversa, por lo tanto en lo que resta de este informe se usaran ambos términos de forma indistinta.

Se dice que cada celda $c \in CG$ tiene asociada una probabilidad $P(c|x_{1:t}, z_{1:t})$ de estar ocupada, donde $z_{1:t}$ es la secuencia pasada de medidas obtenidas de los sensores desde las posiciones $x_{1:t}$.

La función $e : CG \to E$ dado un centro de celda, devuelve uno de los tres estados posibles $E = \{libre, ocupado, desconocido\}$ según la probabilidad de que

c esté ocupada. En el contexto de este proyecto la función e se define según (3.1).

$$e(c) = \begin{cases} \textit{libre} & \text{si} \quad P(c|x_{1:t}, z_{1:t}) < 0{,}5 \\ \textit{desconocido} & \text{si} \quad P(c|x_{1:t}, z_{1:t}) = 0{,}5 \\ \textit{ocupado} & \text{si} \quad P(c|x_{1:t}, z_{1:t}) > 0{,}5 \end{cases} \tag{3.1}$$

Otra decisión tomada en el contexto del proyecto es que la probabilidad de ocupación inicial $P(c)$ de cada celda c se establece como 0,5 indicando que a priori es equiprobable que c se encuentre libre u ocupada. Esto implica que según la función e el estado inicial de todas las celdas es *desconocido*.

La función $ady_4 : CG \to P(CG)$ dada una celda c devuelve el conjunto de celdas horizontalmente adyacentes y la función $ady_8 : CG \to P(CG)$ devuelve el conjunto de celdas adyacentes horizontal y diagonalmente. Las definiciones de ady_4 y ady_8 se presentan en (3.2) y (3.3) respectivamente donde n_2, n_4, n_6, n_8 se corresponden a vecinos horizontales y n_1, n_3, n_5, n_7 se corresponden a los vecinos diagonales de c según se muestran en la figura 3.3.

$$ady_4(c) = \{n_{i*2} : 1 \leq i \leq 4, n_{i*2} \in C\} \tag{3.2}$$

$$ady_8(c) = \{n_i : 1 \leq i \leq 8, n_i \in C\} \tag{3.3}$$

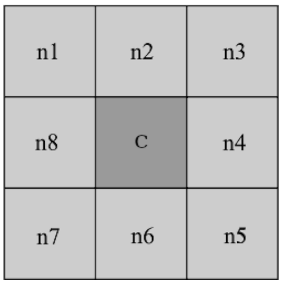


Figura 3.3: Vecinos de una celda en una grilla de ocupación.

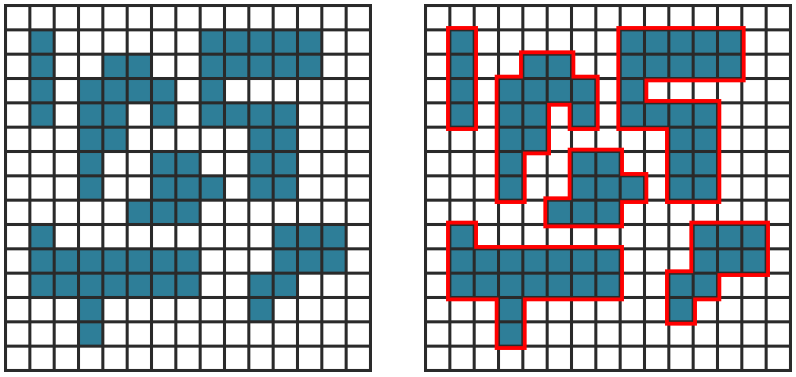
Desde este punto se utilizará la función ady como una función de adyacencia genérica que puede ser tanto ady_4 como ady_8 a no ser que se indique lo contrario. Notar que la relación de adyacencia es simétrica por lo que $c_1 \in ady(c_2) \Leftrightarrow c_2 \in ady(c_1)$.

3.3.2. Componentes conexas

Una descomposición en componentes conexas de un conjunto de celdas $C \subseteq CG$ es un conjunto $CC \in P(C)$ compuesto por N conjuntos C_i con $i \in [1, N]$ tales que:

- $\bigcup_{i=1}^{N} C_i = C$
- Para todo $i, j \in [1, N]$ con $i \neq j$ se cumple que $C_i \cap C_j = \emptyset$
- Para todo $i \in [1, N]$, para todo $c_1, c_2 \in C_i$ se cumple que existe un camino compuesto de celdas en C_i que llevan desde c_1 a c_2.
- No existen $i, j \in [1, N]$ con $i \neq j$ tales que existan $c_1 \in C_i$ y $c_2 \in C_j$ que cumplan con $c_1 \in ady(c_2)$

Un ejemplo de una descomposición en componentes conexas se muestra en la figura 3.5b.



(a) Las celdas pertenecientes a C se marcan con azul.

(b) Cada componente conexa de C se contornea con rojo.

Figura 3.4: Descomposición en componentes conexas.

Es posible obtener las componentes conexas de un conjunto cualquiera de celdas C con el algoritmo 2.

Algoritmo 2: Descomposición en componentes conexas de C

```
   Entrada: C
 1 restantes := C
 2 CC := ∅
 3 pila := Pila vacía
 4 i := 1
 5 mientras restantes ≠ ∅ hacer
 6 |  c := elemento arbitrario de Restantes
 7 |  C_i := {c}
 8 |  restantes := restantes − {c}
 9 |  pila.apilar(c)
10 |  mientras ¬pila.vacia() hacer
11 |  |  c := pila.desapilar()
12 |  |  para cada cA ∈ ady(c) hacer
13 |  |  |  si cA ∈ restantes entonces
14 |  |  |  |  C_i := C_i ∪ {cA}
15 |  |  |  |  restantes := restantes − {cA}
16 |  |  |  |  pila.apilar(cA)
17 |  |  |  fin
18 |  |  fin
19 |  fin
20 |  CC := CC ∪ C_i
21 |  i := i + 1
22 fin
23 devolver CC
```

El algoritmo se resume en elegir una celda $c \in C$ que no esté aún en una componente conexa (línea 6) y aplicar un *depth-first search* (DFS) desde c, agregado todas las celdas recorridas a una misma componente conexa (líneas 7-19). Esto se repite hasta que todas las celdas pertenezcan a alguna componente conexa (línea 5). Este algoritmo es análogo al que está presente en [Hopcroft and Tarjan, 1973].

3.4. Identificación de objetivos

El problema de identificación de objetivos consiste en determinar puntos del espacio a los cuales es conveniente enviar robots para recolectar nueva información sobre el entorno explorado. Estos puntos son los llamados objetivos de exploración (sección 2.1).

3.4.1. Fronteras

En [Yamauchi, 1997] se propone que los lugares que permiten recolectar mayor cantidad de nueva información sobre el entorno son las fronteras entre el espacio conocido y desconocido. Por lo tanto se establece que dichas fronteras deben ser los objetivos de exploración. Al utilizar una grilla de ocupación como mapa, las fronteras se definen como las celdas cuyo estado asociado es *libre* y son adyacentes a una celda cuyo estado asociado es *desconocido* (figura 3.5a). Por lo tanto según Yamauchi los objetivos de exploración serán las celdas frontera F según se definen en (3.4).

$$F = \{c \in CG : e(c) = libre, \exists cA \in ady(c), e(cA) = desconocido\} \tag{3.4}$$

En [Amorin et al., 2019] se argumenta que tratar todas las celdas frontera como objetivos de exploración diferentes podría ser computacionalmente prohibitivo. Por lo tanto, para reducir el costo computacional, se reducen los objetivos de exploración a las celdas frontera más representativas, las cuales se denominan como fronteras significativas.

3.4.2. Fronteras significativas basadas en K-Means

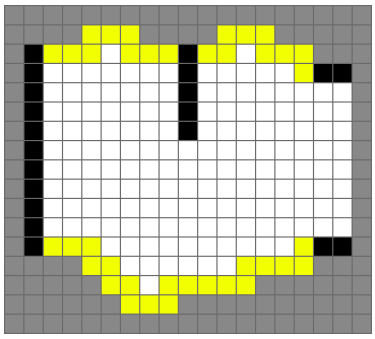
En [Amorin et al., 2019] se propone un método para obtener las fronteras significativas que se basa en el algoritmo K-Means.

El método comienza descomponiendo al conjunto de celdas frontera F en sus componentes conexas $FC = \{F_1, F_2, ...F_N\}$ (sección 3.3.2), un ejemplo de este tipo de descomposición se puede ver en la figura 3.5b.

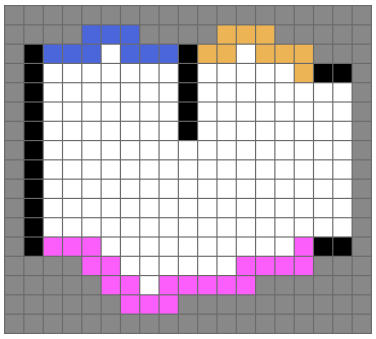
Luego se determinan las fronteras significativas FS_i de cada componente conexa $F_i \in FC$. Esto se hace agrupando las fronteras de F_i con el algoritmo K-Means [MacQueen et al., 1967], y determinando por cada agrupación resultante, a una de las fronteras más cercanas de F_i al centroide de la agrupación como una frontera significativa. Un ejemplo de las fronteras significativas $FS = \bigcup_{i=1}^{N} FS_i$ obtenidas con este método se muestra en la figura 3.5c.

El k utilizado para ejecutar K-Means es el mínimo que logra que para toda frontera $f \in F_i$ existe $fs \in FS_i$ tal que $d_{fs}(f) \leq rango$ siendo $rango$ el alcance de sensado de los robots. Es decir que cada frontera está dentro del rango del sensado desde alguna frontera significativa, cuando esto se cumple se dice que FS_i cubre a F_i, o que FS_i logra el cubrimiento. En la figura 3.5d se puede ver como las fronteras significativas obtenidas logran el cubrimiento.

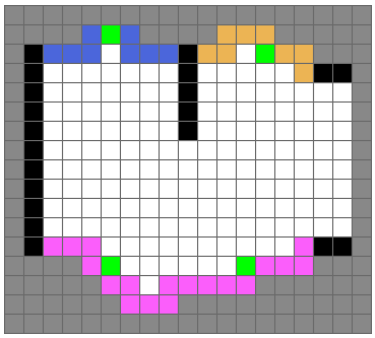
Para encontrar el mínimo k con las que se logra un FS_i que cubra a F_i, se parte con $k = 1$, y si el resultado no cubre a F_i entonces se incrementa k en uno. Esto se repite hasta que el FS_i resultante logre el cubrimiento.



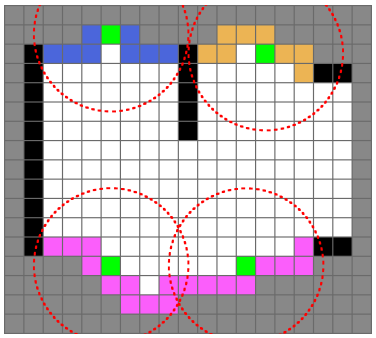
(a) Se identifican las fronteras, indicadas con amarillo.



(b) Descomposición de las fronteras en componentes conexas, cada componente conexa se indica con un color distinto.



(c) Fronteras significativas (indicadas con verde) de cada componente conexa.

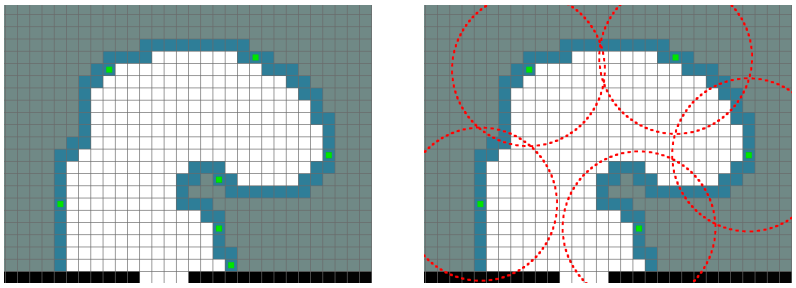


(d) Se logra el cubrimiento.

Figura 3.5: Proceso de obtención de fronteras significativas basado en K-Means. Las circunferencias rojas centradas en las fronteras significativas tienen un radio de 4 *lados de celda* indicando su cubrimiento al usar sensores de rango equivalente. Basado en figuras de [Amorin et al., 2019].

El problema descrito hasta el momento se resume en, dado un conjunto de fronteras F, obtener un conjunto de fronteras significativas FS que cumplen con la restricción de que FS cubre a F. Recordando que el propósito de usar FS como objetivos de exploración en lugar de usar F es reducir los objetivos de exploración, entonces es natural pensar que la soluciones óptimas reducen al mínimo el FS resultante, mientras mantienen la restricción de cubrimiento.

Dado esto es posible ver que en el ejemplo presentado en la figura 3.6 el resultado obtenido por el método presentado en esta sección no es óptimo ya que existen fronteras significativas innecesarias para el cubrimiento.



(a) Fronteras significativas obtenidas según el método basado en K-Means.

(b) Dos de las fronteras significativas de la parte inferior derecha de (a) no son necesarias para lograr el cubrimiento.

Figura 3.6: Resultado subóptimo del método de obtención de fronteras significativas basado en K-Means. Las fronteras de F_i se marcan con azul. Las fronteras significativas se indican con verde y las circunferencias rojas centradas en estas tienen un radio de 6 *lados de celda* indicando su cubrimiento al usar sensores de rango equivalente.

Resultados subóptimos similares se experimentan de forma consistente sobre componentes conexas de fronteras con forma serpenteante, asimétricas y lo suficientemente extensas como para requerir más de 4 fronteras significativas para su cubrimiento. Los resultados empeoran (mayor cantidad de fronteras significativas innecesarias para el cubrimiento) a medida que las componentes conexas de fronteras son más extensas y sus curvas son más pronunciadas.

Esto se presume que se debe principalmente a dos factores relacionados a K-Means: (i) K-Means no considera la restricción de cubrimiento para generar sus resultados, sino que la restricción se fuerza obteniendo a través de prueba y error al mínimo k con el que las fronteras significativas resultantes logran el cubrimiento. (ii) Los centroides resultantes de K-Means no son necesariamente fronteras y deben ser traducidos a fronteras posteriormente.

3.4.3. Fronteras significativas basadas en cubrimiento

En esta sección se detalla un método novedoso de obtención de fronteras significativas, que fue diseñado considerando los dos factores problemáticos relacionados al método de obtención de fronteras significativas basado en K-Means, comentados al final de la sección anterior. El factor (i) se toma en cuenta basando el nuevo método en el concepto del cubrimiento, y el factor (ii) directamente generado fronteras como resultado.

El método de obtención de fronteras significativas que se describe en esta sección, al igual que el descrito en la sección anterior, comienza descomponiendo las fronteras F en sus componentes conexas $FC = \{F_1, F_2, ...F_N\}$, para luego obtener las fronteras significativas FS_i de cada componente F_i. Siendo $FS = \bigcup_{i=1}^{N} FS_i$ el conjunto total de fronteras significativas. La principal diferencia radica en el proceso a través del cual se obtienen las fronteras significativas FS_i de un componente F_i, que se indica en el algoritmo 3.

Algoritmo 3: Obtención de fronteras significativas basada en cubrimiento

Entrada: F_i

1 $FS_i := \emptyset$
2 $UF := F_i$
3 $FP :=$ Cola vacía
4 **para cada** $fp \in \{f \in F_i : \exists c \in ady(f), e(c) = ocupado\}$ **hacer**
5 | $FP.encolar(fp)$
6 **fin**
7 **si** $FP.vacia()$ **entonces**
8 | $FP.encolar($elemento arbitrario de $F_i)$
9 **fin**
10 **mientras** $UF \neq \emptyset$ **hacer**
11 | $fp := FP.desencolar()$
12 | **si** $fp \in UF$ **entonces**
13 | | $dCen := max\{d_{fp}(uf) : uf \in UF\}/2$
14 | | $radio := min\{dCen, rango\}$
15 | | $FSC := UF \cap \mathscr{C}(fp, radio)$
16 | | $fs :=$elemento arbitrario de FSC
17 | | $FS_i := FS_i \cup \{fs\}$
18 | | $RC := \{uf \in UF : d_{fs}(uf) \leq rango\}$
19 | | $UF := UF - RC$
20 | | **para cada** $fp \in \{uf \in UF : \exists rc \in RC, rc \in ady(uf)\}$ **hacer**
21 | | | $FP.encolar(fp)$
22 | | **fin**
23 | **fin**
24 **fin**
25 **devolver** FS_i

El algoritmo mantiene en el conjunto UF a las fronteras de F_i que resta cubrir, y en una cola FIFO llamada FP a las próximas fronteras a cubrir. Se comienza estableciendo que no se tienen todavía fronteras significativas (línea 1) y que resta cubrir a todas las fronteras (línea 2). Luego se inicializa FP encolando en cualquier orden a las celdas frontera de F_i que son adyacentes a celdas de estado ocupado, y de no existir ninguna frontera en estas condiciones encolando una única frontera arbitraria de F_i (líneas 3-9).

Luego el algoritmo se basa en elegir una frontera de UF como frontera significativa fs (líneas 11-16), y posteriormente actualizar las estructuras mantenidas en función de la nueva fs (líneas 17-23). Notar que esto implica reducir el número de celdas en UF ya que como mínimo fs se cubre a sí misma. Este proceso de elección y actualización se repite hasta que no quedan más fronteras de F_i por cubrir (línea 10), y finalmente se devuelven las fronteras significativas FS_i resultantes (línea 25).

Ahora se profundizará sobre los procesos de elección (líneas 11-16) y actualización de estructuras (líneas 17-23).

El proceso de elección de fs tiene como objetivo el asegurar que se cubra una frontera fp desencolada de FP que todavía no esté cubierta. Esto se traduce a la restricción $d_{fp}(fs) \leq rango$. Adicionalmente como heurística, sea $dCen = max\{d_{fp}(uf) : uf \in UF\}/2$ se agrega la restricción $d_{fp}(fs) \leq dCen$. Dicha heurística busca que las fronteras significativas fs elegidas queden centradas en conjunto de las celdas de UF que van a cubrir. Con ambas restricciones en cuenta se determina el conjunto de fronteras significativas candidatas FSC como el conjunto de celdas que pertenecen tanto a UF, como a la circunferencia discretizada centrada en fp y de $radio = min\{dCen, rango\}$, que se denota

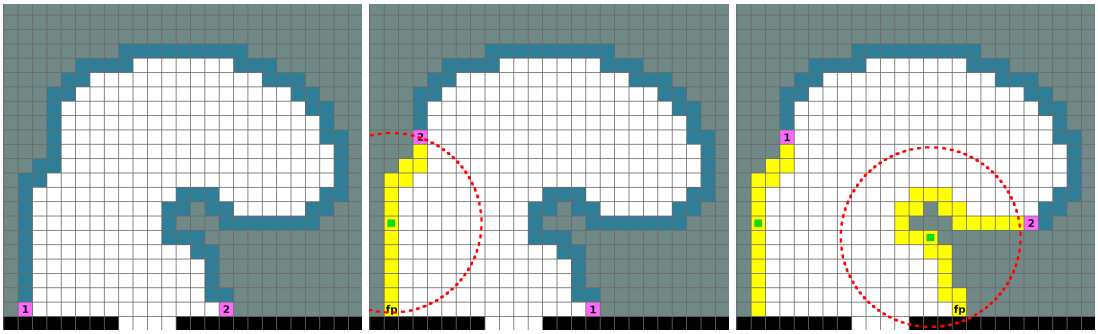
como $\mathscr{C}(fp, radio)$ (3.5).

$$\begin{aligned} \mathscr{D}(fp, radio) &= \{c \in CG : d_{fp}(c) \leq radio\} \\ \mathscr{C}(fp, radio) &= \{c \in \mathscr{D}(fp, radio) : \exists cA \in ady(c), cA \notin \mathscr{D}(fp, radio)\} \end{aligned} \tag{3.5}$$

Y finalmente para elegir una nueva frontera significativa *fs* se elige una frontera de las candidatas FSC, en esta propuesta de forma arbitraria.

Por otro lado el proceso de actualización comienza agregando *fs* a FS_i, y determinando el conjunto de fronteras de UF que son cubiertas por *fs*, al que se denomina como RC (siglas de recién cubiertas). Luego las fronteras de RC se remueven de UF, y por último se encolan en FP las fronteras de UF que son adyacentes a las de RC.

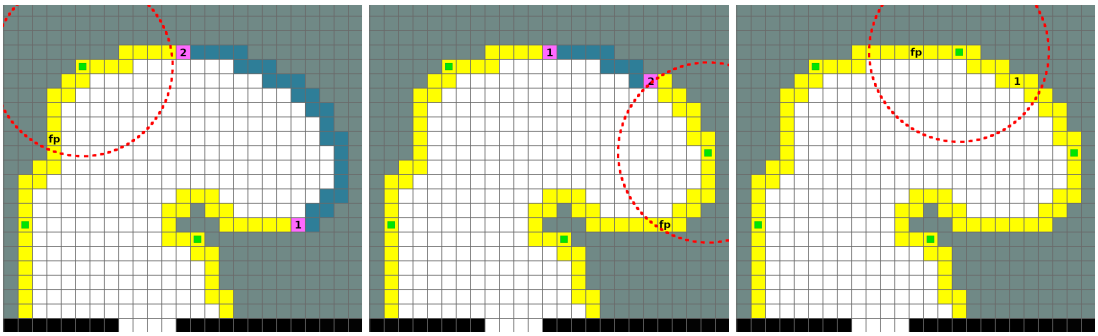
Un ejemplo de ejecución para el caso presentado en la figura 3.6 se muestra en la figura 3.7. En el apéndice B.1 se muestra la misma ejecución con un mayor nivel de detalle.



(a) Incializacion de FP, UF y FS_i.

(b) Se elige un nuevo *fs* y se actualizan UF, FS_i y FP.

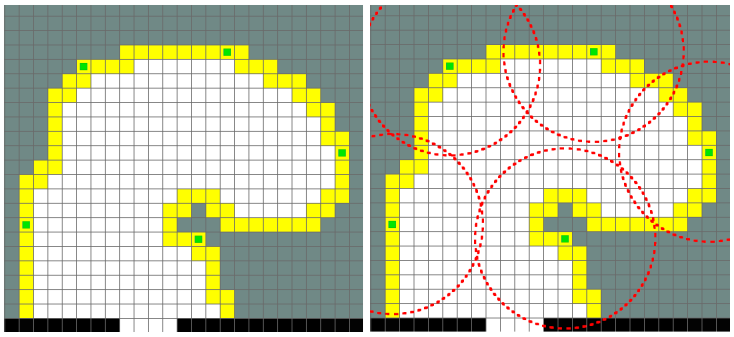
(c) Se elige un nuevo *fs* y se actualizan UF, FS_i y FP.



(d) Se elige un nuevo *fs* y se actualizan UF, FS_i y FP.

(e) Se elige un nuevo *fs* y se actualizan UF, FS_i y FP.

(f) Se elige un nuevo *fs* y se actualizan UF, FS_i y FP.



(g) $UF = \emptyset$ por lo que el algorimo finaliza.

(h) Se logra el cubrimiento.

Figura 3.7: Proceso de obtención de fronteras significativas basado en cubrimiento. Las fronteras de F_i se indican con azul si pertenecen a UF y con amarillo de lo contrario. Con magenta se indican las celdas en FP siendo la numeración su orden en la cola, y fp la última desencolada. Las fronteras significativas se indican con verde y las circunferencias rojas centradas en estas tienen un radio de 6 *lados de celda* indicando su cubrimiento al usar sensores de rango equivalente.

3.5. Asignación de objetivos

Cuando un robot se encuentra ocioso este le solicita a la central que le asigne un nuevo objetivo de exploración, esto desencadena el proceso de asignación de objetivos que se describe a lo largo de esta sección.

La asignación de objetivos se basa en la estrategia de coordinación presentada en 2.3.1, que sostiene que es conveniente que los robots se asignen a los objetivos maximizando la distribución de los robots sobre los segmentos (habitaciones y corredores) que componen un entorno estructurado.

La asignación de objetivos consiste en una subasta donde lo subastado son los objetivos de exploración, el subastador es la estación central y los postores son los robots. La subasta se puede separar en tres partes, *obtención de información*, *valuación* y *resolución*. La *obtención de información* es la etapa inicial donde la central obtiene la información necesaria para llevar adelante la subasta, por ejemplo los objetivos de exploración. La *valuación* abarca la distribución de la información obtenida desde la central hacia los robots, la valuación que cada robot hace para cada objetivo recibido y el posterior envió de las valuaciones desde los robots hacia la central. La *resolución* se lleva a cabo en la central y comprende la recepción de las valuaciones, la determinación de que objetivo asignar a cada a robot y la notificación a cada robot del objetivo que le fue asignado.

3.5.1. Obtención de información

Como se mencionó anteriormente, cuando un robot ocioso solicita un objetivo a la central se da comienzo a una nueva subasta. La etapa de *obtención información* es la primera etapa de la subasta. La información que se obtiene en esta etapa son los objetivos de exploración, los segmentos del entorno explorado y un GVD.

La identificación de objetivos se lleva a cabo con el método descrito en la sección 3.4.3.

Los segmentos se obtienen aplicando una técnica que se basa en la propuesta de [Thrun, 1998] explicada en la sección 2.2.6. Detalles de como esta técnica fue

implementada se presentan en la sección 3.6.

El GVD requerido para determinar los segmentos se construye de forma incremental con una variante del *brushfire dinámico* que se comenta en la sección 3.7.

3.5.2. Valuación

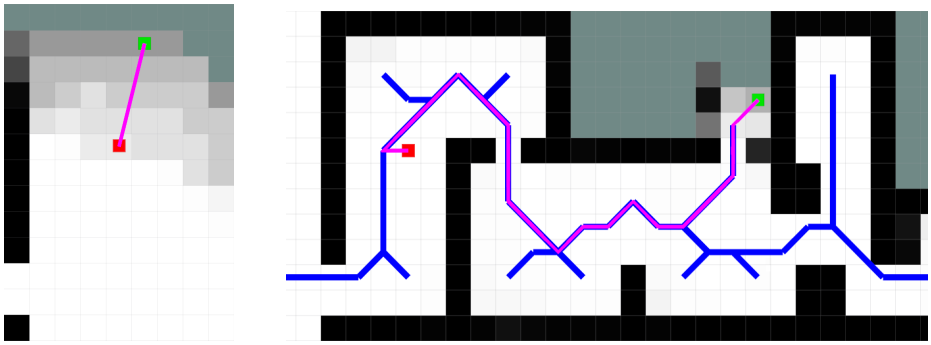
Terminada la etapa obtención de información comienza la etapa de *valuación* con la central distribuyendo los objetivos y el GVD hacia los robots.

Cuando un robot recibe dicha información este calcula dos valores para cada objetivo, el largo de un camino hacia él y un costo asociado a la complejidad de ese camino. Esto implica determinar un camino. En este proyecto los caminos se componen de una secuencia de puntos en el espacio que comienza en la posición del robot y termina en el objetivo. Los caminos se calculan de forma distinta dependiendo de la clase de objetivo al que llevan, existiendo dos clases, trivial y no trivial.

Los objetivos triviales se definen como los objetivos para los cuales el segmento de recta que existe entre el robot y el objetivo no contiene obstáculos, de lo contrario el objetivo es no trivial. Para determinar esto, se discretiza el segmento de recta en celdas [Foley et al., 1996] y se comprueba en la grilla de ocupación almacenada en el robot (que constituye el mapa completo del entorno) si alguna de esas celdas tiene estado ocupado.

Los caminos hacia los objetivos triviales se determinan como los puntos ubicados en los extremos del segmento de recta que se encuentra libre de obstáculos que existe entre el robot y el objetivo. Un ejemplo de un camino trivial se muestra en la figura 3.8a.

En el caso de los objetivos no triviales se hace uso del GVD recibido como parte de la información de la subasta, aprovechando que este constituye un *roadmap* (sección 2.2.1). En este caso el proceso para construir el camino se puede dividir en tres partes, cada una asociada a una de las propiedades que caracterizan a un *roadmap*: accesibilidad, conectividad y capacidad de salida. La parte asociada a la accesibilidad consiste en encontrar un camino desde el robot al GVD, lo cual se hace a través de un *breadth-first search* (BFS), comenzando en la celda asociada a la posición del robot y deteniéndose apenas se visita una celda c_1 perteneciente al GVD. La parte que se relaciona a la capacidad de salida es análoga a la parte asociada a la accesibilidad pero partiendo desde el objetivo y llegando a una celda c_2 perteneciente al GVD. Por último en la parte vinculada con la conectividad se determina un camino sobre las celdas pertenecientes al GVD desde c_1 a c_2, y para esto se utiliza el algoritmo A^*. Finalmente conectando los tres caminos se obtiene el camino completo. En la figura 3.8b se presenta un ejemplo de un camino para un objetivo no trivial.



(a) Trivial. (b) No trivial.

Figura 3.8: Caminos calculados desde la posición del robot indicada en rojo, hasta el objetivo indicado en verde. Los puntos del camino se conectan entre sí con líneas magenta.

Respecto al costo asociado a la complejidad, en el caso de objetivos triviales este es proporcional al mínimo ángulo que el robot debe rotar para que su parte delantera apunte hacia el objetivo. Y en el caso de objetivos no triviales es igual al máximo costo asociado a la complejidad que fue asignado a un objetivo trivial, esto se hace para diferenciar los objetivos triviales sin dejar beneficiados a los objetivos no triviales. Este costo tiene el propósito de incentivar que los robots continúen siendo asignados a los objetivos triviales que tienen por delante, evitando rotaciones, a no ser que la diferencia en los largos de los caminos lo amerite.

Una vez calculados los valores para cada objetivo estos se envían hacia la central junto a la ubicación actual del robot, siendo toda esta información en su conjunto considerada una valuación.

3.5.3. Resolución

La etapa de resolución comienza con la central recibiendo las valuaciones realizadas por los robots. Al recibirse la primera valuación existe una ventana de tiempo en que la central continuará recibiendo nuevas valuaciones. Terminada esta ventana se ejecuta un algoritmo que asigna objetivos a los robots. Y por último las asignaciones realizadas son comunicadas a los robots.

Recepción de valuaciones

Como se mencionó anteriormente luego de recibida la primera valuación existe una ventana de tiempo en la que la central continuará recibiendo nuevas valuaciones.

El propósito de utilizar una ventana de tiempo es que en escenarios reales existen razones por las cuales los robots pueden dejar de comunicarse, como por ejemplo accidentes que causen problemas de hardware. Por lo tanto esperar por las valuaciones de todos los robots puede resultar en una espera infinita que deja inoperativa a toda la flota.

Dado que los problemas que cada robot debe resolver para generar su valuación son similares entre sí, se asume que los tiempos que demoran en generarla también son similares. Adicionalmente dichos tiempos tienden a aumentar a medida que el entorno es explorado. Por lo tanto se optó por utilizar una ventana dinámica cuya duración es calculada en función de la demora de la recepción de valuaciones experimentada en la subasta anterior, buscando que la duración de

la ventana sea suficiente para que todos los robots que estén en condiciones de enviar sus valuaciones tengan el tiempo necesario para hacerlo.

Notar que una vez que se reciben las valuaciones de todos los robots en funcionamiento es innecesario continuar esperando, por lo tanto de recibirse dichas valuaciones la ventana es terminada prematuramente, evitando esperas innecesarias.

Sea Δr_{i-1} el tiempo que paso en la subasta $i-1$ desde la recepción de la primera valuación hasta que se da por terminada la recepción valuaciones, la duración Vr_i de la ventana en la subasta i se define según (3.6).

$$Vr_i = \begin{cases} 0{,}5s & si\ i = 0 \\ max(0{,}5s + 2\Delta r_{i-1},\ Vr_{i-1}) & si\ i > 0 \end{cases} \tag{3.6}$$

El valor de la duración Vr_i resulta de asumir que el tiempo que se demora en recibir las valuaciones no aumenta más del doble de una subasta a la siguiente. Adicionalmente dando un margen de error de 0.5 segundos, especialmente útil cuando Δr_{i-1} es muy pequeño.

Algoritmo de asignación de objetivos

Cuando la central recibe una valuación de un robot r_i, esta se encarga de procesar la información contenida en la valuación de forma de obtener un único costo $c_{i,j}$ para cada objetivo o_j. Dado un objetivo o_j la valuación del robot r_i contiene el largo $cl_{i,j}$ del camino de r_i a o_j, y el costo $cc_{i,j}$ asociado a la complejidad de dicho camino. Adicionalmente contiene la posición de r_i a través de la cual se determina un descuento $ds_{i,j}$ que toma un valor constante menor a cero si el robot r_i pertenece al mismo segmento que el objetivo o_j y cero de lo contrario. El costo $c_{i,j}$ se calcula según la ecuación (3.7).

$$c_{i,j} = cl_{i,j} + cc_{i,j} + ds_{i,j} \tag{3.7}$$

Los costos $cl_{i,j}$ y $cc_{i,j}$ tienen el propósito de penalizar a los objetivos o_j según se estime que toman más tiempo en ser completados por r_i. Por otro lado el descuento $ds_{i,j}$ incentiva que los robots se mantengan explorando un mismo segmento, lo que lleva a comportamientos deseables como que un robot continue explorando un mismo corredor, revelando rápidamente la estructura del entorno.

Con los costos $c_{i,j}$ determinados, se ejecuta un algoritmo que resulta en la asignación de objetivos a robots considerando tanto los costos, como la estrategia de coordinación aplicada, maximizando la distribución de los objetivos asignados sobre los segmentos del entorno. Este algoritmo de asignación tiene dos ventajas respecto al algoritmo utilizado en [Wurm et al., 2008] comentado en la sección 2.3.1. La primera es que resuelve la asignación de objetivos directamente, en lugar de asignar primero robots a segmentos, para luego resolver las asignaciones de objetivos localmente en cada segmento. La segunda es que maximiza la distribución de los robots en los segmentos teniendo en cuenta el número de objetivos que existe en cada segmento, en lugar de simplemente realizar una distribución uniforme. Una situación que ejemplifica el problema de las distribuciones uniformes es una en la que se tienen 10 robots y 2 segmentos, s_1 con 2 objetivos y s_2 con 10. Una distribución uniforme resulta de asignar 5 robots a cada segmento aunque esto implique que en s_1 queden 3 robots ociosos (sin objetivo asignado). El algoritmo presentado a continuación resuelve este tipo de problemas, y resulta en distribuciones uniformes de haber suficientes objetivos en cada segmento. Por ejemplo para el caso planteado anteriormente su ejecución resulta en 2 robots asignados en s_1 y los restantes 8 asignados en

s_2, mientras que para un caso alternativo en el que s_1 y s_2 tienen 5 (o más) objetivos cada uno, resulta en una distribución uniforme.

El algoritmo de asignación se corresponde al algoritmo 4 donde R es el conjunto de identificadores de los robots de los cuales se recibieron valuaciones, O es el conjunto de identificadores de los objetivos valuados, y S el conjunto de identificadores de los segmentos. El conjunto $Costos$ contiene una tupla $(c_{i,j}, r_i, o_j)$ por cada costo $c_{i,j} \in \mathbb{R}$ de cada robot $r_i \in R$ para cada objetivo $o_j \in O$. La función $segmento : O \to S$ dado un objetivo devuelve el segmento al que pertenece, y la función $\#objetivos : S \to \mathbb{N}$ devuelve el número de objetivos que existen en un segmento.

Algoritmo 4: Asignación de objetivos a robots

Entrada: $R, O, S, Costos$
$segObjs :=$ Multiconjunto vacío
para cada $s \in S$ **hacer**
  $segObjs = segObj \cup \{\#objetivos(s)\}$
fin
$numSegs := |S|$
$numRobs := |R|$
$converge := falso$
mientras $segObjs \neq \emptyset \ \wedge \ \neg converge$ **hacer**
  $cocienteDist := numRobs \ \ div \ \ numSegs$
  $restoDist := numRobs \ \ mod \ \ numSegs$
  $segObj := min \ segObjs$
  $segObjs := segObjs - \{segObj\}$
  $converge := (segObj > cocienteDist) \vee$
   $(restoDist = 0 \wedge segObj = cocienteDist)$
  si $\neg converge$ **entonces**
    $numRobots := numRobots - segObj$
    $numSeg := numSeg - 1$
  fin
fin
$asignaciones := \emptyset$
$segRobNums :=$ diccionario de S a $\mathbb{N}$, siendo 0 el valor por defecto
mientras $R \neq \emptyset \wedge O \neq \emptyset$ **hacer**
  $(c_{i,j}, r_i, o_j) := min \ Costos$ `// Tupla de` $Costos$ `con mínimo` $c_{i,j}$
  $Costos := Costos - \{(c_{i,j}, r_i, o_j)\}$
  $s := segmento(o_j)$
  $distribuido := segRobNums[s] < cocienteDist \vee$
   $(restoDist > 0 \wedge segRobNums[s] = cocienteDist)$
  si $r_i \in R \wedge o_j \in O \wedge distribuido$ **entonces**
    $asignaciones := asignaciones \cup \{(o_j, r_i)\}$
    $segRobNums[s] := segRobNums[s] + 1$
    $R := R - \{r_i\}$
    $O := O - \{o_j\}$
    si $segRobNums[s] = cocienteDist + 1$ **entonces**
      $restoDist := restoDist - 1$
    fin
  fin
fin
devolver asignaciones

El algoritmo se puede dividir en dos partes, el *calculo de valores* (líneas 1-18) y la *asignación distribuida* (líneas 19-36). En el *calculo de valores* se calculan los valores *cocienteDist* y *restoDist*, que son fundamentales para llevar a cabo la *asignación distribuida*, en la que se determina la asignación de objetivos a robots.

La distribución deseada se corresponde con la que se logra al asignar haciendo N rondas, en las cuales se asigna un solo robot a cada segmento que al comenzar la ronda contenga objetivos sin asignar. La ronda N termina cuando no quedan robots por asignar, u objetivos a los cuales asignarlos. La distribución resultante de esta asignación se considera la deseada porque en cada ronda se maximiza la distribución de los robots sobre los segmentos que tienen objetivos suficientes.

La idea detrás de los valores *cocienteDist* y *restoDist* es que en la distribución deseada existen K segmentos que cumplen con la restricción de distribución uniforme (2.8), existiendo *restoDist* segmentos con *cocienteDist*$+1$ robots asignados, y $K-$*restoDist* segmentos con *cocienteDist* robots asignados. Mientras que los segmentos restantes contienen una cantidad de objetivos menor a *cocienteDist* y todos sus objetivos son asignados.

El *calculo de valores* se hace inicializando los valores *cocienteDist* y *restoDist* asumiendo que la distribución deseada coincide con una completamente uniforme, sin considerar si esto es consistente con el número de objetivos contenidos en cada segmento (líneas 5-6 y 9-10 primera iteración). Luego de forma iterativa, segmento a segmento, y comenzando por los que tienen menos objetivos, se comprueba si el número de objetivos del segmento cumple con ser lo suficientemente grande para que los valores actuales lleven a la distribución deseada (líneas 11-13). De no cumplirse esto se ajusta los valores para que la distribución sea uniforme en los segmentos que restan iterar, considerando que los objetivos del segmento actual son todos asignados (líneas 14-17 y 9-10), y después se continúa iterando. Por otro lado, de cumplirse se sabe que el resto de segmentos a iterar también cumplen ya que contienen una cantidad mayor o igual de objetivos, y por lo tanto se puede decir que los valores *cocienteDist* y *restoDist* convergen a los valores que llevan a la distribución deseada.

Con los valores *cocienteDist* y *restoDist* calculados la *asignación distribuida* consiste en retirar las tuplas $(c_{i,j}, r_i, o_j)$ de *Costos* comenzando por las de menor costo $c_{i,j}$ (líneas 22-23). Para cada tupla retirada se asigna el objetivo o_j al robot r_i (líneas 27-33), si tanto r_i como o_j no forman parte de ninguna de las asignaciones realizadas hasta el momento, y si la asignación es consistente con la distribución deseada (líneas 25-26). Esto se repite hasta que todos los objetivos estén asignados, o que no queden más robots por asignar (línea 21).

Luego de ejecutar el algoritmo de asignación la central informa a cada robot el objetivo que le fue asignado, quedando a la espera de nuevos pedidos de comenzar una nueva subasta por parte de los robots.

3.6. Segmentación

El método de segmentación utilizado para identificar los segmentos del entorno se basa en el que fue comentado en la sección 2.2.6. Dado que el entorno se representa con grillas discretas en lugar de ser un espacio continuo, es necesario definir los equivalentes discretizados de GVD, como también de sus puntos críticos y líneas críticas asociadas, a partir de las cuales se dividen los segmentos.

Un *GVD discretizado* se define como el conjunto *GVDD* de celdas que pertenecen al GVD discretizado, la construcción de este se trata en la sección 3.7. Desde este punto se refiere al GVD discretizado (*GVDD*) simplemente como *GVD*.

Dada la función de despeje discretizada $DD : CG \to \mathbb{R}$ que para cada celda

de *CG* devuelve la distancia a su celda más cercana que pertenece a un generador, se define el conjunto de las *celdas críticas* (puntos críticos discretizados) *CCr* según (3.8).

$$\begin{aligned} CCr = \{c \ \in GVD : &\forall c_1 \in ady(c) \cap GVD, DD(c) \leq DD(c_1), \\ &\exists c_2 \in ady(c) \cap GVD, DD(c) < DD(c_2)\} \end{aligned} \tag{3.8}$$

Esta definición traduce el concepto de punto crítico, del espacio continuo al espacio discreto, aunque no solo se remite a traducir sino que se altera ligeramente la definición original. Específicamente las celdas críticas además de ser las celdas del GVD que son mínimos locales estrictos de *DD* en el GVD, son también las que no tienen celdas adyacentes en el GVD con menor *DD* y al menos una de ellas tiene un *DD* mayor. El motivo de esto es permitir celdas críticas en situaciones como la presentada en la figura 3.10c, donde se puede observar un portal horizontal que lleva de un corredor (arriba) a una habitación (abajo). La celda crítica que se ubica en dicho portal no es un mínimo local estricto ya que la celda de arriba adyacente en el GVD tiene igual *DD*, por lo tanto si esta fuera la única condición de ser una celda crítica dicha celda no lo sería. Esto resultaría en que el corredor y la habitación sean identificados como un único segmento, lo cual no es deseable. En cambio la celda en cuestión sí cumple con las condiciones planteadas en (3.8) ya que la celda de arriba tiene igual *DD*, la de abajo tiene un *DD* mayor y esas son las únicas dos celdas adyacentes en el GVD.

Para adaptar el concepto de línea crítica del espacio continuo al espacio discreto, primero es necesario adaptar el concepto de punto base. Los puntos base de p se corresponden con el conjunto de puntos pertenecientes a generadores, que están a mínima distancia a p. Sea *CGen* el conjunto formado por todas las celdas pertenecientes a algún generador, las *celdas base* (puntos base discretizados) CB_c de una celda c se definen como el conjunto de las celdas pertenecientes a *CGen* que están a una distancia de c menor o igual a $DD(c) + L$, como se expresa en (3.9), siendo L el largo de las celdas.

$$CB_c = \{b \in CGen : d_c(b) \leq DD(c) + L\} \tag{3.9}$$

Adaptar directamente la definición de punto base en el espacio continuo, a celda base en el espacio discreto, resulta en que solo las celdas de *CGen* que están a mínima distancia de c son celdas base de c. Sin embargo la definición dada en (3.9) no se corresponde con esta, ya que al adaptarse de forma directa existen casos donde los resultados difieren de los que se obtienen en el espacio continuo, lo que puede llevar a errores de segmentación. Uno de estos casos es el presentado en la figura 3.9 donde se muestra una sección horizontal de un corredor. En el espacio continuo los puntos ubicados en el centro del corredor tienen dos puntos base, uno en cada pared a los lados del corredor. El problema ocurre cuando la discretización de dicho corredor resulta en un número par de celdas libres entre las celdas ocupadas b_1 y b_2 correspondientes a las paredes del corredor. En esta situación se tienen dos celdas c_1 y c_2 en el centro del corredor, lo deseable es que al menos una de estas se corresponda con los puntos p del centro del corredor que se ubican en el borde entre c_1 y c_2. Sin perdida de generalidad se asume que c_1 es la celda que se corresponde con dichos p, y por lo tanto análogamente a lo que sucede con los puntos base de estos p, c_1 debería tener dos celdas base una en cada pared a los lados del corredor (celdas b_1 y b_2). La celda $b_1 \in CGen$ está a distancia $d_{c_1}(b_1) = DD(c_1) = d - \frac{L}{2}$ de c_1, y al ser la mínima es seguro que b_1 es una celda base de c_1. Por otro lado la celda $b_2 \in CGen$ está a distancia $d_{c_1}(b_2) = d + \frac{L}{2}$ de c_1, y aunque no sea la mínima, dado lo que ocurre en el espacio continuo b_2 también debería ser una

celda base de c_1. Para considerar estos casos la definición presentada en (3.9) incluye en CB_c a todas las celdas de $CGen$ que tienen una distancia a c que está entre la mínima posible de $DD(c)$ y una máxima tolerada de $DD(c) + L$, donde la diferencia entre ambos extremos se conoce como tolerancia. Con esta definición se obtiene que $b_2 \in CB_{c_1}$ como es deseado, ya que recordando que $b_2 \in CGen$ entonces:

$$b_2 \in CB_{c_1} \iff d_{c_1}(b_2) \leq DD(c_1) + L \iff d + \frac{L}{2} \leq (d - \frac{L}{2}) + L$$
$$\iff d + \frac{L}{2} \leq d + \frac{L}{2} \iff 0 \leq 0 \blacksquare$$

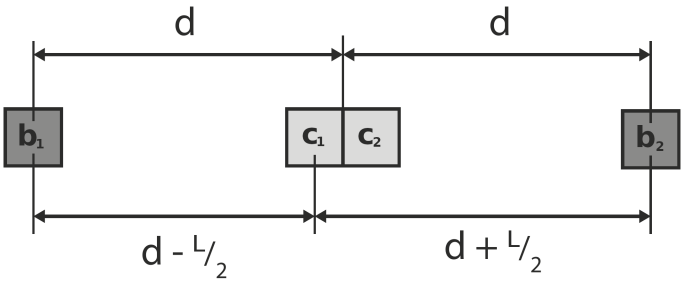


Figura 3.9: Caso en el cual se requiere de una tolerancia para detectar correctamente a las celdas base. Modificada de [Liu et al., 2015]

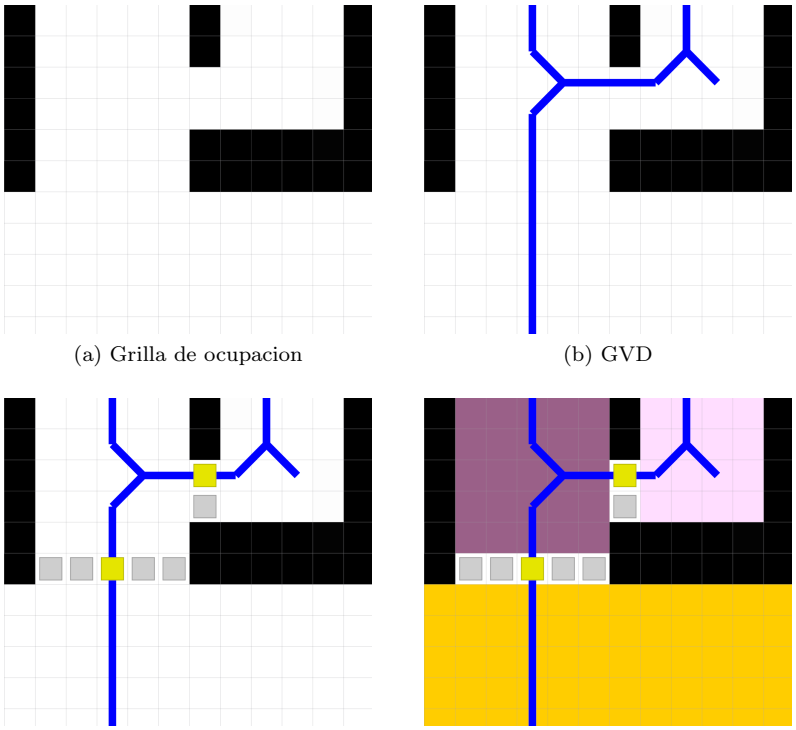
La definición (3.9) se basa en el análisis realizado en [Liu et al., 2015] donde se resuelve que la la tolerancia debe ser L.

Finalmente las *líneas críticas discretizadas* se definen como las líneas discretizadas [Foley et al., 1996] definidas entre los centros de las celdas críticas y los centros de cada una de sus celdas base. El conjunto de todas las celdas pertenecientes a alguna línea crítica discretizada se conoce como $LCrD$.

Una vez determinadas las celdas críticas y las líneas críticas discretizadas es posible obtener la segmentación del entorno. Esto se hace aplicando un algoritmo de descomposición en componentes conexas (sección 3.3.2) donde cada componente conexa resultante es un segmento, y el conjunto que se descompone es el que está compuesto por las celdas libres de CG que no son celdas críticas ni forman parte de una línea crítica discretizada, es decir:

$$\{c \in CG : estado(c) = libre, c \notin CCr, c \notin LCrD\}$$

En la figura 3.10 se muestra el proceso de segmentación completo, desde la grilla de ocupación usada como entrada 3.10a, pasando por el GVD generado 3.10b, la detección de celdas críticas y líneas críticas discretizadas 3.10c, y terminando con la segmentación del entorno 3.10d.



(a) Grilla de ocupacion

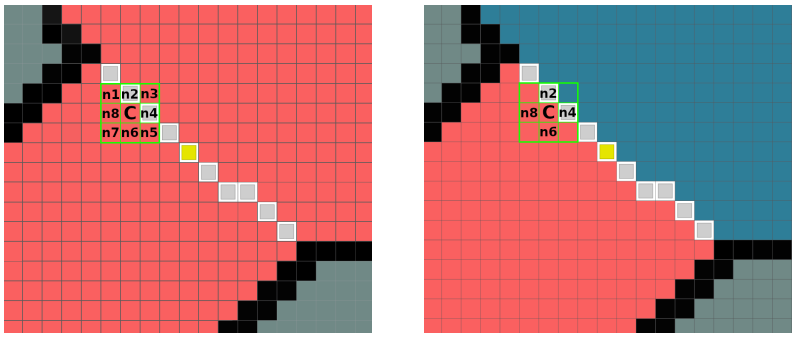
(b) GVD

(c) Celdas críticas, contienen cuadrados amarillos. Y celdas pertenecientes a líneas críticas discretizadas, contienen cuadrados grises

(d) Segmentación del entorno, cada segmento se indica con un color distinto

Figura 3.10: Proceso de segmentación de un entorno representado con una grilla.

Una consideración a tener en cuenta es que para obtener una segmentación correcta el algoritmo 2 debe ejecutarse con ady_4 como función de adyacencia (sección 3.3.1), ya que si se usa ady_8 las líneas críticas discretizadas diagonales no separan segmentos. Esto se debe a que siempre existen celdas a ambos lados de una línea discretizada diagonal que son adyacentes diagonalmente como se muestra en la figura 3.11. De usarse ady_4 las líneas críticas diagonales sí separan segmentos al no estarse permitiendo la adyacencia diagonal.



(a) Usando ady_8

(b) Usando ady_4

Figura 3.11: Línea crítica discretizada diagonal, y sus efectos en la segmentación al usar ady_4 o ady_8. La celda c es un ejemplo de una celda a un lado de la línea que es adyacente según ady_8 a la celda $n3$ al otro lado de la línea, cosa que no sucede con ady_4.

3.7. Diagrama generalizado de Voronoi

Considerando los problemas de eficiencia del algoritmo no incremental de construcción del GVD llamado *brushfire* (sección 2.2.4), y los problemas de su variante incremental denominada *brushfire dinámico* (sección 2.2.5) como se propone originalmente en [Kalra et al., 2009]. Se decidió basar la construcción del GVD en la versión de *brushfire dinámico* propuesta en [Lau et al., 2013], a la que se estará haciendo referencia cuando se mencione "*brushfire dinámico* original" desde este punto.

La implementación realizada en este proyecto mantiene el concepto de ondas consistentes e inconsistentes del *brushfire dinámico* original para el calculo incremental de un mapa de distancia. Adicionalmente mantiene que dichas ondas consistentes e inconsistentes remueven del GVD a las celdas que atraviesan, y que las celdas candidatas a pertenecer al GVD son las celdas en las que ocurren choques de ondas consistentes.

Los cambios principales respecto al *brushfire dinámico* original se dan en el criterio con el cual se determina la pertenencia de las celdas candidatas al GVD y en como se considera el espacio desconocido al construir el GVD durante la exploración. Estos se tratan en las siguientes secciones.

3.7.1. Criterio de pertenencia

En el *brushfire dinámico* original la pertenencia de las celdas al GVD se determina en los choques de ondas consistentes. Dada la función $gen : CG \rightarrow CGen$ que para cada celda c de la grilla devuelve una de las celdas que pertenecen a un generador que están a mínima distancia de c, el criterio de pertenencia original esencialmente consiste en que, sean c_1 y c_2 las celdas involucradas en un choque de ondas consistentes, estas pertenecen al GVD si $gen(c_1) \neq gen(c_2)$ y $gen(c_1) \notin ady(gen(c_2))$, es decir si $gen(c_1)$ y $gen(c_2)$ no son iguales ni adyacentes entre sí.

La definición de GVD para el espacio continuo establece que un punto p pertenece al GVD si tiene al menos dos generadores distintos a mínima distancia, lo cual sucede si y solo si al menos dos de los puntos pertenecientes a generadores, que están a mínima distancia de p pertenecen a generadores distintos. Dado esto se puede decir que un punto pertenece al GVD si tiene al menos dos puntos base (sección 3.6) que pertenecen a generadores distintos, que en un espacio discretizado en celdas se traduce a que una celda tenga dos celdas base que pertenecen a generadores distintos.

Dado este análisis y la necesidad de determinar las celdas base que fue planteada en la sección 3.6, se decidió cambiar el criterio de pertenencia al GVD, siendo el nuevo criterio de pertenencia de una celda al GVD que al menos dos de sus celdas base pertenezcan a generadores distintos.

Las celdas base se obtienen en los choques de ondas consistentes. En cada actualización del mapa de distancia correspondiente a un incremento, se registran todas las celdas involucradas en choques de ondas consistentes. Luego de finalizar la actualización del mapa de distancia, con el algoritmo 5 se obtienen las celdas base CB_c de cada celda c registrada previamente. Notar que la función de despeje discretizada $DD : CG \rightarrow \mathbb{R}$ se corresponde los valores del mapa de distancia calculado.

Algoritmo 5: Obtención de las celdas base CB_c de una celda c (simplificada)

```
   Entrada: c
 1 minD := ∞
 2 para cada cA ∈ ady(c) hacer
 3 |  b := gen(cA)
 4 |  tolerado := d_c(b) ≤ DD(c) + L
 5 |  noIgNiAdy := b ≠ gen(c) ∧ b ∉ ady(gen(c))
 6 |  masCercano := d_c(b) < minD
 7 |  si tolerado ∧ noIgNiAdy ∧ masCercano entonces
 8 |  |  minD := d_c(b)
 9 |  |  ultimo := b
10 |  fin
11 fin
12 CB_c := {gen(c)}
13 si minD < ∞ entonces
14 |  CB_c := CB_c ∪ {ultimo}
15 fin
16 devolver CB_c
```

El algoritmo para cada celda cA adyacente a c (línea 2) obtiene un candidato a ser una celda base de c (línea 3) y comprueba una serie de condiciones utilizadas para determinar las celdas base de c (líneas 4-6). La condición que se comprueba en la línea 4 es que la celda base candidata esté dentro del intervalo definido por la tolerancia (sección 3.6). La condición que se verifica en la línea 5 es que el candidato no sea $gen(c)$ ni adyacente a él. Finalmente la condición presente en la línea 6 establece que el candidato actual debe tener una distancia a c menor a la de los candidatos procesados anteriormente que cumplieron con todas las condiciones. Al finalizar la ejecución se devuelve como celdas bases a $gen(c)$ y en el caso de que exista al último candidato en cumplir todas las condiciones (líneas 12-16).

Una vez obtenidas las celdas base de un celda c, para determinar la pertenencia de c al GVD resta verificar si existen dos celdas base de c que pertenecen a generadores distintos. Para esto se aplica una técnica derivada del criterio de pertenencia de celdas al GVD presente en el *brushfire dinámico* original, que consiste en que dos celdas pertenecen a distintos generadores si son distintas y no adyacentes entre sí, y de lo contrario pertenecen al mismo generador. De esta forma se logra obtener incrementalmente tanto el GVD, como las celdas base necesarias para segmentar.

Es pertinente aclarar que el algoritmo 5 es una versión simplificada del que está presente en la implementación del proyecto. Las simplificaciones son dos, la primera es que para cada celda cA adyacente a c solo se considera como candidata a una de las celdas pertenecientes a un generador que están a mínima distancia de cA (la resultante de $gen(cA)$). Y la segunda es que solo se permiten hasta dos celdas base. En la sección A.1 se presenta el algoritmo completo sin las simplificaciones mencionadas, es decir que considera como candidatas a todas las celdas que pertenecen a un generador que están a mínima distancia, y permite más de dos celdas base.

3.7.2. Consideraciones sobre el espacio desconocido

Mientras la exploración está en curso existe espacio desconocido del entorno que puede ser explorado, este se representa con celdas de estado desconocido en la grilla de ocupación (sección 3.3.1) usada como mapa del entorno. Dado que la definición del GVD (sección 2.2.3) no contempla el espacio desconocido, para

poder construir un GVD sobre una grilla de ocupación durante la exploración, es necesario decidir cómo considerar las celdas de estado desconocido. Sin embargo, este no es un tema que se trate de forma explicita en los trabajos estudiados sobre la construcción incremental del GVD (sección 2.2.5).

A partir de código e imágenes disponibles es posible deducir que en [Kalra et al., 2009] y [Lau et al., 2013] al construir el GVD las celdas de estado desconocido se consideran iguales a las de estado libre. De esta forma se obtienen resultados similares al que se presenta en la figura 3.12. Notar que otra característica particular presente en estas implementaciones es que las celdas ubicadas en los bordes del entorno se consideran como pertenecientes a generadores.

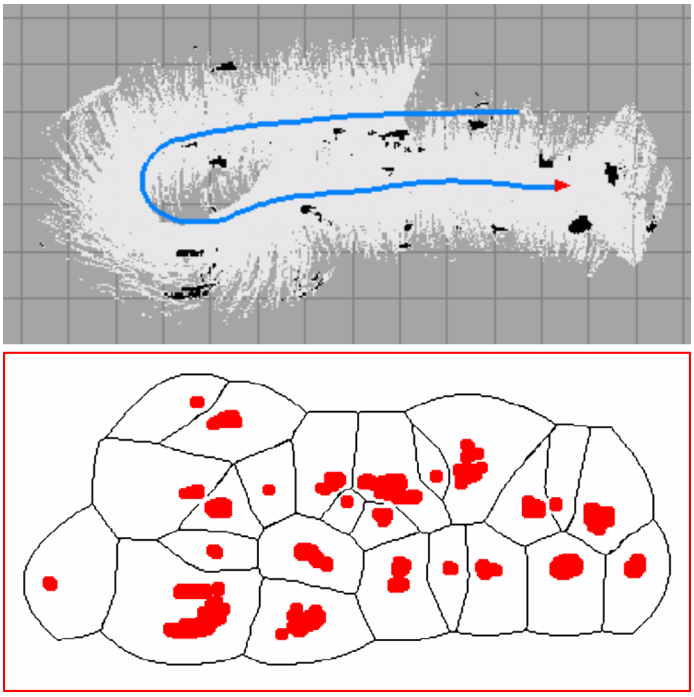
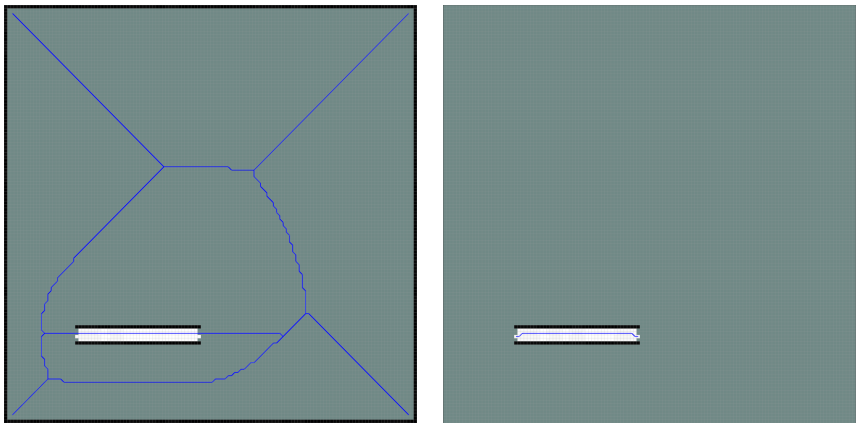


Figura 3.12: Resultado de considerar las celdas de estado desconocido iguales a las de estado libre. En la parte superior se muestra la grilla de ocupación actual. En la inferior su GVD correspondiente, indicándose con rojo los obstáculos. Extraída de [Kalra et al., 2009].

Que las celdas desconocidas se consideren iguales a las celdas libres implica que las ondas utilizadas para construir el GVD pueden propagarse por el espacio desconocido y generar la pertenencia de celdas desconocidas al GVD. En la figura 3.13a se puede observar que considerar las celdas desconocidas como celdas libres lleva a un GVD que abarca todo el entorno sin importar que solo una pequeña porción de este sea conocida.

Dado que en el contexto del proyecto no es necesario construir el GVD fuera del área conocida, se desarrolló un método que solo construye el GVD en las celdas libres, evitando el costo computacional de construir el GVD en el espacio desconocido. Sea UF el conjunto de celdas desconocidas que tienen al menos una celda adyacente de estado libre, el método consiste en considerar que las celdas desconocidas no propagan ondas, y que las celdas de UF pertenecen a generadores ($UF \subseteq CGen$) al igual que las celdas ocupadas. Con este método para el caso presentado en la figura 3.13a se obtiene un GVD que se reduce al espacio conocido como se muestra en la figura 3.13b.

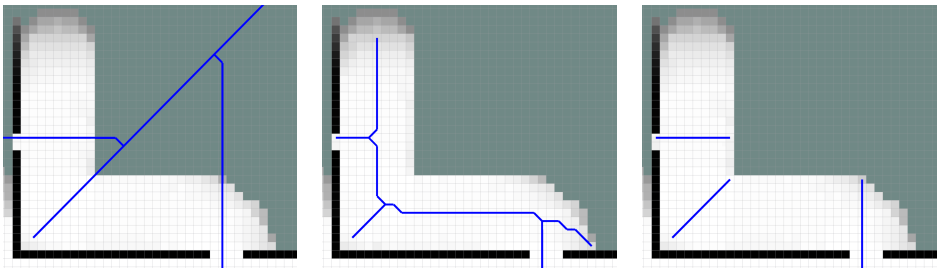


(a) Celdas desconocidas se consideran como libres.

(b) Celdas desconocidas no propagan ondas y $UF \subseteq CGen$.

Figura 3.13: Formas de considerar el espacio desconocido. El GVD generado se indica en azul.

Considerar únicamente que las celdas desconocidas no propagan ondas también restringe la construcción del GVD al espacio conocido, aunque esto puede ser más simple e intuitivo tiene el problema de generar GVD disconexos en casos donde esto se puede evitar. El problema con los GVD disconexos es que estos no cumplen con la propiedad de *conectividad* necesaria para ser un *roadmap* (sección 2.2.1), lo que genera errores al utilizar el GVD como *roadmap* para la planificación. En la figura 3.14 se muestra el GVD generado con cada una de las tres formas de considerar el espacio desconocido mencionadas hasta el momento. Es posible observar que el GVD resultante de solo considerar que las celdas desconocidas no propagan ondas es disconexo, mientras que el GVD generado por otros dos métodos es conexo.



(a) Celdas desconocidas se consideran como libres.

(b) Celdas desconocidas no propagan ondas y $UF \subseteq CGen$.

(c) Celdas desconocidas no propagan ondas.

Figura 3.14: Conectividad del GVD en diferentes formas de considerar el espacio desconocido. El GVD generado se indica en azul.

3.8. Planificación

Una vez que un robot recibe un objetivo de exploración desde la central (sección 3.5), este debe moverse físicamente hasta el lugar indicado para completar dicho objetivo y progresar en la exploración del entorno.

El módulo que se encarga de recibir objetivos de exploración es el *controlador del robot*. Al recibir un objetivo de exploración este módulo recupera el camino

que fue determinado durante la valuación de dicho objetivo (sección 3.5.2) y lo envía al módulo *controlador de movimiento*.

El módulo *controlador de movimiento* interpreta cada punto que compone al camino recibido como un subobjetivo y se encarga de decidir que subobjetivos enviar al módulo *Move Base* que es el responsable de generar las directivas necesarias para que el robot avance hacia dichos subobjetivos. Estos subobjetivos deben enviarse en una secuencia que permita que el robot pueda seguir el camino hasta el objetivo de exploración. Para lograr esto, asumiendo que se tiene un criterio que permite determinar cuando un subobjetivo se considera completado, el funcionamiento del *controlador de movimiento* consiste en determinar el primer subobjetivo no completado del camino, enviar dicho subobjetivo a *Move Base* y esperar a que el robot avance lo suficiente para completarlo. Una vez completado se repite el proceso con el siguiente subobjetivo sin completar, esto sucede hasta que se completa el último subobjetivo del camino que es el objetivo de exploración.

3.8.1. Criterios de compleción

El *controlador de movimiento* tiene dos criterios para determinar que un subobjetivo fue completado, el criterio de compleción normal y criterio de compleción forzosa.

La compleción forzosa ocurre cuando el robot está a una distancia menor a *DistF* del subobjetivo en cuestión. Este criterio de compleción tiene como propósito determinar cuándo un robot está lo suficientemente cerca de un subobjetivo como para considerarlo completo, a pesar de que no se dio aún la compleción normal. Se recomienda que el valor *DistF* sea similar a las dimensiones del robot.

Por otro lado para la compleción normal existen dos condiciones, la primera es que la línea de ancho *An* que existe entre el robot y el subobjetivo, al discretizarse en celdas no contenga ninguna celda ocupada. La segunda, en el caso del objetivo de exploración es que la distancia entre este y el robot sea menor a *DistNo*, y en el caso del resto de los subobjetivos es que la distancia hacia estos sea menor a *DistNs*.

El valor *DistNo* debe cumplir con $DistF \leq DistNo < rango$, siendo *rango* el alcance de los sensores del robot. Esto se debe a que la idea detrás del criterio de compleción normal aplicado al objetivo de exploración, es considerar que este es completado cuando es seguro que fue sensado por el robot. Esto implica que el robot está en condiciones de recopilar la nueva información del entorno asociada al objetivo, siendo este el motivo por el cual el objetivo fue asignado en un primer lugar.

El valor *DistNs* debe cumplir con $DistF \leq DistNs$, y establece qué tan estrictamente se sigue el camino original en espacios despejados de obstáculos. Valores pequeños de *DistNs* hacen que el robot siga el camino original de forma estricta aunque esté en espacios despejados, ya que se fuerza que el robot se acerque a los subobjetivos para completarlos. Al aumentar el valor de *DistNs* se permite mayor libertad en como seguir el camino original cuando el espacio está despejado. Esto sucede porque en los espacios despejados la línea de ancho *An* no se solapa con obstáculos y por lo tanto todos los subobjetivos del camino que estén a una distancia menor a *DistNs* se consideran completados. Completar varios subobjetivos a la vez implica libertad de movimiento, al ser *Move Base* el que se encarga de la trayectoria hasta el primer subobjetivo no completado del camino, sin ser necesario pasar por los subobjetivos previamente completados.

3.8.2. Planificación jerárquica

Los GVD son estructuras que se representan con un cantidad considerablemente menor de celdas que las presentes en las grillas de ocupación sobre las cuales son generados. A pesar de esto los GVD son *roadmaps* (sección 2.2.1) por lo que permiten planificar caminos entre cualquier par de puntos del espacio, al igual que las grillas de ocupación.

La diferencia de tamaños entre las estructuras implica que generar caminos sobre el GVD es computacionalmente más eficiente que generarlos sobre la grilla de ocupación. Pero a su vez, los caminos generados sobre el GVD se mantienen lo más alejado posible de los obstáculos (sección 2.2.3), lo que lleva a caminos que aunque son seguros, pueden resultar innecesariamente largos.

La motivación de la planificación jerárquica es aprovechar la eficiencia de generar caminos sobre el GVD, evitando los desvíos innecesarios que estos caminos contienen.

La planificación jerárquica está basada en un camino de alto nivel construido a partir de información topológica del camino generado sobre el GVD. El camino de alto nivel esta formado por la secuencia de los puntos del camino sobre el GVD que se encuentran en las transiciones entre segmentos (habitaciones y corredores), finalizando con el objetivo. Luego se establecen caminos de bajo nivel entre el robot y el siguiente punto del camino de alto nivel, la idea de estos caminos es que el robot pueda navegar libremente (sin considerar el GVD) dentro de los segmentos, moviéndose de transición en transición. La planificación de los caminos de bajo nivel se delega a *Move Base* que genera caminos sin desvíos innecesarios.

Es pertinente notar que en la implementación realizada *Move Base* se configuró con un *global_planner* (sección 3.2.1) que calcula caminos sobre la grilla de ocupación, de igual manera utilizar la planificación jerárquica simplifica los cálculos de los caminos sobre la grilla, ya que hace que sean locales al segmento que se debe transitar.

Recordando que los caminos no triviales (sección 3.5.2) que llevan a un objetivo de exploración, enviados desde el *controlador del robot* al *controlador de movimiento* son caminos generados sobre el GVD. Para llevar a cabo la planificación jerárquica se establece $DistNs = \infty$ y An mayor al tamaño estimado de los portales de transición entre segmentos (puertas, pasajes abiertos). De esta forma asumiendo segmentos despejados, exceptuando al objetivo de exploración todos los subobjetivos asociados a un segmento en el que se encuentra un robot son completados de forma normal. Por lo tanto en caminos largos que abarcan varios segmentos, nuevamente exceptuando al objetivo de exploración, los únicos subobjetivos enviados a *Move Base* son los que se ubican en las transiciones entre un segmento y el siguiente. Estos subobjetivos no son completados de forma normal ya que dichas transiciones consisten en portales de ancho menor a An que no permiten la existencia de una línea discretizada de ancho An que no contenga obstáculos.

En la figura 3.15 se muestra un ejemplo en el que se puede apreciar la diferencia entre el largo del camino de bajo nivel que lleva a la celda ubicada sobre la transición al siguiente segmento del camino, y el largo de la porción del camino original ubicado sobre el GVD que lleva a esa misma celda.

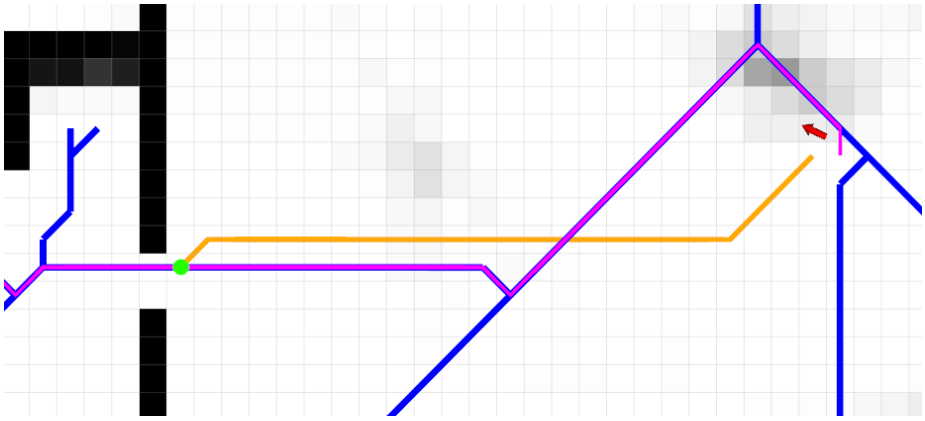


Figura 3.15: Planificación jerárquica. El GVD se indica con azul. En magenta se marca el camino original y en naranja el de bajo nivel, siendo el punto verde el primer subobjetivo sin completar del camino original. El vector rojo indica la posición y orientación del robot.

3.9. Construcción cooperativa del mapa

A lo largo de la exploración los robots utilizan sus sensores para recopilar información del entorno. Esta información sensorial debe ser manejada para aportar a la construcción del mapa, que es representado con una grilla de ocupación (sección 3.3.1) que centraliza toda la información del entorno obtenida por los robots. A continuación se describe el proceso completo desde que el robot obtiene información sensorial, hasta que esta impacta en el mapa.

Cada robot está constantemente recopilando información del entorno a través de sus sensores y enviando dicha información al nodo ROS *global_ costmap* de su módulo *Move Base* (sección 3.2.1). Al recibir información sensorial el nodo *global_ costmap* envía una actualización al módulo *combinador de mapas* que consiste en el conjunto de las celdas c que están en el alcance de la información sensorial recibida y sus probabilidades $P(c|x_t, z_t)$ asociadas. El valor de $P(c|x_t, z_t)$ indica la probabilidad de que una celda c esté ocupada dada la información sensorial z_t obtenida desde la posición x_t. Dependiendo de la evidencia que aporte (x_t, z_t) sobre el estado de c, la probabilidad $P(c|x_t, z_t)$ puede tomar uno de los siguientes valores: $p_{prior} = 0{,}5$ si no aporta evidencia, $p_{occ} = 0{,}55$ si la evidencia indica que la celda está ocupada y $p_{free} = 0{,}45$ si la evidencia indica que la celda está libre. Estos valores de probabilidad son los que se encuentran presentes en la solución desarrollada y se determinaron de forma experimental asegurando que cumplen con (3.10) según se establece en [Stachniss, 2009].

$$0 < p_{free} < p_{prior} < p_{occ} < 1 \tag{3.10}$$

Cuando el módulo *combinador de mapas* recibe una actualización, este aplica la regla de actualización presentada en [Stachniss, 2009] para integrar la información recibida al mapa que mantiene, representado por una grilla de ocupación que concentra toda la información proporcionada por los robots. La regla de actualización para una celda c está dada por (3.11) donde $P(c|x_{1:t-1}, z_{1:t-1})$ es la probabilidad anterior a la actualización, $P(c|x_t, z_t)$ es la probabilidad en la actualización recibida y $P(c|x_{1:t}, z_{1:t})$ es la probabilidad luego de la actualización. En la figura 3.16 se muestra un ejemplo de la aplicación de esta regla de actualización.

$$P(c|x_{1:t}, z_{1:t}) = \left(1 + \frac{1 - P(c|x_t, z_t)}{P(c|x_t, z_t)} \frac{1 - P(c|x_{1:t-1}, z_{1:t-1})}{P(c|x_{1:t-1}, z_{1:t-1})}\right)^{-1} \tag{3.11}$$

Luego de que el módulo *combinador de mapas* integra toda la información recibida en una actualización, este envía la porción actualizada del mapa a los diversos módulos (sección 3.2) que requieren un mapa completo del entorno explorado.

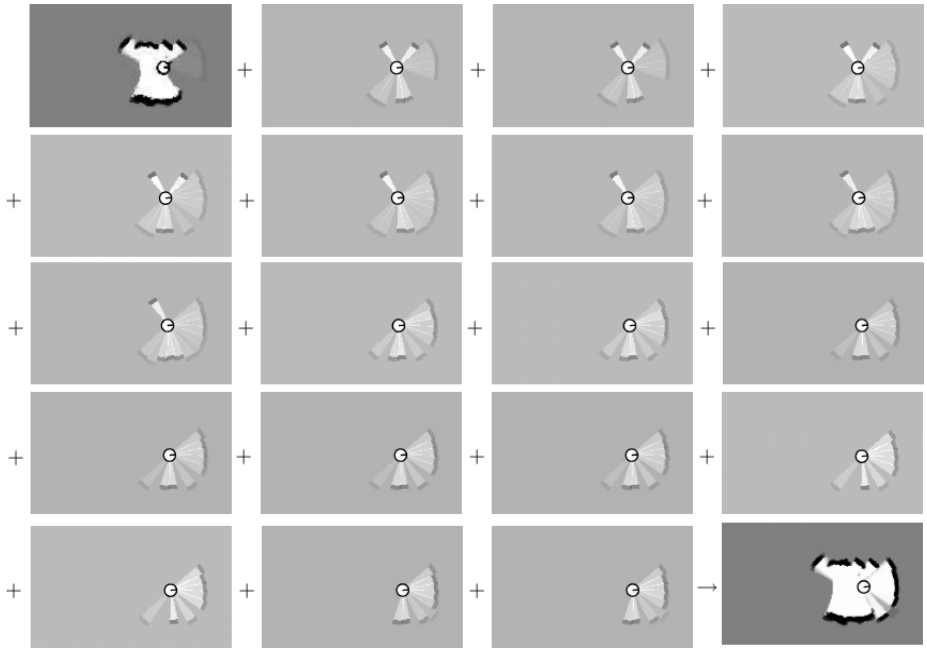


Figura 3.16: Aplicación de la regla de actualización propuesta en [Stachniss, 2009] en un corredor. En la esquina superior izquierda muestra el mapa inicial y en la inferior derecha el mapa resultante. Los mapas intermedios corresponden a las actualizaciones integradas. Extraída de [Stachniss, 2009].

Capítulo 4

Experimentación

Contenidos

4.1. Especificación de las pruebas 59
4.1.1. Simulador 59
4.1.2. Tipos de pruebas 59
4.1.3. Entorno 60
4.1.4. Robots 60
4.1.5. Software 61
4.1.6. Hardware 61
4.2. Métricas 61
4.3. Resultados 62
4.3.1. Completitud y calidad de los mapas 62
4.3.2. Incrementalidad 65
4.3.3. Identificación de objetivos 67
4.3.4. Consideración del espacio desconocido 70

Este capítulo está dedicado a describir las pruebas realizadas y analizar sus resultados. Las pruebas se llevaron a cabo en un simulador y consisten en resolver una instancia del problema de exploración multi-robot con distintas soluciones. Los resultados de las pruebas se presentan y analizan en cuatro secciones. En la sección 4.3.1 se evalúan los mapas construidos en las pruebas. En la sección 4.3.2 se estudia el impacto de construir el GVD de forma incremental. En la sección 4.3.3 se comparan las diversas técnicas de identificación de objetivos. Y finalmente, en la sección 4.3.4 se analiza el efecto de considerar el espacio desconocido de distintas formas.

4.1. Especificación de las pruebas

A lo largo de esta se sección se especifican las pruebas realizadas en este proyecto.

4.1.1. Simulador

Las pruebas fueron simuladas con Gazebo [gazebosim.org, 2021], el cual fue elegido a partir del análisis comparativo entre varios simuladores candidatos realizado en la sección 2.4.

4.1.2. Tipos de pruebas

Las pruebas consisten en resolver una instancia del problema de exploración multi-robot. Específicamente en explorar un entorno cerrado con una flota de robots hasta no detectar más espacio desconocido explorable.

El propósito de las pruebas es evaluar ciertos aspectos de la solución propuesta en este proyecto para dicho problema. Para lograr evaluar cada aspecto por separado se ejecutan pruebas tanto con la solución como fue propuesta originalmente, como con variantes de la misma que cambian únicamente el aspecto a evaluar. Los aspectos de la solución original a evaluar son: la construcción incremental del GVD (sección 3.7). La identificación de objetivos que obtiene fronteras significativas basándose en el cubrimiento (sección 3.4.3). Y la consideración del espacio desconocido al construir el GVD presentada en la sección 3.7.2, donde las celdas desconocidas no propagan ondas y el conjunto UF de celdas desconocidas que son adyacentes a celdas libres pertenecen a generadores ($UF \subseteq CGen$).

Para evaluar la construcción incremental del GVD se compara contra el método de construcción no incremental brushfire (sección 2.2.4). En el caso de la identificación de objetivos se compara contra identificar todas las fronteras como objetivos (sección 3.4.1), y solo identificar como objetivos a las fronteras significativas obtenidas basandose en K-Means (sección 3.4.2). Por último, la consideración del espacio desconocido en la construcción del GVD se compara contra considerar que el espacio desconocido es libre (sección 3.7.2).

Se prueban entonces cinco soluciones, la que utiliza todos los aspectos a evaluar a la vez y las restantes cuatro que difieren de la primera únicamente en uno de dichos aspectos.

Con el motivo de probar distintas cargas computacionales, cada prueba se repite cambiando la granularidad de la grilla de ocupación utilizada para representar el entorno explorado. La granularidad se indica a través de la cantidad de celdas de la grilla que corresponden a un metro cuadrado de la realidad. Los valores de granularidad utilizados para las pruebas son $1, 4, 9$ y 16 celdas por metro cuadrado ($\frac{celdas}{m^2}$).

Adicionalmente debido a que es posible que ejecuciones de un misma simulación lleven a distintos resultados, con el fin de obtener resultados estadísticamente significativos cada prueba fue ejecutada 20 veces. Siendo $5 * 4 * 20 = 400$ el total de pruebas ejecutadas.

4.1.3. Entorno

Las pruebas se realizan en un entorno cerrado que tiene un área explorable de aproximadamente $10000m^2$. El entorno está estructurado en habitaciones, puertas y corredores de varios tamaños. Los únicos obstáculos presentes en él son paredes y los mismos robots que pueden obstaculizarse entre sí. Un mapa de este entorno se muestra en la figura 4.1.

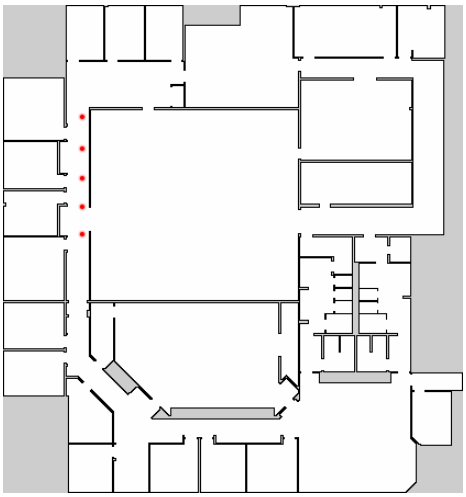


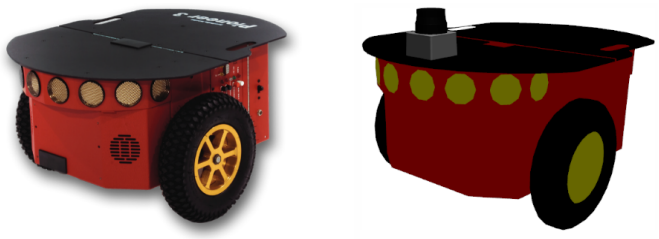
Figura 4.1: Mapa del entorno utilizado en las pruebas. En negro se indican las paredes, en blanco el espacio libre y en gris el espacio inaccesible. Las posiciones iniciales de los robots se indican en rojo.

El entorno fue construido a partir de un modelo que se encuentra disponible por defecto en Gazebo, llamado "Willow Garage", el cual se modifico para reducir el área a explorar y que sea cerrado (sin salidas al exterior).

4.1.4. Robots

Los robots simulados[1] utilizados en las pruebas modelan al robot diferencial Pioneer 3-DX [MobileRobots, 2021] (figura 4.2). Cada robot está equipado con un sensor LiDAR basado en el modelo URG-04LX-UG01 [Hokuyo, 2021], que permite tomar medidas de distancia de hasta 5,6m, con una frecuencia de $10hz$ de un máximo de $36hz$. Adicionalmente el sensor fue alterado para tomar medidas en los 360°al rededor del robot y para proporcionar medidas perfectas (sin ruido).

[1] Especificación disponible en línea:
https://gitlab.fing.edu.uy/federico.ciuffardi/pioneer_p3dx_model.
Accedido por última vez: 25/02/2022.



(a) Real. (b) Simulado.

Figura 4.2: Robot diferencial Pioneer 3-DX.

La flota de exploración se compone de cinco robots ubicados inicialmente en las posiciones indicadas en la figura 4.1.

4.1.5. Software

El software utilizado en las pruebas fue Ubuntu *20.04*, ROS *Noetic* y Gazebo *11.5.1*.

4.1.6. Hardware

El simulador junto al resto de procesos necesarios para llevar a cabo una prueba fueron ejecutados en una computadora personal equipada con un procesador Intel Core i3-9100F, un procesador gráfico GeForce GTX 660 y 16GB de memoria RAM.

4.2. Métricas

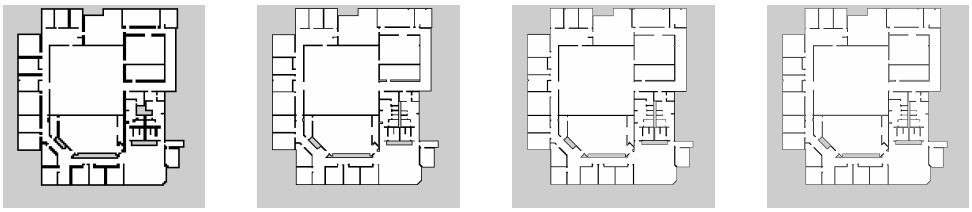
Con el propósito comparar cuantitativamente los resultados de las pruebas se establece un conjunto de métricas. Estas se describen en los que resta de la sección.

Una parte de las métricas utilizadas se proponen en [Yan et al., 2015b] y evalúan el problema de exploración en general, estas son: *tiempo de exploración*, *distancia total recorrida por la flota*, *completitud del mapa* y *calidad del mapa*.

El *tiempo de exploración* refiere al tiempo que ocurre desde que comienza la primera asignación de objetivos (sección 3.5) hasta que no se detecta más espacio desconocido por explorar y se da por terminada la exploración.

La *distancia total recorrida por la flota* es la suma de las distancias recorridas por cada robot de la flota a lo largo de la exploración. En [Yan et al., 2015b] esta métrica se presenta como *costo de exploración*, ya que los autores la consideran como una buena aproximación del costo energético del robot.

Para calcular las métricas de *completitud* y *calidad* de los mapas es necesario contar con mapas de referencia considerados como correctos. En este proyecto estos se representan a través de grillas de ocupación al igual que los generados en la exploración. Se definen cuatro mapas de referencia, uno por cada nivel de granularidad de la grilla utilizado en las pruebas, con el propósito de poder comparar cualquiera de los mapas obtenidos en pruebas con un mapa de referencia de igual granularidad. Los mapas de referencia se pueden apreciar en la figura 4.3.



(a) $1\frac{celda}{m^2}$ (b) $4\frac{celda}{m^2}$ (c) $9\frac{celda}{m^2}$ (d) $16\frac{celda}{m^2}$

Figura 4.3: Mapas de referencia utilizados. En negro se indican las paredes, en blanco el espacio libre y en gris el espacio desconocido por ser inaccesible.

Notar que existe espacio desconocido en los mapas de referencia, esto se debe a que las grillas de ocupación no se ajustan al espacio explorable. En términos de las métricas calculadas, el espacio que es desconocido en los mapas de referencia no es considerado como parte de los mapas, tanto en los mapas de referencia como en los obtenidos en las pruebas.

La *completitud del mapa* se define según (4.1) donde M_{exp} es el área conocida del mapa resultante de la exploración y M_{ref} es el área del mapa de referencia.

$$completitud\ del\ mapa = \frac{M_{exp}}{M_{ref}} \tag{4.1}$$

La *calidad del mapa* se define en (4.2) donde se introduce E_{exp} equivalente al área ocupada por las celdas del mapa resultante de la exploración cuyo estado difiere del estado de su celda correspondiente en el mapa de referencia.

$$calidad\ del\ mapa = \frac{M_{exp} - E_{exp}}{M_{ref}} \tag{4.2}$$

El resto de las métricas fueron diseñadas específicamente para este proyecto, estas son: *tiempo promedio en construcción de GVD* y *tiempo promedio en obtención de fronteras significativas*.

El *tiempo promedio en construcción de GVD* es el tiempo promedio que toma construir el GVD en la etapa de obtención de información de cada asignación de objetivos (sección 3.5).

El *tiempo promedio en obtención de fronteras significativas* es el tiempo promedio que se demora en obtener las fronteras significativas en la etapa de obtención de información de cada asignación de objetivos.

4.3. Resultados

Los resultados experimentales son presentados y analizados en esta sección. Las tablas que se encuentran a lo largo de la sección indican los promedios y desviaciones estándar de las métricas obtenidas en las 20 repeticiones que se hacen debido al no determinismo del simulador.

4.3.1. Completitud y calidad de los mapas

En la tabla 4.1 se muestra la completitud y calidad de los mapas obtenidos en todas las pruebas realizadas.

Variante	$\frac{celdas}{m^2}$	Completitud del mapa	Calidad del mapa
Propuesta sin cambios	1	0.999725±1.5e-04	0.999039±3.1e-04
	4	0.999925±6.5e-05	0.999703±1.7e-04
	9	0.999983±1.2e-05	0.999858±5.7e-05
	16	0.999998±3.6e-06	0.999929±3.4e-05
No incremental	1	0.999763±8.9e-05	0.999208±2.2e-04
	4	0.999957±4.2e-05	0.999805±1.3e-04
	9	0.999984±1.6e-05	0.999864±4.7e-05
	16	0.999999±1.9e-06	0.999941±3.3e-05
Fronteras significativas basadas en K-Means	1	0.999802±1.3e-04	0.999179±2.8e-04
	4	0.999977±2.7e-05	0.999767±8.0e-05
	9	0.999976±2.3e-05	0.999831±7.4e-05
	16	0.999999±2.5e-06	0.999930±1.8e-05
Fronteras	1	0.999937±7.6e-05	0.999493±2.0e-04
	4	0.999995±9.8e-06	0.999837±4.8e-05
	9	0.999983±4.0e-05	0.999729±7.6e-05
	16	0.999998±4.1e-06	0.999921±2.2e-05
Desconocidas se consideran libres	1	0.999812±1.1e-04	0.999256±3.2e-04
	4	0.999972±1.9e-05	0.999786±9.7e-05
	9	0.999987±1.2e-05	0.999840±6.9e-05
	16	0.999998±3.5e-06	0.999940±2.8e-05

Cuadro 4.1: Completitud y calidad de los mapas obtenidos en todas las pruebas realizadas.

Con respecto a la completitud es posible apreciar que no existen diferencias mayores entre las variantes de la implementación, teniendo todos los mapas generados una completitud alta (ver figura 4.4). Dado que el criterio de parada es

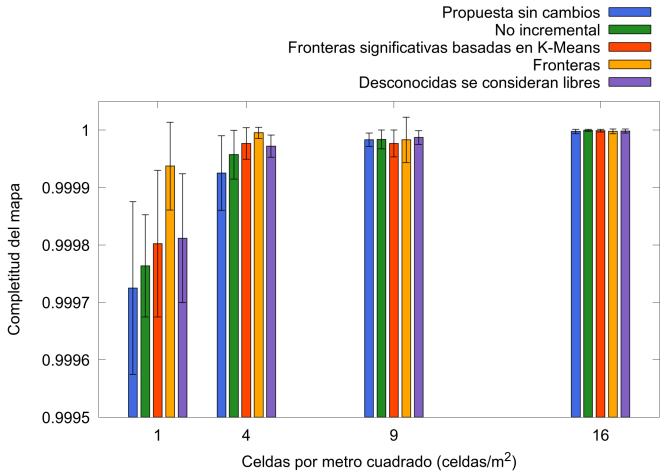
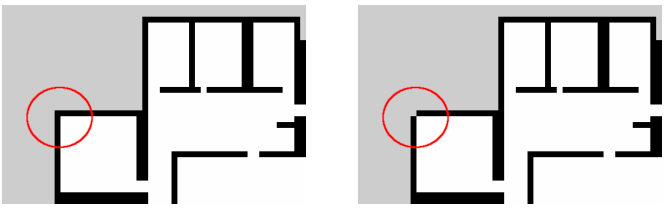


Figura 4.4: Completitud del mapa en función de celdas por metro cuadrado.

que no se detecte más espacio por explorar, estos valores hablan principalmente de la capacidad de detectar el espacio desconocido explorable que tienen las soluciones. Todas las soluciones detectan espacio desconocido explorable mientras exista una un celda c_1 libre y otra c_2 desconocida tal que $c_1 \in ady_4(c_2)$ (sección 3.3.1), principalmente porque en estas condiciones se asume que los robots no pueden atravesar de c_1 a c_2. Dicho esto se presume que la completitud de los

mapas no es perfecta debido a la presencia de celdas ubicadas en esquinas de paredes que a pesar de no cumplir con las condiciones de ser detectadas como explorables, lo son. Un ejemplo de este tipo de celda se muestra en la figura 4.5. Sin embargo los valores de completitud obtenidos indican que este tipo de situaciones son despreciables.



(a) Mapa de referencia. (b) Mapa generado.

Figura 4.5: Esquina superior izquierda del mapa referencia y de uno de los mapas obtenidos en las pruebas que presenta una celda explorable que no se reconoce como tal. Ambos mapas corresponden a una granularidad de una celda por metro cuadrado.

Los resultados de calidad de los mapas obtenidos son similares a los de completitud, los mapas construidos tienen una calidad alta, sin existir diferencias significativas entre las variantes (ver figura 4.6). Esto es esperable ya que los sensores simulados fueron configurados para proporcionar medidas perfectas. Aunque a pesar de esto la calidad de los mapas no es perfecta, esto puede explicarse en primer lugar por el impacto de la completitud del mapa en su calidad. En segundo lugar por la existencia de varios robots que pueden detectase como obstáculos entre sí, generado celdas obstaculizadas que no se encuentran en el mapa de referencia. Y en tercer lugar porque a pesar de la configuración de los sensores, estos proporcionan medidas con un ruido pequeño.

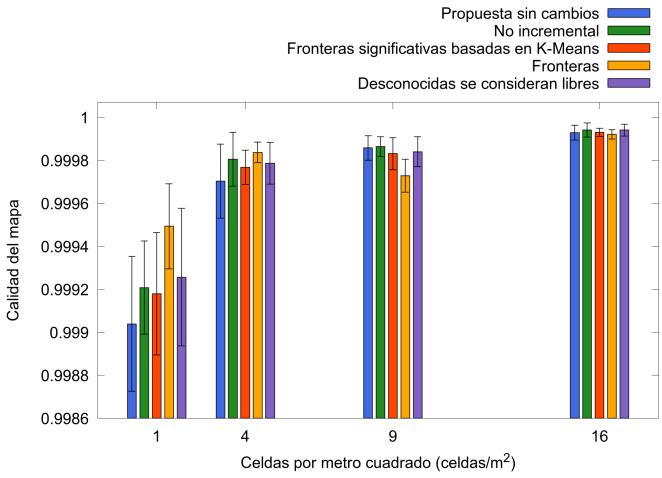


Figura 4.6: Calidad del mapa en función de celdas por metro cuadrado.

Estos resultados de completitud y calidad del mapa permiten concluir que todas las variantes logran resolver de manera satisfactoria el problema de exploración multi-robot en términos de los mapas construidos. Las siguientes secciones se dedican a discutir el impacto de las diferentes variantes en términos de tiempo y costo de exploración.

Como un comentario adicional se puede observar que ambas métricas tienen una tendencia a mejorar al aumentar la granularidad de la grilla. En el caso de la completitud, una explicación posible es que cuanto mayor es la granularidad más pequeñas son las celdas, esto incluye a las celdas ubicadas en las esquinas de paredes que impactan negativamente en el completitud, y por lo tanto como estas celdas son más pequeñas su impacto es menor. En el caso de la calidad, en primer lugar está la incidencia de la completitud en la calidad, y en segundo lugar que detectar un robot como obstáculo solo genera un efecto negativo para la calidad en las celdas que se corresponden a la superficie de ese robot que es percibida por los sensores de otro robot, entonces como celdas más pequeñas representan una superficie con un area menor se tiene que una granularidad mayor lleva a un menor impacto negativo sobre la calidad.

4.3.2. Incrementalidad

En esta sección se comparan los resultados de las pruebas realizadas con la solución propuesta que construye el GVD de forma incremental, contra los de la variante que solo difiere en que la construcción del GVD es no incremental. Las métricas analizadas en esta sección se encuentran en la tabla 4.2.

Construcción del GVD	$\frac{celdas}{m^2}$	Tiempo de exploración (s)	Distancia total recorrida por la flota (m)	Tiempo promedio en construcción de GVD (s)
Incremental	1	463.0±22.6	2480.5±109.7	0.0442±0.0017
	4	496.4±19.0	2745.2±107.3	0.1762±0.0057
	9	549.2±18.4	2849.3±95.2	0.4204±0.0174
	16	678.4±24.8	3050.3±105.2	0.7641±0.0334
No incremental	1	501.9±22.4	2653.4±96.5	0.7199±0.0122
	4	745.7±23.1	2851.1±144.0	2.8603±0.0484
	9	1198.7±35.8	3038.1±159.2	6.8253±0.1713
	16	1856.7±56.2	3117.0±170.3	12.9256±0.3056

Cuadro 4.2: Construcción del GVD: incremental vs. no incremental.

En cada nivel de granularidad de la grilla la variante incremental reduce en un ~94 % el tiempo promedio en construcción del GVD con respecto a la variante no incremental (ver figura 4.7). Esto valida la idea de que la construcción incremental del GVD es más eficiente que su contraparte no incremental.

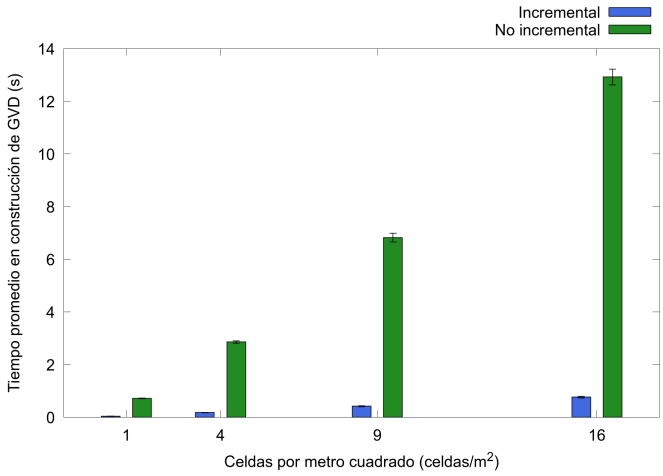


Figura 4.7: Tiempo promedio en construcción de GVD en función de celdas por metro cuadrado.

Con respecto a los tiempos de exploración, estos también son reducidos al construir el GVD de forma incremental. La reducción comienza siendo de ~8 % en el nivel de granularidad de la grilla más bajo, y a medida que aumenta el nivel de granularidad las reducciones también aumentan hasta llegar a una reducción de ~63 % en el nivel más alto (ver figura 4.8). Esto se explica porque el tiempo

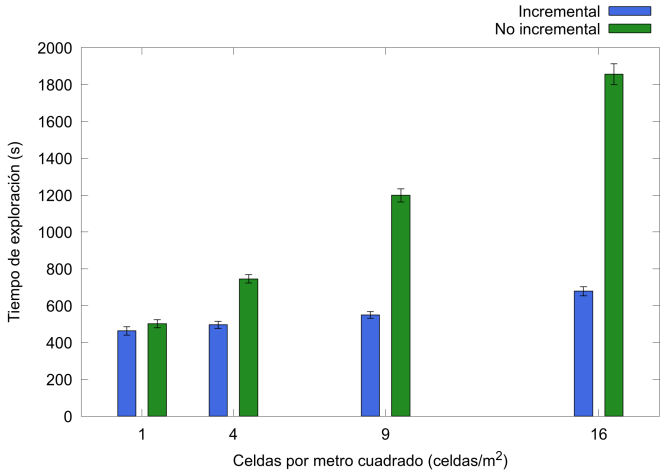


Figura 4.8: Tiempo de exploración en función de celdas por metro cuadrado.

de construcción de GVD impacta directamente al tiempo que ocurre desde que un robot pide un objetivo y uno le es asignado, por ser una parte necesaria y no paralelizada de la etapa de obtención de información (sección 3.5.1) de cada asignación de objetivos (sección 3.5). Y a su vez los tiempos de asignación de objetivos impactan a los tiempos de exploración, pero en este caso el impacto no es directo. Esto se debe a que el hecho de que exista una asignación de objetivos en curso solo asegura que un único robot está ocioso, porque mientras que un robot pide un objetivo y uno le es asignado el resto puede estar completando objetivos que le fueron asignados previamente. Cuanto más corta sea la demora en la asignación de objetivos, más probable es que un único robot quede ocioso, mientras que si dicha demora aumenta, también lo hace la probabilidad de que

en el transcurso de la asignación más robots completen sus objetivos y requieran nuevos objetivos, quedando ociosos.

Construir el GVD de forma incremental también parece reducir las distancias totales recorridas por la flota en todos niveles de granularidad de la grilla, aunque en este caso las reducciones están comprendidas entre 2 % y 7 %, fluctuando al aumentar los niveles de granularidad (ver figura 4.9). Esto puede deberse a que tiempos más rápidos de asignación de objetivos hacen más probable que los robots cambien su trayectoria a un nuevo objetivo antes de llegar a la ubicación exacta del objetivo previamente completado. Esto puede reducir la distancia recorrida por el robot si los caminos hacia el objetivo anterior y el actual no se solapan.

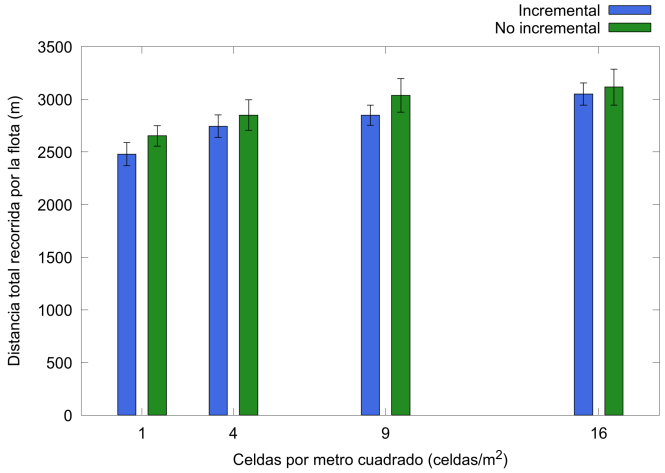


Figura 4.9: Distancia total recorrida por la flota en función de celdas por metro cuadrado.

4.3.3. Identificación de objetivos

En esta sección se realiza un análisis comparativo de los resultados obtenidos en pruebas realizadas con tres soluciones distintas que difieren únicamente en el método usado para identificar objetivos (sección 3.4). Estos métodos son: la obtención de fronteras significativas basada en cubrimiento propuesta en este proyecto, la obtención de fronteras significativas basada en K-Means como es utilizada en [Amorin et al., 2019] y la identificación de todas las fronteras como objetivos introducida en [Yamauchi, 1997]. Los resultados de las métricas analizadas en esta sección se presentan en la tabla 4.3.

Identificación de objetivos	$\frac{celdas}{m^2}$	Tiempo de exploración (s)	Distancia total recorrida por la flota (m)	Tiempo promedio en obtención de fronteras significativas (s)
Fronteras significativas basadas en cubrimiento	1	463.0±22.6	2480.5±109.7	0.0299±0.0021
	4	496.4±19.0	2745.2±107.3	0.1143±0.0076
	9	549.2±18.4	2849.3±95.2	0.2622±0.0149
	16	678.4±24.8	3050.3±105.2	0.5007±0.0434
Fronteras significativas basadas en K-Means	1	487.6±22.1	2567.6±110.0	0.0102±0.0021
	4	490.3±21.6	2689.3±111.4	0.0309±0.0075
	9	537.2±19.3	2831.4±100.8	0.1134±0.0618
	16	642.3±29.0	2987.4±136.3	0.1745±0.1085
Fronteras	1	487.9±24.3	2750.7±142.3	0 ± 0
	4	532.8±24.7	3028.3±136.1	0 ± 0
	9	643.3±22.8	3216.9±121.3	0 ± 0
	16	925.8±104.7	3352.5±208.7	0 ± 0

Cuadro 4.3: Identificación de objetivos: comparación de métodos.

Es posible apreciar que los métodos que solo identifican a las fronteras significativas como objetivos reducen los tiempos de exploración con respecto al método que identifica a todas las fronteras como objetivos. Aunque en el nivel de granularidad de la grilla más bajo las reducciones son menores a ~5 %, estas crecen junto a la granularidad hasta alcanzarse una reducción de ~29 % en el nivel más alto (ver figura 4.10). Esto puede deberse al tiempo de asignación de

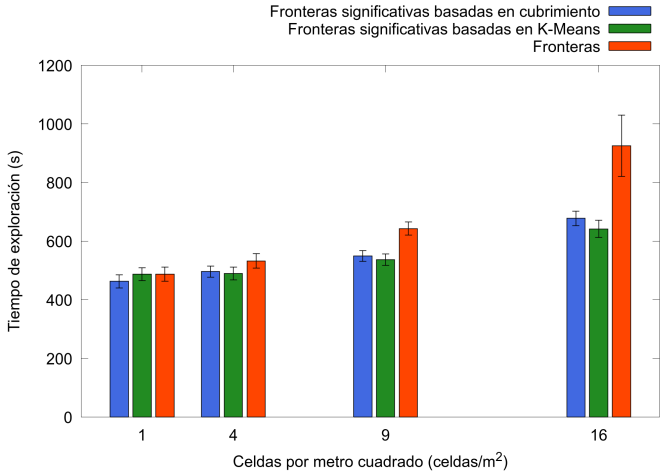


Figura 4.10: Tiempo de exploración en función de celdas por metro cuadrado.

objetivos como se explica en la sección 4.3.2. En este caso se considera que los factores con mayor impacto sobre los tiempos de asignación de objetivos son dos: el tiempo de obtención de fronteras significativas y la cantidad de objetivos identificados. Mayores tiempos de obtención de fronteras significativas retrasan la identificación de objetivos, parte necesaria y no paralelizada de la etapa de obtención de información (sección 3.5.1) en la asignación de objetivos. Mientras que disminuir la cantidad de objetivos identificados implica reducciones en la duración de las etapas de valuación (sección 3.5.2) y de resolución (sección 3.5.3) al ser menor la cantidad de objetivos a valuar y a considerar en la resolución.

Entonces lo que se evidencia en los resultados, es que invertir tiempo en disminuir los objetivos identificados obteniendo las fronteras significativas lleva a una reducción total de los tiempos de asignación de objetivos, y en consecuencia de los tiempos de exploración frente a no invertir dicho tiempo y utilizar todas las fronteras como objetivos. Otra razón posible para las reducciones del tiempo de exploración es que usar solo las fronteras significativas como objetivos puede aportar a la coordinación de los robots. Por ejemplo de usarse todas las fronteras como objetivos se permite que un robot sea asignado a un objetivo adyacente a una pared o que dos robots sean asignados a objetivos adyacentes entre sí, situaciones en las cuales se desaprovecha capacidad de sensado, y se complejiza la navegación. Al usarse solo las fronteras significativas como objetivos, especialmente cuando se logra el cubrimiento con la menor cantidad de fronteras significativas posible, se evitan situaciones como las descritas lo cual puede verse como una coordinación pasiva.

De acuerdo con los resultados usar solo las fronteras significativas como objetivos también reduce las distancias totales recorridas por la flota frente a usar a todas las fronteras. Estas reducciones son en general de ~10 % (ver figura 4.11), y pueden ser causadas por la disminución del tiempo de asignación de objetivos, debido a las mismas razones que se explican en la sección 4.3.2. Aunque la coordinación pasiva también puede estar jugando un rol al evitar que los robots se trasladen a objetivos que desaprovechan sus capacidades sensoriales o que puedan llevar a inconvenientes en la navegación.

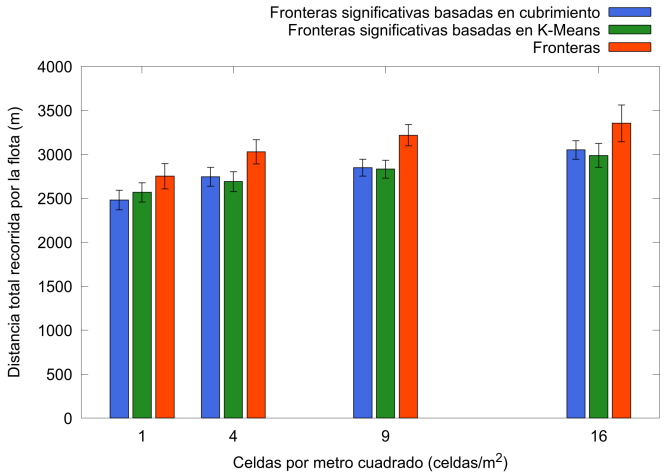


Figura 4.11: Distancia total recorrida por la flota en función de celdas por metro cuadrado.

Con relación a los dos métodos de obtención de fronteras significativas, en el nivel de granularidad de la grilla más bajo, el método basado en cubrimiento logra reducciones con respecto al basado en K-Means de ~3 % en las distancias totales recorridas por la flota y de ~5 % en los tiempos de exploración. Esto se invierte a medida que aumenta el nivel de granularidad, hasta que en el nivel más alto en lugar de reducirse, las métricas aumentan. Las distancias totales recorridas por la flota aumentan en un ~3 % y los tiempos de exploración en un ~6 % (ver figuras 4.10 y 4.11). Por otro lado según los tiempos promedio en obtención de fronteras significativas, el método basado en cubrimiento en todo nivel de granularidad es entre 2 y 4 veces más lento que el basado en K-Means (ver figura 4.12). Estos resultados indican que la obtención de fronteras significativas basada en cubrimiento es menos eficiente que la basada en K-

Means en cuanto al tiempo de ejecución, a pesar de esto no hay diferencias estadísticamente relevantes en los tiempos de exploración ni en las distancias totales recorridas por la flota. Estos resultados no permiten deducir de forma concreta cual método logra mejores reducciones de objetivos.

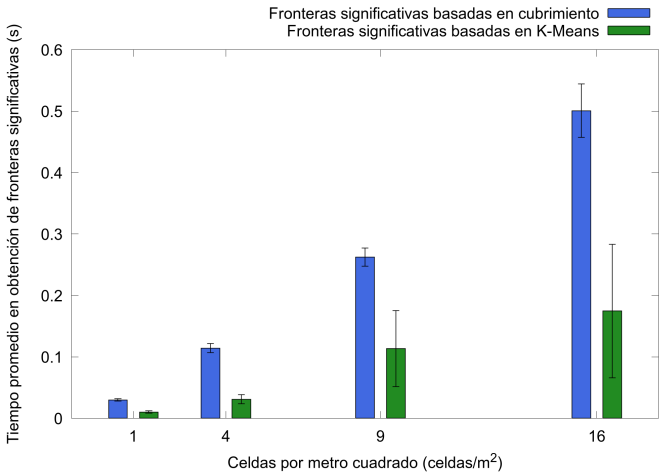


Figura 4.12: Tiempo promedio en obtención de información función de celdas por metro cuadrado.

4.3.4. Consideración del espacio desconocido

Las pruebas analizadas en esta sección corresponden a las realizadas con soluciones que varían únicamente en la forma de considerar el espacio desconocido al construir el GVD. Dichas formas son dos: considerar las celdas desconocidas como libres y considerar que las celdas desconocidas no propagan ondas y que el conjunto UF de celdas desconocidas que son adyacentes a celdas libres pertenecen a generadores ($UF \subseteq CGen$). Ambas formas son descritas en la sección 3.7.2. Los resultados obtenidos en estas pruebas se presentan en la tabla 4.4.

Consideración del espacio desconocido	$\frac{celdas}{m^2}$	Tiempo de exploración (s)	Distancia total recorrida por la flota (m)	Tiempo promedio en construcción de GVD (s)
Desconocidas no propagan olas y $UF \subseteq CGen$	1	463.0±22.6	2480.5±109.7	0.0442±0.0017
	4	496.4±19.0	2745.2±107.3	0.1762±0.0057
	9	549.2±18.4	2849.3±95.2	0.4204±0.0174
	16	678.4±24.8	3050.3±105.2	0.7641±0.0334
Desconocidas se consideran libres	1	479.1±34.4	2522.2±149.5	0.1071±0.0053
	4	510.5±27.1	2728.9±135.2	0.4535±0.0193
	9	677.8±26.4	3134.2±148.8	1.1141±0.0419
	16	955.4±42.1	3303.6±240.5	2.0167±0.1690

Cuadro 4.4: Construcción del GVD: incidencia del espacio desconocido.

Considerar que las celdas desconocidas no propagan ondas y que $UF \subseteq CGen$ logra una reducción de ~61 % en el tiempo promedio de construcción del GVD en cada nivel de granularidad de la grilla respecto a considerar a las celdas desconocidas como libres (ver figura 4.13). Estos resultados validan que

restringir la construcción del GVD al espacio conocido lleva a mejoras en los tiempos de construcción del GVD.

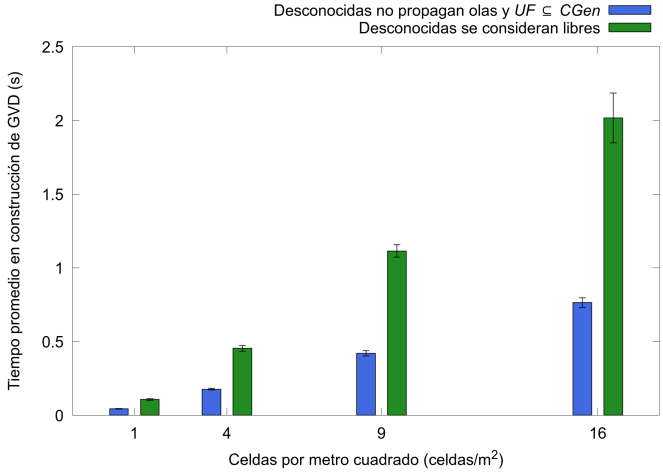


Figura 4.13: Tiempo promedio en construcción de GVD en función de celdas por metro cuadrado.

Adicionalmente tanto el tiempo de exploración como las distancias totales recorridas por las flota son reducidas al considerar que las celdas desconocidas no propagan ondas y que $UF \subseteq CGen$. La reducción del tiempo de exploración comienza siendo de ~3 % en el nivel de granularidad más bajo, llegando a una reducción de ~29 % en el nivel más alto (ver figura 4.14). Mientras que la reducción de la distancia total recorrida por la flota fluctúa entre ninguna reducción y una reducción de ~9 % (ver figura 4.15). Esto puede ser causado porque la reducción de los tiempos de construcción del GVD implica una reducción en los tiempos de asignación de objetivos que se asocia a reducciones en los tiempos de exploración y de las distancias totales recorridas por la flota, como se explica en la sección 4.3.2. Aunque en este caso las reducciones de los tiempos de construcción del GVD son menores y por lo tanto los efectos sobre las métricas son menos pronunciados. A su vez, considerar el espacio desconocido de distinta manera cambia significativamente la forma del GVD lo cual también puede afectar a los caminos que los robots generan a partir del GVD y por lo tanto a las métricas en cuestión.

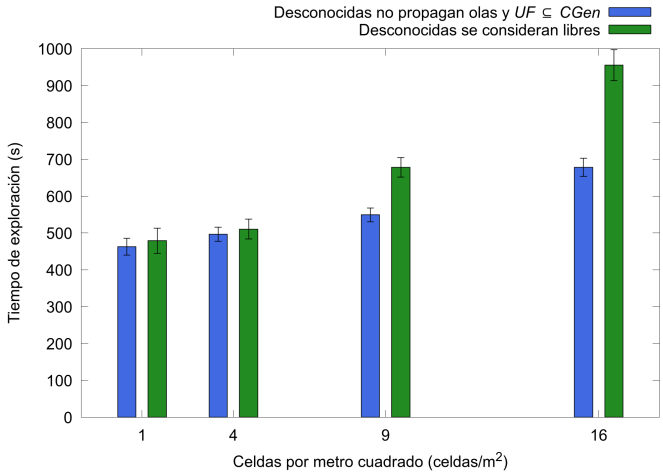


Figura 4.14: Tiempo de exploración en función de celdas por metro cuadrado.

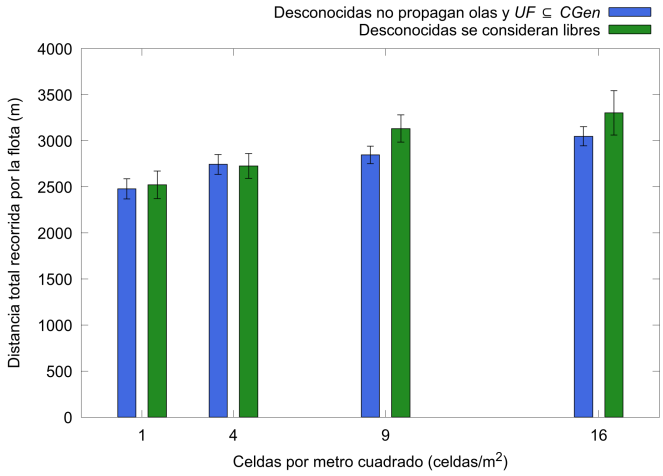


Figura 4.15: Distancia total recorrida por la flota en función de celdas por metro cuadrado.

Capítulo 5

Conclusiones y trabajo futuro

Contenidos

5.1. Conclusiones 74
5.2. Trabajo a futuro 75
5.2.1. Experimentación con robots reales 75
5.2.2. Identificación de objetivos 75
5.2.3. Mapa topológico 76
5.2.4. Algoritmo de asignación de objetivos y Planificación 76
5.2.5. Potenciales mejoras de rendimiento 76
5.2.6. Reusabilidad 76

Este proyecto de grado se planteó el objetivo de mejorar la solución del problema de exploración multi-robot propuesta en [Wurm et al., 2008], donde se propone una estrategia de coordinación para entornos estructurados que consiste en explorar el entorno maximizando la distribución de los robots sobre las habitaciones y corredores (segmentos) que se identifican a partir de un GVD. Para esto se evaluaron varios aspectos de la propuesta en cuestión y del problema de exploración multi-robot en general relevando una serie de potenciales mejoras. Estas fueron implementadas como parte de una solución al problema de exploración multi-robot[1] con la cual se experimentó dentro de un simulador con el propósito de validar su impacto.

En este ultimó capítulo se plantean las conclusiones derivadas del trabajo realizado en el proyecto de grado y se establecen líneas de trabajo que pueden ser continuadas en el futuro.

5.1. Conclusiones

En general, a partir de los resultados obtenidos en la experimentación, se concluye que la solución desarrollada resuelve el problema de exploración multi-robot de forma satisfactoria en términos de los mapas generados.

Por otro lado se puede concluir que es posible realizar la construcción del GVD de forma incremental sin perder la información secundaria necesaria para identificar los segmentos. Es decir, que es posible aplicar la estrategia de coordinación de [Wurm et al., 2008] construyendo el GVD de forma incremental. Esto se logró adaptando el algoritmo presentado en [Lau et al., 2013] para incluir la información necesaria. A su vez los datos experimentales permiten concluir que la construcción incremental del GVD acelera significativamente los tiempos de construcción del GVD y de exploración, frente a realizar una construcción no incremental como en la propuesta original. Esto es una mejora considerable en tanto hace viable aplicar la estrategia de coordinación en entornos de mayor tamaño o que requieran mapas con un mayor nivel de detalle.

También se concluye que el espacio desconocido puede ser considerado de distintas maneras en la construcción del GVD. En especial que puede considerase de forma de limitar la construcción del GVD al espacio conocido, lo cual lleva a una reducción de los tiempos de construcción del GVD y de exploración, con respecto a realizar la construcción en todo el espacio (conocido y desconocido) como es usual.

En relación a la identificación de objetivos, según los resultados experimentales se concluye que utilizar todas las fronteras como objetivos tiene un impacto negativo considerable en los tiempos de exploración y en menor medida en la distancia que recorren los robots, frente a reducir el número de objetivos usando solo las fronteras significativas como objetivos. Respecto a la obtención de fronteras significativas se mencionan dos métodos, la obtención de fronteras significativas basada en K-Means que se encuentra presente en el estado del arte, y la obtención de fronteras significativas basada en cubrimiento que es propuesta en este proyecto como mejora de la anterior. Los resultados experimentales permiten concluir que la obtención de fronteras significativas basada en K-Means es la más eficiente en términos de tiempo de ejecución, mientras que respecto al tiempo de exploración y la distancia total recorrida por la flota no existen diferencias significativas entre los métodos.

Adicionalmente existen dos mejoras potenciales sobre las cuales únicamente se puede concluir que funcionan correctamente por formar parte de la solución

[1] Disponible en línea:
https://gitlab.fing.edu.uy/federico.ciuffardi/pgmappingcooperativo.
Accedido por última vez: 25/02/2022.

implementada, pero que su impacto no fue aislado y estudiado experimentalmente. La primera es el modelo de planificación que combina jerárquicamente la planificación sobre el GVD, y la planificación sobre todo el espacio libre. Esta forma de planificar debería implicar reducciones en las distancias recorridas frente a navegar únicamente sobre el GVD y de costos computacionales frente a calcular los caminos directamente sobre todo el espacio libre. La segunda es el algoritmo de asignación de objetivos que fue propuesto como alterativa al utilizado en el trabajo original de [Wurm et al., 2008]. La principal ventaja de este algoritmo es que logra maximizar la distribución de los robots sobre los segmentos evitando asignar más robots a un segmento que los objetivos contenidos en él.

5.2. Trabajo a futuro

Esta sección se dedica a comentar las líneas de trabajo que surgen del proyecto realizado y pueden ser retomadas a futuro.

5.2.1. Experimentación con robots reales

Una de estas líneas de trabajo es la de realizar experimentos con una flota de robots reales con el propósito de validar los resultados obtenidos en los experimentos simulados. Esto no es una tarea trivial ya que, además de requerirse hardware y un entorno en donde hacer las pruebas, también se deben remover las hipótesis de comunicación y localización ideales de los robots [Amigoni et al., 2017, Khairuddin et al., 2015].

5.2.2. Identificación de objetivos

Si bien los resultados experimentales no permiten concluir que la obtención de fronteras significativas basada en cubrimiento es mejor que la basada en K-Means en términos de reducción de objetivos de exploración identificados, tampoco lo niegan. Esto motiva una línea de trabajo futuro que es la de realizar experimentos dedicados a comprobar cual de los dos métodos es mejor en términos de reducciones de objetivos. Para lograr esto los nuevos experimentos deben reportar una nueva métrica que sea un indicador directo de cantidad de objetivos identificados.

De comprobarse que el método de obtención de fronteras significativas basado en cubrimiento logra mejores reducciones de objetivos se abren dos líneas de trabajo que profundizan en el estudio y desarrollo de este método. La primera está orientada al análisis del método y su implementación en busca de posibles optimizaciones que logren mejorar su rendimiento computacional, que actualmente es su mayor desventaja. La segunda consiste en el desarrollo de variantes del método antes mencionado, que en lugar de determinar de forma arbitraria cual frontera es la significativa del conjunto de candidatas, se determine con algún criterio específico, como por ejemplo elegir la frontera candidata con mayor ganancia de información [Amorin et al., 2019].

De seguir alguna de estas direcciones también sería conveniente diseñar nuevas pruebas que se concentren en evaluar la identificación de objetivos, ya que por ejemplo en el entorno utilizado la mayoría de habitaciones son pequeñas y no son propensas a generar fronteras con una complejidad suficiente como para presentar un desafió a los métodos de obtención de fronteras significativas estudiados.

5.2.3. Mapa topológico

En la solución desarrollada la extensión de los segmentos es la única información necesaria y que se tiene disponible relacionada al mapa topológico. Por lo que una dirección de trabajo posible sería la de desarrollar un método para procesar dicha información con el propósito obtener un mapa topológico completo, como por ejemplo un grafo que contenga un vértice por segmento y las aristas entre dichos vértices implique la adyacencia entre los segmentos.

Evaluar los mapas topológicos generados a partir de la solución desarrollada con las ideas presentadas en la sección 2.2.8 también se presenta como una línea de trabajo interesante. Al igual que el estudio, implementación y validación de técnicas que proponen mejorar la segmentación como las que se describen en [Thrun, 1998, Wurm et al., 2008, Liu et al., 2015].

5.2.4. Algoritmo de asignación de objetivos y Planificación

Otra posible dirección de trabajo que puede ser retomada a futuro es la de realizar pruebas que permitan comprobar el impacto del algoritmo de asignación de objetivos introducido en este proyecto, comparándolo contra con otros algoritmos de asignación de objetivos, como por ejemplo el método húngaro como es utilizado en [Wurm et al., 2008].

De forma similar otro trabajo futuro posible es el de validar el impacto de la planificación jerárquica presentada en este proyecto comparándola con otras estrategias alterativas de planificación.

5.2.5. Potenciales mejoras de rendimiento

Existen varias mejoras potenciales de rendimiento que pueden ser estudiadas, implementadas y validadas como trabajo futuro. A continuación se presentan algunos ejemplos.

Un ejemplo sería el paralelizar la segmentación y la identificación de objetivos en la etapa de obtención de información.

Otro ejemplo consiste en cambiar el algoritmo de segmentación no incremental por una variante incremental de este, como la presentada en [Liu et al., 2015].

Los métodos de obtención de fronteras significativas son no incrementales y cambiarlos por variantes incrementales también es un un ejemplo. Aunque en este caso no se encontró información al respecto por lo que se debería estudiar si esto es posible.

5.2.6. Reusabilidad

Otro trabajo futuro posible es el de refactorizar, organizar y documentar cada módulo de la solución desarrollada, con el propósito de proporcionar dichos módulos como nodos de ROS independientes, que sigan los estándares propuestos por la comunidad de dicha herramienta. De esta forma se facilita la reutilización de lo desarrollado en este proyecto permitiendo alcanzar a más investigadores y estudiantes.

Modificar la solución actual que utiliza la primera versión de ROS (ROS 1) para utilizar en su lugar a su segunda versión (ROS 2), puede ser otra línea de trabajo de interés pensando en la reusabilidad a futuro, ya que ROS 1 termina su soporte en el 2025.

Appendices

Apéndice A

Algoritmos completos

A.1. Obtención de celdas base

El algoritmo 6 es una versión del algoritmo 5 sin las simplificaciones comentadas en la sección 3.7.1.

Algoritmo 6: Obtención de las celdas base CB_c de una celda c

```
   Entrada: c
 1 CB_c := ∅
 2 para cada cA ∈ ady(c) hacer
 3 |  para cada b ∈ gens(cA) hacer
 4 |  |  tolerado := d_c(b) ≤ DD(c) + L
 5 |  |  masCercano := verdadero
 6 |  |  aRemover := ∅
 7 |  |  para cada b' ∈ CB_c hacer
 8 |  |  |  si b' = b ∨ b' ∈ ady(b) entonces
 9 |  |  |  |  si d_c(b) < d_c(b') entonces
10 |  |  |  |  |  aRemover := aRemover ∪ {b'}
11 |  |  |  |  en otro caso
12 |  |  |  |  |  masCercano := masCercano ∧ falso
   |  |  |  |  |  // No es necesario continuar con la iteración
   |  |  |  |  |     de la línea 7 una vez ejecutada esta línea
13 |  |  |  |  fin
14 |  |  |  fin
15 |  |  fin
16 |  |  si tolerado ∧ masCercano entonces
17 |  |  |  CB_c := CB_c − aRemover
18 |  |  |  CB_c := CB_c ∪ {b}
19 |  |  fin
20 |  fin
21 fin
22 devolver CB_c
```

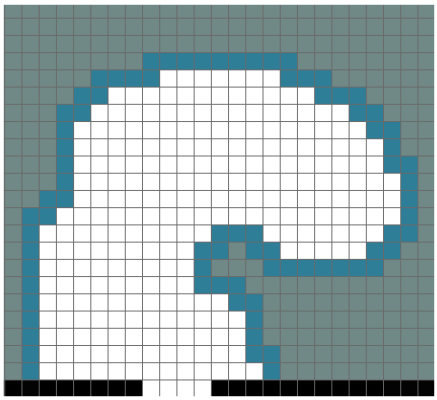
Uno de los principales cambios respecto al algoritmo 5 es el remplazo de la función gen por la función $gens : CG \to P(CGen)$ que dada una celda c de la grilla devuelve el conjunto de todas las celdas que pertenecen a un generador que están a mínima distancia de c. Otro cambio a notar es que la condición asociada a la variable $masCercano$ (líneas 5-15) pasa a ser que la celda base candidata b esté a menor distancia de c que las celdas base actuales b' que son iguales o adyacentes a b. El último cambio a destacar es que la condición que en algoritmo 5 se controla con la variable $noIgNiAdy$, en este algoritmo se controla a partir las variables $masCercano$ y $aRemover$.

Apéndice B

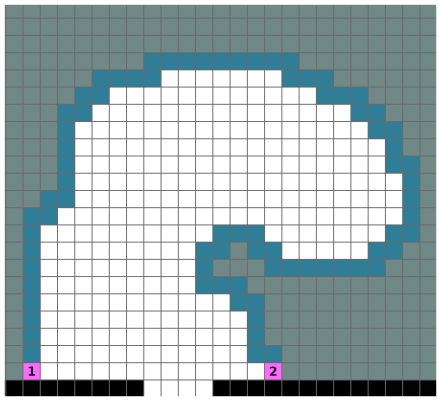
Ejemplos completos

B.1. Obtención de fronteras significativas basada en cubrimiento

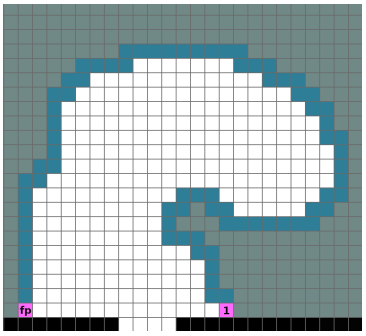
La figura B.1 presenta la versión completa del ejemplo mostrado en la figura 3.7.



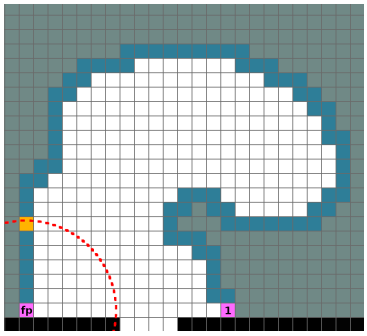
(a) Estado inicial.



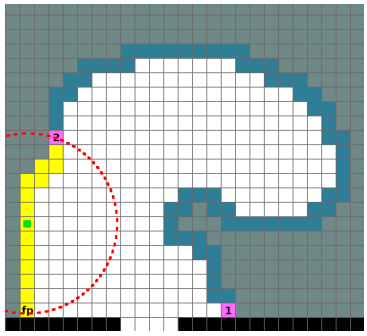
(b) Se inicializa FP.



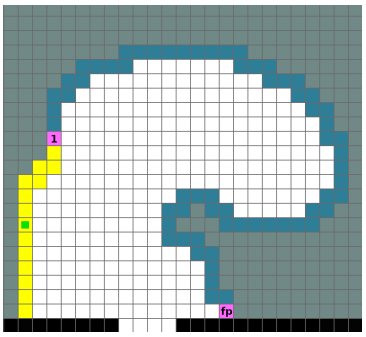
(c) Se obtiene fp desencolando de FP.



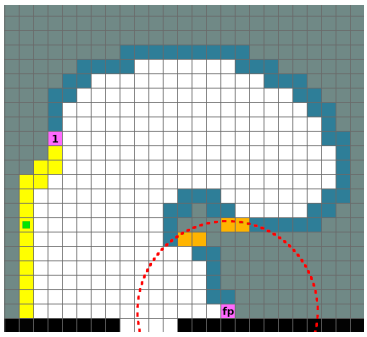
(d) Existen celdas en UF a una distancia mayor que $2 * rango$ de fp, los candidatos se obtienen con $radio = rango$.



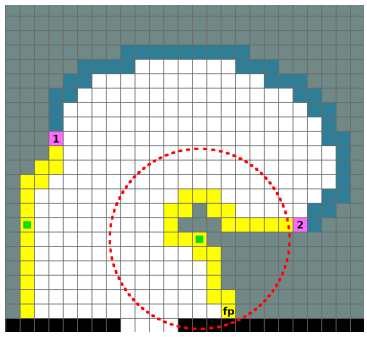
(e) Se elige el único candidato como fs, se actualiza UF, FS_i y FP.



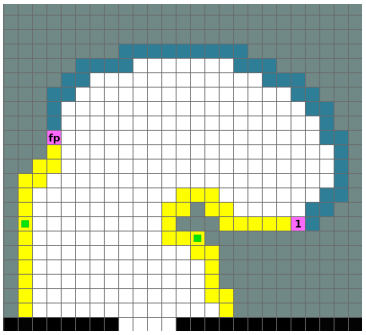
(f) Se obtiene *fp* desencolando de *FP*.



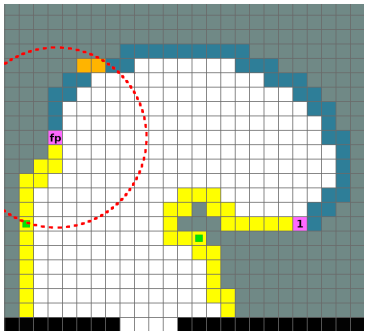
(g) Existen celdas en *UF* a una distancia mayor que $2 * rango$ de *fp*, los candidatos se obtienen con $radio = rango$.



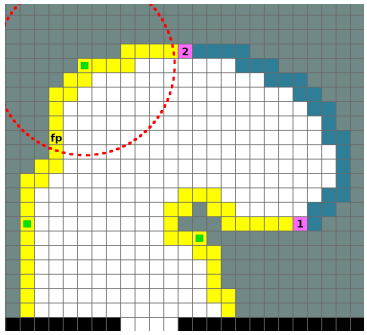
(h) Se elige arbitrariamente un candidato como *fs*, se actualiza *UF*, FS_i y *FP*.



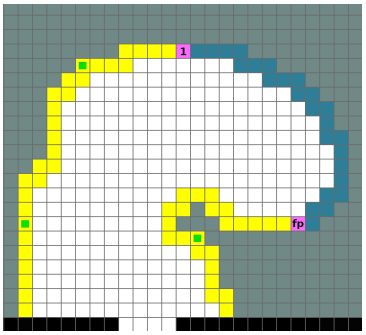
(i) Se obtiene *fp* desencolando de *FP*.



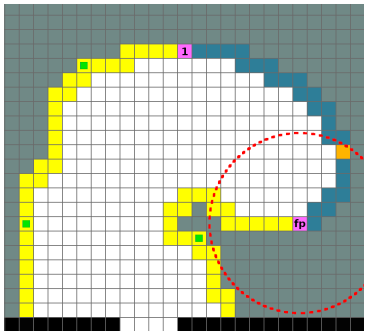
(j) Existen celdas en *UF* a una distancia mayor que $2 * rango$ de *fp*, los candidatos se obtienen con $radio = rango$.



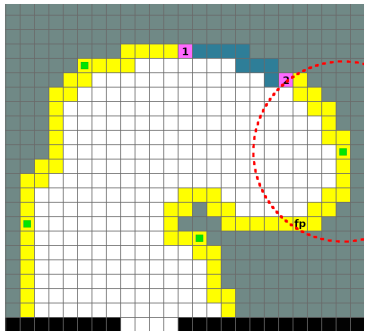
(k) Se elige arbitrariamente un candidato como *fs*, se actualiza *UF*, FS_i y *FP*.



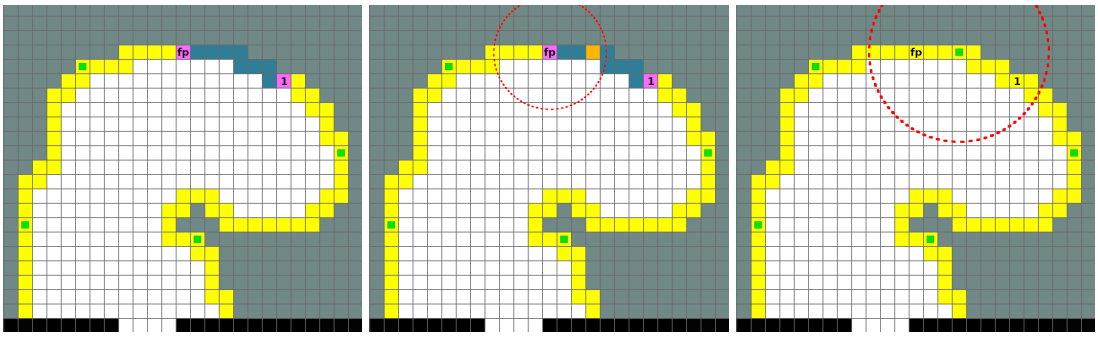
(l) Se obtiene *fp* desencolando de *FP*.



(m) Existen celdas en *UF* a una distancia mayor que $2 * rango$ de *fp*, los candidatos se obtienen con $radio = rango$.



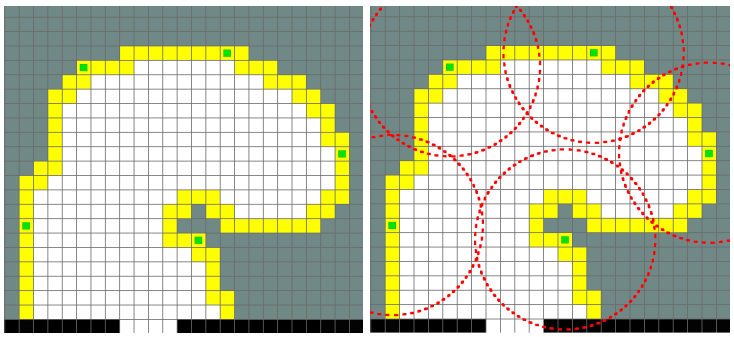
(n) Se elige el único candidato como *fs*, se actualiza *UF*, FS_i y *FP*.



(ñ) Se obtiene *fp* desencolando de *FP*.

(o) No existen celdas en *UF* a una distancia mayor que $2 * rango$ de *fp*, la celda de *UF* más alejada está a $\sim 7{,}4$ (largos de celda), los candidatos se obtienen con $radio \cong 3{,}7$ (largos de celda).

(p) Se elige el único candidato como *fs*, se actualiza *UF*, FS_i y *FP*.



(q) $UF = \emptyset$ por lo que el algoritmo finaliza.

(r) Se logra el cubrimiento.

Figura B.1: Proceso de obtención de fronteras significativas basado en cubrimiento. Las fronteras de F_i se indican con azul si pertenecen a *UF* y con amarillo de lo contrario. Con magenta se indican las celdas en *FP* siendo la numeración su orden en la cola y *fp* la última desencolada. Las circunferencias rojas centradas en *fp* tienen como radio la distancia utilizada para obtener los candidatos *FSC* los cuales se indican con naranja. Las fronteras significativas se indican con verde y las circunferencias rojas centradas en estas tienen un radio de 6 *lados de celda* indicando su cubrimiento al usar sensores de rango equivalente.

Bibliografía

[Amigoni et al., 2017] Amigoni, F., Banfi, J., and Basilico, N. (2017). Multirobot exploration of communication-restricted environments: A survey. *IEEE Intelligent Systems*, 32(6):48–57.

[Amorin et al., 2019] Amorin, G., Benavides, F., and Grampín, E. (2019). A Novel Stop Criterion to Support Efficient Multi-Robot Mapping. In *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, number December, pages 369–374. IEEE.

[Bormann et al., 2016] Bormann, R., Jordan, F., Li, W., Hampp, J., and Hägele, M. (2016). Room segmentation: Survey, implementation, and analysis. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1019–1026. IEEE.

[Bzhikhatlov and Perepelkina, 2017] Bzhikhatlov, I. and Perepelkina, S. (2017). Research of robot model behaviour depending on model parameters using physic engines bullet physics and ode. In *2017 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, pages 1–4. IEEE.

[Cao et al., 1997] Cao, Y. U., Fukunaga, A. S., and Kahng, A. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous robots*, 4(1):7–27.

[Choset et al., 2005] Choset, H. M., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., Thrun, S., and Arkin, R. C. (2005). *Principles of robot motion: theory, algorithms, and implementation*. MIT press.

[Dudek et al., 1996] Dudek, G., Jenkin, M. R., Milios, E., and Wilkes, D. (1996). A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375–397.

[Eitan Marder-Eppstein, 2021] Eitan Marder-Eppstein, David V. Lu, D. H. (2021). Paquete de ros llamado costmap_2d.
http://wiki.ros.org/costmap_2d
Accedido por última vez: 24/11/2021.

[Erez et al., 2015] Erez, T., Tassa, Y., and Todorov, E. (2015). Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4397–4404. IEEE.

[Fermin-Leon et al., 2017] Fermin-Leon, L., Neira, J., and Castellanos, J. A. (2017). Incremental contour-based topological segmentation for robot exploration. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2554–2561. IEEE.

[Foley et al., 1996] Foley, Dam, V., and Feiner (1996). *Introducción a la Graficación por Computador*. Addisson-Wesley Iberoamericana, S.A.

[gazebosim.org, 2021] gazebosim.org (2021). Simulador robótico gazebo.
http://gazebosim.org
Accedido por última vez: 20/11/2021.

[Goldberg and Mataric, 1997] Goldberg, D. and Mataric, M. J. (1997). Interference as a tool for designing and evaluating multi-robot controllers. In *Aaai/iaai*, pages 637–642.

[Guzzoni et al., 1997] Guzzoni, D., Cheyer, A., Julia, L., and Konolige, K. (1997). Many robots make short work: Report of the sri international mobile robot team. *AI magazine*, 18(1):55–55.

[Hartigan and Wong, 1979] Hartigan, J. A. and Wong, M. A. (1979). Ak-means clustering algorithm. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 28(1):100–108.

[Hokuyo, 2021] Hokuyo (2021). Sensor lidar.
https://www.hokuyo-aut.jp/search/single.php?serial=164
Accedido por última vez: 21/12/2021.

[Hopcroft and Tarjan, 1973] Hopcroft, J. and Tarjan, R. (1973). Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378.

[Jiménez, 2021] Jiménez, F. M. (2021). Ejemplo interactivo de diagrama de voronoi.
http://www.geogebra.org/m/VE77tYTS
Accedido por última vez: 22/06/2021.

[Kalra et al., 2009] Kalra, N., Ferguson, D., and Stentz, A. (2009). Incremental reconstruction of generalized voronoi diagrams on grids. *Robotics and Autonomous Systems*, 57(2):123–128.

[Khairuddin et al., 2015] Khairuddin, A. R., Talib, M. S., and Haron, H. (2015). Review on simultaneous localization and mapping (slam). In *2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pages 85–90.

[Kuhn, 1955] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

[Lau et al., 2013] Lau, B., Sprunk, C., and Burgard, W. (2013). Efficient grid-based spatial representations for robot navigation in dynamic environments. *Robotics and Autonomous Systems*, 61(10):1116–1130.

[Liu et al., 2015] Liu, M., Colas, F., Oth, L., and Siegwart, R. (2015). Incremental topological segmentation for semi-structured environments using discretized GVG. *Autonomous Robots*, 38(2):143–160.

[Lopez-Perez et al., 2018] Lopez-Perez, J. J., Hernandez-Belmonte, U. H., Ramirez-Paredes, J.-P., Contreras-Cruz, M. A., and Ayala-Ramirez, V. (2018). Distributed Multirobot Exploration Based on Scene Partitioning and Frontier Selection. *Mathematical Problems in Engineering*, 2018:1–17.

[Lu, 2021] Lu, D. (2021). Paquete de ros llamado global_planner.
http://wiki.ros.org/global_planner
Accedido por última vez: 24/11/2021.

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

[MacQueen et al., 1967] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

[Magyar, 2021] Magyar, B. (2021). Paquete de ros que facilita el control de robots diferenciales.
http://wiki.ros.org/diff_drive_controller
Accedido por última vez: 20/11/2021.

[Marder-Eppstein, 2021a] Marder-Eppstein, E. (2021a). Paquete de ros llamado move_base.
http://wiki.ros.org/move_base
Accedido por última vez: 24/11/2021.

[Marder-Eppstein, 2021b] Marder-Eppstein, E. (2021b). Paquete de ros llamado navigation.
http://wiki.ros.org/navigation
Accedido por última vez: 24/11/2021.

[Mart et al., 2006] Mart, O., Rottmann, A., Triebel, R., and Burgard, W. (2006). Semantic labeling of places using information extracted from laser and vision sensor data. *researchgate*.

[Meijster et al., 2002] Meijster, A., Roerdink, J. B., and Hesselink, W. H. (2002). A general algorithm for computing distance transforms in linear time. In *Mathematical Morphology and its applications to image and signal processing*, pages 331–340. Springer.

[MobileRobots, 2021] MobileRobots, A. (2021). Robot diferencial.
https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf
Accedido por última vez: 21/12/2021.

[Nieto-Granda et al., 2014] Nieto-Granda, C., Rogers III, J. G., and Christensen, H. I. (2014). Coordination strategies for multi-robot exploration and mapping. *The International Journal of Robotics Research*, 33(4):519–533.

[Nogueira, 2014] Nogueira, L. o. E. a. E. d. C. (2014). Comparative Analysis Between Gazebo and V-REP Robotic Simulators. *Seminario Interno de Cognicao Artificial-SICA*.

[Pitonakova et al., 2018] Pitonakova, L., Giuliani, M., Pipe, A., and Winfield, A. (2018). Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10965 LNAI, pages 357–368. Springer International Publishing.

[Ramli et al., 2015] Ramli, N. R., Razali, S., and Osman, M. (2015). An overview of simulation software for non-experts to perform multi-robot experiments. In *2015 International Symposium on Agents, Multi-Agent Systems and Robotics (ISAMSR)*, pages 77–82. IEEE.

[Rönnau et al., 2013] Rönnau, A., Sutter, F., Heppner, G., Oberländer, J., and Dillmann, R. (2013). Evaluation of physics engines for robotic simulations with a special focus on the dynamics of walking robots. In *2013 16th International Conference on Advanced Robotics (ICAR)*, pages 1–7. IEEE.

[Rösmann, 2021] Rösmann, C. (2021). Paquete de ros llamado teb_local_planner.
http://wiki.ros.org/teb_local_planner
Accedido por última vez: 24/11/2021.

[Santos Pessoa de Melo et al., 2019] Santos Pessoa de Melo, M., Gomes da Silva Neto, J., Jorge Lima da Silva, P., Natario Teixeira, J. M. X., and Teichrieb, V. (2019). Analysis and Comparison of Robotics 3D Simulators. In *2019 21st Symposium on Virtual and Augmented Reality (SVR)*, pages 242–251. IEEE.

[Schapire and Singer, 1999] Schapire, R. E. and Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336.

[Schuster et al., 2019] Schuster, M. J., Brunner, S. G., Bussmann, K., Büttner, S., Dömel, A., Hellerer, M., Lehner, H., Lehner, P., Porges, O., Reill, J., et al. (2019). Towards autonomous planetary exploration. *Journal of Intelligent & Robotic Systems*, 93(3):461–494.

[Solanas and Garcia, 2004] Solanas, A. and Garcia, M. A. (2004). Coordinated multi-robot exploration through unsupervised clustering of unknown space. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1:717–721.

[Stachniss, 2009] Stachniss, C. (2009). *Robotic mapping and exploration*, volume 55. Springer.

[Thrun, 1998] Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71.

[Vardy, 2021] Vardy, A. (2021). A ros plugin for the argos robot simulator, enabling large-scale swarm robotic simulations using ros.
http://github.com/BOTSlab/argos_bridge
Accedido por última vez: 20/11/2021.

[Wallgrün, 2005] Wallgrün, J. O. (2005). Autonomous Construction of Hierarchical Voronoi-Based Route Graph Representations. In *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, volume 3343, pages 413–433. Springer, Berlin, Heidelberg.

[Wu et al., 2007] Wu, L., Ángel García García, M., Valls, D. P., and Ribalta, A. S. (2007). Voronoi-based space partitioning for coordinated multi-robot exploration. *Journal of Physical Agents*, 1(1):37–44.

[Wurm et al., 2008] Wurm, K. M., Stachniss, C., and Burgard, W. (2008). Coordinated multi-robot exploration using a segmentation of the environment. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1160–1165. IEEE.

[www.buenosaires.gob.ar, 2021] www.buenosaires.gob.ar (2021). Mapa del subte y combinaciones.
https://www.buenosaires.gob.ar/subte/mapa-y-combinaciones
Accedido por última vez: 15/06/2021.

[www.fsf.org, 2021] www.fsf.org (2021). Free software foundation.
http://www.fsf.org
Accedido por última vez: 20/11/2021.

[www.google.com/maps, 2021] www.google.com/maps (2021). Google maps.
https://www.google.com/maps/
Accedido por última vez: 16/06/2021.

[www.ros.org, 2021] www.ros.org (2021). Robot operating system.
http://www.ros.org
Accedido por última vez: 20/11/2021.

[Yamauchi, 1997] Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*, pages 146–151. IEEE.

[Yan et al., 2015a] Yan, Z., Fabresse, L., Laval, J., and Bouraqadi, N. (2015a). Metrics for performance benchmarking of multi-robot exploration. *IEEE International Conference on Intelligent Robots and Systems*, 2015-Decem:3407–3414.

[Yan et al., 2015b] Yan, Z., Fabresse, L., Laval, J., and Bouraqadi, N. (2015b). Metrics for performance benchmarking of multi-robot exploration. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3407–3414. IEEE.

[Zhang and Suen, 1984] Zhang, T. and Suen, C. Y. (1984). A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239.

[Zivkovic et al., 2006] Zivkovic, Z., Bakker, B., and Krose, B. (2006). Hierarchical map building and planning based on graph partitioning. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 803–809. IEEE.