Análisis de estrategias de Path Planning para equipos de múltiples robots móviles

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

Gonzalo Menéndez, Lucas Bruzzone, Agustina Salmantón, Lucas Corazza

Supervisor
Facundo Benavides

Montevideo, Uruguay
2022

Resumen de Trabajo

Los sistemas robóticos móviles se han vuelto cada día más populares y es de interés que un equipo de robots sea capaz de trabajar en paralelo. Esto permite aumentar la productividad paralelizando el trabajo y realizando tareas que requieren más de un robot trabajando simultáneamente, pero a su vez conlleva la dificultad de resolver problemas de coordinación y de evitar posibles colisiones entre robots. En tal sentido es de vital importancia que los robots puedan navegar en un entorno de manera autónoma y segura.

En este proyecto se presentan 3 estrategias para equipos de múltiples robots móviles, una estrategia con enfoque cooperativo y dos con enfoque independiente. Entre las independientes tenemos una estrategia puramente reactiva y una estrategia híbrida a la que se le agrega un paso previo de planificación del camino.

La estrategia reactiva utiliza campos de potenciales en donde cada robot actúa independientemente del resto, basándose en la información que recibe a través de sus sensores utilizando fuerzas atractivas, repulsivas y tangenciales. La estrategia cooperativa utiliza el algoritmo de búsqueda A* sobre una grilla de ocupación bidimensional que obtiene para cada robot los caminos óptimos libres de colisiones con obstáculos estáticos del ambiente. A partir de estos caminos, se realiza una coordinación para adaptar la forma en que los robots recorren estos caminos, evitando así que ocurran colisiones entre ellos. La estrategia híbrida utiliza A* para obtener los caminos óptimos al destino pero los robots llevan a cabo el camino sin coordinación, utilizando campos potenciales para evitar choques.

La implementación de las estrategias es realizada en el entorno ROS (C++) y las pruebas de las mismas se llevan a cabo utilizando el simulador Webots. Para evaluar el desempeño se realizaron pruebas automáticas para distintos tipos de escenarios, obteniendo un conjunto de métricas (tiempo promedio de resolución, tiempo promedio de procesamiento, distancia recorrida promedio).

Teniendo en cuenta los análisis realizados, cada estrategia maximiza su potencial en diferentes entornos dependiendo de las restricciones del problema y características del mismo.

Índice general

Índice de figuras

Índice de cuadros

Índice de Pseudocódigos

Capítulo 1

Introducción

1.1. Motivación

Actualmente la robótica forma parte de nuestro día a día y se ha involucrado en la sociedad invadiendo muchos aspectos de nuestra vida. La robótica es la ciencia y tecnología para diseñar y construir máquinas capaces de imitar las tareas humanas e incluso nuestra capacidad de decisión. Se ocupa del diseño, manufactura, aplicaciones, y combina diversas disciplinas como la mecánica, la electrónica, la informática, la inteligencia artificial y la ingeniería de control.
Una de las tareas fundamentales de la robótica es planificar movimientos libres de colisiones para cuerpos complejos, desde el principio de una trayectoria hasta su fin. Este problema, llamado *Collision Free Path Planning*, es difícil computacionalmente y cuenta con dificultades como limitaciones mecánicas, sensoriales, de retroalimentación y distintas restricciones [13].

Este proyecto se centrará en el problema anteriormente mencionado, el cual tiene como objetivo encontrar una secuencia de configuraciones para lograr el movimiento de un conjunto de robots desde sus puntos origen hasta sus respectivos puntos destino. A su vez, la utilización del trabajo en paralelo entre robots trae como requisito que cada robot debe moverse evitando colisionar con el resto de los robots [14]. Este tipo de trabajo cooperativo se utiliza en distintos rubros, como el transporte de materiales de fábrica y la robótica agrícola

1.2. Objetivos

Se focalizará en la navegación de los robots en áreas compartidas. Si bien se pueden usar varios métodos y enfoques para abordar este problema, el objetivo del proyecto será la implementación y el estudio de tres algoritmos, donde uno ejecuta de manera cooperativa y los otros dos de manera independiente.

Se implementarán estos algoritmos y se estudiará su comportamiento utilizando un simulador.

Se pretende comparar los algoritmos, obtener métricas y evaluar sus puntos fuertes y débiles. Se discutirán los problemas, limitaciones encontradas y posibles mejoras a futuro para cada una de las estrategias. Se demostrará que las estrategias son muy distintas entre sí, teniendo cada una sus respectivas fortalezas dentro de un entorno de requisitos y usabilidad.

1.3. Organización general del documento

Capitulo 2
Presentación de conceptos fundamentales de la robótica, especificación de las estrategias independiente y cooperativa, y ambiente de desarrollo utilizado.

Capitulo 3
Implementación de estrategias, especificación del funcionamiento y decisiones tomadas, diseño de la arquitectura, análisis y algoritmos.

Capitulo 4
Experimentación de los algoritmos. Especificación de los casos de prueba, obtención de métricas y análisis de resultados.

Capitulo 5
Conclusiones y trabajo a futuro.

Capítulo 2

Marco de Trabajo

Contenido

En este capítulo se presentan los conocimientos previos necesarios referentes a conceptos teóricos del problema a resolver, antecedentes académicos relacionados y las herramientas utilizadas.

2.1. Conceptos Fundamentales [1]

A continuación se describirán los conceptos habitualmente utilizados en la formulación del problema *Path Planning* (*Collision Free Path Planning*).

2.1.1. Paradigmas de robótica

Los paradigmas son modelos que definen la manera en que un robot opera. Se clasifican a partir de la relación que hay entre las 3 primitivas definidas en el Cuadro 2.1 (Sensar, Planificar y Actuar). Cada primitiva realiza una función, donde hay un conjunto de información o datos de entrada y una salida.

Primitiva	**Entrada**	**Salida**
Sensar	Datos de sensores	Información sensada
Planificar	Información	Directivas
Actuar	Información sensada o directivas	Comandos a los actuadores

Cuadro 2.1: Primitivas robóticas

Se definen 3 tipos de paradigmas [21] para clasificar el sistema:

El paradigma **reactivo** (Figura 2.1) o basado en comportamientos no posee un componente de planificación. Este paradigma se puede presenciar en la naturaleza de manera habitual. Por ejemplo es aplicable a los insectos, ya que los comportamientos de los mismos se pueden entender como reacciones ante estímulos captados por sus sentidos sin la presencia de una etapa de planificación.

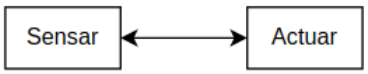


Figura 2.1: Paradigma reactivo

El paradigma **jerárquico** o **deliberativo** (Figura 2.2) se caracteriza por planificar antes de actuar. Un ejemplo de la aplicación de este paradigma se puede dar en el caso que un robot deba utilizar un mapa para llegar a su destino. Este robot podría sensar para ubicarse en el mapa y calcular un plan para llegar a su destino. Una vez calculado el plan se ejecutan movimientos para llevarlo a cabo.

[1] Las definiciones presentadas en esta sección provienen mayormente de [13].

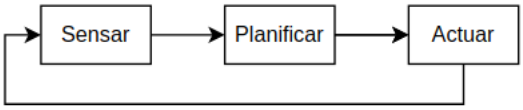


Figura 2.2: Paradigma jerárquico

El paradigma **híbrido** (Figura 2.3) se caracteriza por ejecutar siguiendo un paradigma reactivo y un paradigma jerárquico, dependiendo de la lógica del algoritmo. Siguiendo el ejemplo presentado para el paradigma jerárquico, si además el robot tuviera que evitar obstáculos no conocidos en el mapa, el robot chocaría. Utilizando el paradigma híbrido el robot podría además de ejecutar un plan, sensar durante la ejecución del mismo para adaptar el plan y evitar chocar contra los obstáculos desconocidos.

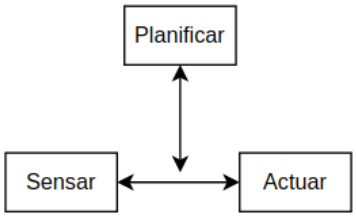


Figura 2.3: Paradigma híbrido

2.1.2. Formulación del problema para un robot

Para la solución del problema *Path Planning* se proporciona una descripción completa de la geometría de un robot A y de un espacio de trabajo W, siendo W un un entorno estático (de dos o tres dimensiones) con obstáculos.

El objetivo es encontrar una ruta libre de colisiones para que el robot A pueda moverse desde una posición y orientación inicial a una posición y orientación final. Para lograrlo se debe especificar la ubicación de cada punto en la geometría del robot o una configuración q.

El **espacio de configuración** C, donde $q \in C$, corresponde al espacio de todas las posibles configuraciones y representa todas las transformaciones aplicables a un robot dada su cinemática, es decir, las formas en que se pueden mover cada una de las partes del robot (Figura 2.4). La dimensión del espacio de configuración de un robot y los grados de libertad de movimiento del mismo no son independientes entre sí: si un robot tiene n grados de libertad, entonces su espacio de configuración tiene n dimensiones.

Sea el conjunto cerrado $O \subset W$ la región del espacio de trabajo ocupada por obstáculos que limitan el movimiento del robot, y el conjunto cerrado $A(q) \subset W$ que denota el conjunto de puntos del espacio de trabajo ocupados por el robot en la configuración $q \in C$.

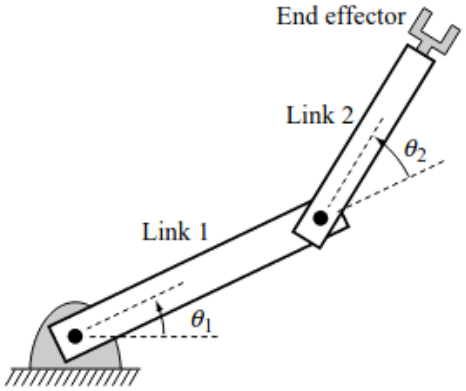


Figura 2.4: Descripción geométrica de un robot A : los ángulos θ_1 y θ_2 especifican la configuración del robot, el cual es una composición de dos robots. Extraído de [13]

El **conjunto de configuraciones que tienen obstáculos del espacio C** (C_{obs}) es definida como:

$$C_{obs} = \{q \in C | A(q) \cap O \neq \emptyset\}$$

El conjunto de configuraciones que evitan las colisiones es $C_{free} = C \setminus C_{obs}$ y es denominado **espacio libre**.

Un camino es una curva en el espacio de configuración libre C_{free} conectando c_i y c_j. (Figura 2.5)

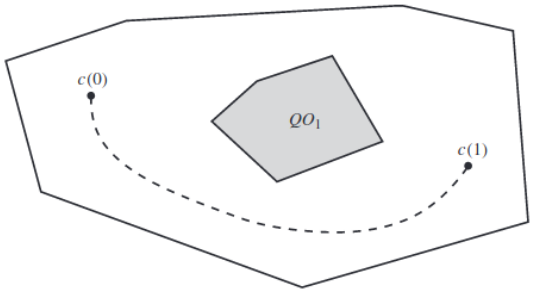


Figura 2.5: Ejemplo de camino entre c(0) y c(1). QO_1 representa la región de obstáculos C_{obs}. Extraído de [1]

Podemos clasificar los algoritmos que encuentran soluciones al problema de *Path Planning* de acuerdo a la **completitud** (se garantiza encontrar una solución si existe), **complejidad** (requerimientos computacionales para el proceso de búsqueda) y **optimalidad** (la calidad de las soluciones encontradas). Frecuentemente, las técnicas más completas y óptimas, requieren un mayor esfuerzo computacional, limitando su utilidad práctica, mientras que las técnicas computacionalmente viables suelen sacrificar optimalidad y/o completitud.

Para planear los caminos en un ambiente, se utiliza una estructura de datos para contener la información, obteniendo un mapa. Los mapas se pueden clasificar en **topológicos** (grafos donde cada nodo representan un dato y las aristas una relación entre los datos), **geométricos** (usan primitivas geométricas para representar el entorno) y **grillas de ocupación** (cada celda asigna una probabilidad de ocupación a la posición correspondiente). Un tipo de mapa topológico muy utilizado es **roadmap** que consiste en representar mediante nodos a puntos específicos en el espacio de configuración y mediante las aristas a caminos conocidos que los conectan.

2.1.3. Múltiples Robots[2]

El problema de *Path Planning* se puede extender a múltiples robots.
Supongamos que tenemos m robots. Se define el espacio que considera las configuraciones simultáneas de todos los robots como:

$$X = C^1 * C^2 * ... * C^m$$

Un estado $x \in X$ especifica las configuraciones de cada uno de los robots, y puede estar expresado como $x = (q^1, q^2, ..., q^m)$.
La dimensión de X es N, donde $N = \sum_{i=1}^{m} dim(C^i)$.

[2]Las definiciones presentadas en esta sección proviene mayormente de [14].

Hay dos posibles regiones de obstáculos en el espacio de estados:

colisiones robot-obstáculo

colisiones robot-robot

Para cada i tal que $1 \leq i \leq m$, siendo m el total de robots, el subconjunto de X que corresponde al robot A^i en colisión con la región obstáculo O es:

$$X^i_{obs} = \{x \in X | A^i(q^i) \cap O \neq \emptyset\} \tag{2.1}$$

Para cada par de robots A^i y A^j, con $i \neq j$ el subconjunto de X que corresponde a A^i en colisión con A^j es:

$$X^{ij}_{obs} = \{x \in X | A^i(q^i) \cap A^j(q^j) \neq \emptyset\} \tag{2.2}$$

La región de obstáculos $X_{obs} \subset X$ se obtiene al combinar (2.1) y (2.2).

$$X_{obs} = \left(\bigcup_{i=1}^{m} X^i_{obs} \right) \cup \left(\bigcup_{ij, i \neq j} X^{ij}_{obs} \right) \tag{2.3}$$

Una vez realizadas estas definiciones, se puede aplicar cualquier algoritmo de planificación de caminos de propósito general (p.e. A^*, el cual se explica en la siguiente sección) para determinar una planificación de movimientos para toda la flota de robots. Sin embargo, buscar un camino en este espacio se vuelve costoso ya que aumenta rápidamente la dimensionalidad del espacio de búsqueda al incorporar más robots, por lo que se han buscado distintas estrategias para encontrar soluciones de una forma más práctica.

2.1.4. Algoritmo A*

El algoritmo A^* presentado en [11] es un algoritmo de búsqueda de caminos en grafos que garantiza, siempre que se cumplan algunas condiciones, encontrar el camino de menor costo entre dos nodos de un grafo ponderado. El algoritmo además analiza la mínima cantidad de nodos necesaria para encontrarlo.
Se puede tener un conjunto de metas y el algoritmo buscará el camino hasta la que se pueda llegar con menor costo.
El algoritmo parte del nodo inicial y se expande explorando los nodos vecinos. Para cada uno de los nodos explorados se guarda el costo mínimo de llegar hasta el mismo desde el nodo inicial, y el nodo anterior en el camino que genera ese costo. De esta forma, una vez que el algoritmo termine en la meta, se puede computar el camino desde la meta hasta el nodo inicial recorriendo el camino de menor costo hacia atrás.
Para poder evaluar la menor cantidad de nodos posible durante la búsqueda, se deben tomar decisiones tan informadas como sea posible a la hora de elegir qué nodo evaluar. Para esto se utiliza alguna función de evaluación $f(n)$ que

permita estimar el costo desde un nodo n hasta la meta. Entonces en cada paso se elige el nodo n que presenta el menor valor de f, entre los nodos vecinos a los explorados.
El pseudocódigo 2.1 presenta los pasos del algoritmo A^*, siendo s el nodo inicial y T el conjunto de nodos meta.

Pseudocódigo 2.1: Algoritmo de búsqueda A^*

```
1    Se marca s como abierto y se calcula f(s).
2    Se selecciona el nodo abierto n cuyo valor de f sea
       menor. Los empates se resuelven de forma arbitraria,
       pero siempre a favor de los nodos pertenecientes a T.
3    Si el nodo n seleccionado pertenece al conjunto de metas
       T: Se marca el nodo como cerrado y se finaliza el
       algoritmo.
4    De lo contrario: Se marca n como cerrado y se evalúan con
       la función f los nodos adyacentes. Se marcan como
       abiertos todos los nodos sucesores que no estén
       cerrados, o cuyo valor de f sea ahora menor que lo
       era cuando se marco como cerrado previamente. Volver
       al paso 2.
```

2.2. Revisión de Antecedentes

Existen diversas estrategias para resolver el problema de *Path Planning* para equipos de robots móviles. Se pueden separar en dos grandes categorías: estrategias cooperativas y estrategias independientes, según si los robots del equipo se comunican entre sí para coordinar sus caminos de forma de evitar colisiones.

2.2.1. Estrategias Independientes

En algunos documentos las estrategias independientes son también categorizadas como estrategias de *Motion Coordination* [23]. En este tipo de estrategias, los robots generan su camino y avanzan por este sin ninguna comunicación con el resto de los robots. En el caso de detectarse un conflicto con otro robot, estos son tratados por el robot como un obstáculo más, y se lo evita. La forma de resolver este conflicto depende de la estrategia elegida.

Traffic Control

Este enfoque se basa en definir una serie de reglas de tráfico o control que los robots deben seguir mientras se mueven. Generalmente, los robots se mueven por caminos que planificaron de manera independiente, tomando solo en cuenta su propia meta. Luego, cuando el robot se encuentra en regiones con recursos compartidos (como el espacio en una intersección de caminos), los robots siguen las reglas de tráfico para coordinar sus movimientos con otros robots que requieran acceso a los mismos recursos.

Campos Potenciales

Este método de paradigma reactivo fue presentado inicialmente en 1985 por Khatib [16], se basa en la metáfora de aplicar un campo potencial al entorno en donde los objetos o puntos de interés generan fuerzas atractivas o repulsivas que condicionan el movimiento del robot.

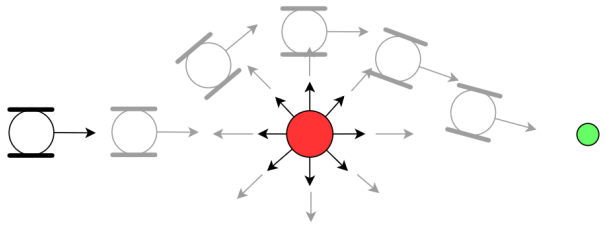


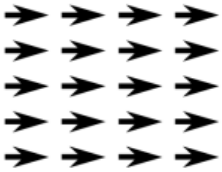
Figura 2.6: Robot evitando un obstáculo

En la Figura 2.6 se observa la manera en que actúa esta estrategia, donde el robot es atraído al destino y repelido por el obstáculo.
Se puede implementar de manera *offline*, cuando se conoce la información completa del entorno, o sensando el entorno en tiempo real. Los tipos de campos de fuerzas potenciales se ilustran en la Figura 2.7.

(a) Campo atractivo. Aplica una fuerza **atractiva**, la cual es utilizada para que los robots sean atraídos a un punto.

(b) Campo repulsivo. Aplica una fuerza **repulsiva**, la cual es utilizada para que los robots eviten obstáculos.

(c) Campo tangencial. Aplica una fuerza **tangencial** al obstáculo, utilizada para que los robots sean capaces de bordear obstáculos.



(d) Campo uniforme. Aplica una fuerza **uniforme**, utilizada para que los robots se mueven en determinado sentido.

(e) Campo perpendicular. Aplica una fuerza **perpendicular** al obstáculo, utilizada para que los robots sean capaces de evitar obstáculos o límites.

Figura 2.7: Tipos de fuerzas

Una desventaja de los algoritmos de campos potenciales es que el robot puede quedar atrapado en una región debido a la presencia de **mínimos locales**.
En la Figura 2.8 se dispone de un ejemplo donde el robot se sitúa en un lugar del espacio en el cual las meras fuerzas atractivas y repulsivas no permiten que el mismo alcance su destino. Esto ocurre debido a que la fuerza atractiva lo dirige hacia una región donde éste queda atrapado sin poder llegar al punto final.

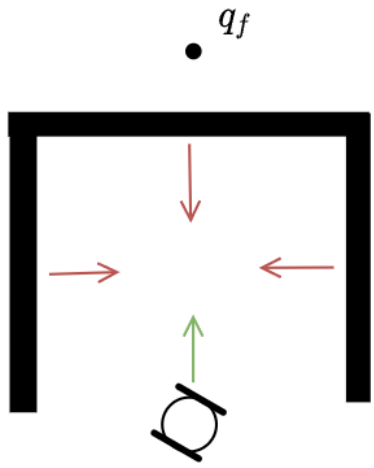
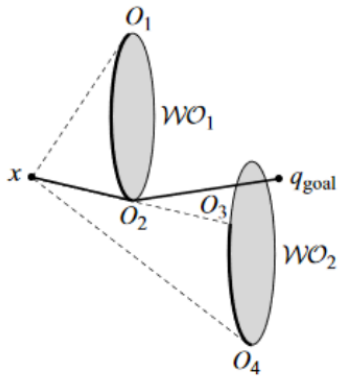


Figura 2.8: Robot atrapado en mínimo local, sin capacidad de alcanzar el punto q_f con fuerzas atractivas y repulsivas

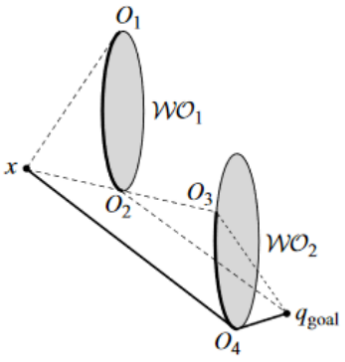
En [15] se describe una colección de técnicas efectivas para este enfoque que permiten escapar de mínimos locales.

Algoritmos Bug

Los algoritmos Bug son algoritmos basados en seguir una línea recta hacia el punto final reaccionando a los obstáculos cuando se topa con éstos. Son algoritmos simples, poco eficientes y están pensados para escenarios con obstáculos estáticos.
El algoritmo Bug tangencial consiste en que los robots sensen el entorno para decidir qué camino hacer en base a los obstáculos sensados y las distancias entre dichos puntos hasta la meta. Los robots alternan entre los estados de "movimiento hacia meta" y "rodear obstáculo".
En las Figuras 2.9(a) y 2.9(b), vemos dos ejemplos en el que el robot sensa y debe elegir qué dirección tomar en base a la información sensada. La posición del robot se representa con x, los bordes de los obstáculos sensados con O_i y el destino con q_{goal}. Los obstáculos están representados por óvalos grises.



(a) Se selecciona el camino hacia la meta $x - O_2 - q_{goal}$



(b) Se selecciona el camino hacia la meta $x - O_4 - q_{goal}$

Figura 2.9: Caminos calculados por un robot para llegar a la meta durante algoritmo Bug. Figura tomada de [13].

En la Figura 2.9(a) se elige el camino $x - O_2 - q_{goal}$ y en la Figura 2.9(b) el $x - O_4 - q_{goal}$. Sin embargo hay que tener en cuenta que se puede tomar caminos que no son óptimos, ya que por ejemplo el camino $O_2 - q_{goal}$ atraviesa un obstáculo.

En la Figura 2.10 vemos un ejemplo en donde un robot realiza un recorrido hacia la meta. Los círculos punteados representan el rango del lidar. Las líneas punteadas corresponden al camino realizado por el robot. Los obstáculos son representados por las figuras grises.

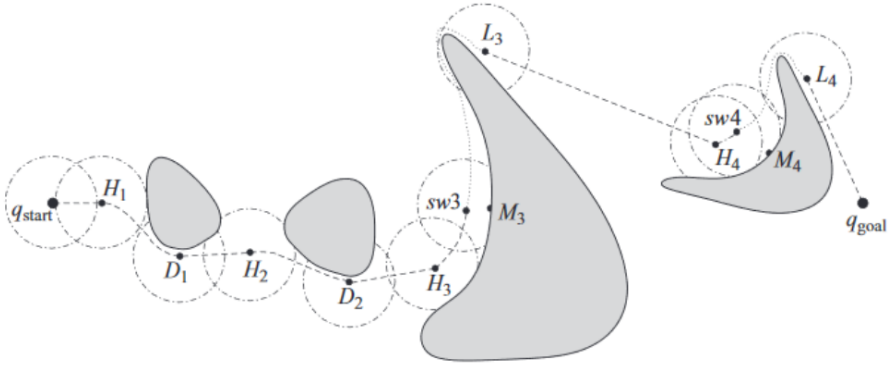


Figura 2.10: Ejemplo del recorrido de un robot desde q_{start} a q_{goal} utilizando la estrategia Bug tangencial. Figura tomada de [13].

En la Figura 2.10 el robot empieza en el estado “movimiento hacia meta”. En

los puntos H_i $(i = 1, 2, 3, 4)$ el robot detecta obstáculos e intenta ir hacia el destino por el camino más corto evitando chocar con los mismos. En los puntos D_i $(i = 1, 2)$ el robot no detecta obstáculos en el camino y puede desplazarse de manera directa al destino.
En el punto $sw3$ se detecta que el robot se está alejando del destino, siendo M_3 el punto más cercano al mismo, de esta manera pasa al estado "rodear obstáculo". Cuando el robot llega al punto L_3, el robot detecta que puede llegar a un punto más cercano a la meta que M_3, por lo tanto vuelve al estado "movimiento hacia meta". Este comportamiento se repite para $sw4$.

Comportamientos

Otras estrategias reactivas se basan en comportamientos basados en reglas. Estas estrategias consisten en la definición de reglas o acciones para los distintos estímulos a los que es sometido el robot.
Toda la información se obtiene de la entrada de los sensores del robot, y el robot utiliza esa información para ir cambiando gradualmente sus acciones y corrigiéndolas de acuerdo con los cambios del entorno. Los robots son programados con un conjunto básico de características iniciales, se les da un repertorio de comportamientos para usar y se les especifican las condiciones bajo las que se activa cada uno de estos. Simplemente reaccionan a su entorno de cierta manera para resolver determinado problema. Algunos pueden basarse en el conocimiento interno aprendido de sus experiencias pasadas combinadas con sus comportamientos básicos para resolver problemas.

La arquitectura Subsumption (Figura 2.11) divide las tareas en varias capas donde conductas simples van formando incrementalmente conductas más complejas y pueden coexistir con estas.

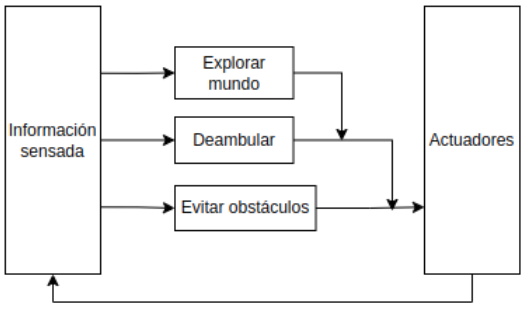


Figura 2.11: Arquitectura Subsumption

Otras estrategias

En [8], se presenta otra estrategia independiente donde se calcula la probabilidad de colisión y si se determina que la probabilidad es alta, los robots relacionados

entran en un estado de evasión de colisión, donde a partir de prioridades y posiciones, cada robot decide los ángulos y velocidades necesarias para evitar la misma. En la Figura 2.12 si los robots R1 y R2 avanzan en dirección de las flechas punteadas, los robots no colisionarían. En cambio si siguen la trayectoria de las flechas no punteadas colisionarían en el punto P_{12}.

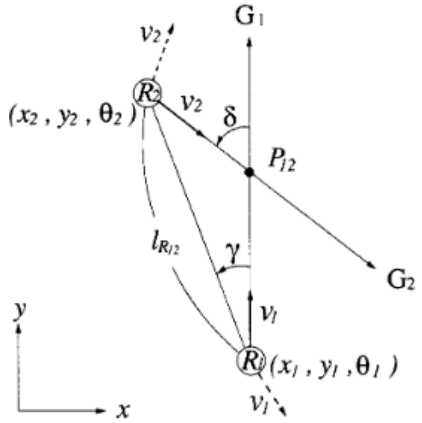


Figura 2.12: Configuración de los robots R1 y R2, y proyección de punto de choque P_{12}. Tomado de [8]

.

Se menciona que en escenarios donde los robots están aglomerados, se pueden observar colisiones y se presentan alternativas pero no son analizadas en el artículo.

2.2.2. Estrategias Cooperativas

En las estrategias cooperativas los robots se comunican entre sí y los movimientos de todos los robots se planifican de forma de evitar colisiones.

Acopladas

En las estrategias acopladas, los robots son modelados como un solo robot para el que se buscan los caminos. De esta manera, el espacio de configuración en el que se busca el camino, es la combinación de los espacios de configuración de todos los robots del equipo, y las configuraciones en las que se darían las colisiones se considerarán como obstáculos, por lo que los caminos que se encuentren no generarán colisiones entre los robots.

Las ventajas de los algoritmos acoplados es que suelen ser completos para determinados tipos de problemas y al tomar en cuenta a todos los robots a la vez

pueden llegar a soluciones mejores (considerando los resultados para todos los robots como conjunto) que aquellas encontradas por otros tipos de algoritmos que trabajan con menos información. Sin embargo, en la práctica pueden no resultar útiles ya que suelen tener problemas relacionados al tiempo de computación del problema, y el espacio necesario para la representación del espacio de búsqueda.
Como se mencionó en la Sección 2.1, la dimensión del espacio de un robot se relaciona con los grados de libertad de movimiento del robot: si un robot tiene n grados de libertad, entonces el espacio de configuración tiene n dimensiones, y buscar un camino entre dos configuraciones requiere encontrar un camino en un espacio n-dimensional. Si en lugar de un único robot tenemos k robots, entonces se debe buscar un camino en un espacio nk-dimensional para encontrar una solución que desplace a los k robots desde sus posiciones iniciales a sus posiciones finales. La dimensionalidad del espacio en que se debe buscar el camino es directamente proporcional a la cantidad de robots con que se trabaja, llevando a los problemas previamente mencionados.

A continuación mencionamos algunos enfoques acoplados:

Robot compuesto

Consiste en utilizar métodos clásicos de *Path Planning* para robots individuales para hallar el camino para el robot compuesto en su espacio de configuración.

La ventaja de estos algoritmos es que pueden garantizar completitud, sin embargo, debido a la alta dimensión del espacio de configuración del robot compuesto, suelen ser demasiado costosos computacional-mente si se buscan los caminos en todo el espacio de búsqueda. Una forma de utilizarlos es limitar los movimientos de los robots, por ejemplo utilizando *roadmaps*.

En [24] se presenta una estrategia de esta clase, en la que se incorporan todos los obstáculos y robots móviles a un único espacio de configuración unificado. El algoritmo primero genera una descomposición en células del espacio libre y luego busca un camino en el grafo de adyacencia asociado.

Super-graphs

Este enfoque, presentado en [30], se basa en la creación de un *roadmap* compuesto que representa una red de movimientos posibles para el robot compuesto. Primero se construye un *roadmap* para cada robot del equipo, y luego estos se combinan para formar el *roadmap* compuesto. Se proponen dos formas para los grafos compuestos: **flat super-graphs y multi-level super-graphs.**
En los primeros, cada nodo representa un posicionamiento válido de los n robots, y una arista representa el movimiento de un único robot por un camino no bloqueado. La principal desventaja de esta representación es que el tamaño de la *flat super-graph* crece exponencialmente según el número de robots.

Por otro lado, en el *multi-level super-graph*, se reduce el tamaño del super-graph al combinar múltiples nodos en un único nodo del grafo, permitiendo que múltiples robots se muevan al mismo tiempo.

En la Figura 2.13 se muestra un ejemplo de secuencia generada por esta estrategia. En este caso tenemos 5 robots ubicados en línea en un pasillo y se quiere que terminen alineados pero con el orden al revés. En cada imagen se representa con flechas los movimientos que deben realizar los robots en cada paso. Como se puede ver, primero todos los robots dejan libre el pasillo, luego cruzan el mismo para quedar del lado correcto, y finalmente van en orden hacia sus posiciones finales. Cabe destacar que cuando los caminos no se cruzan, se permite que varios robots se muevan simultáneamente.

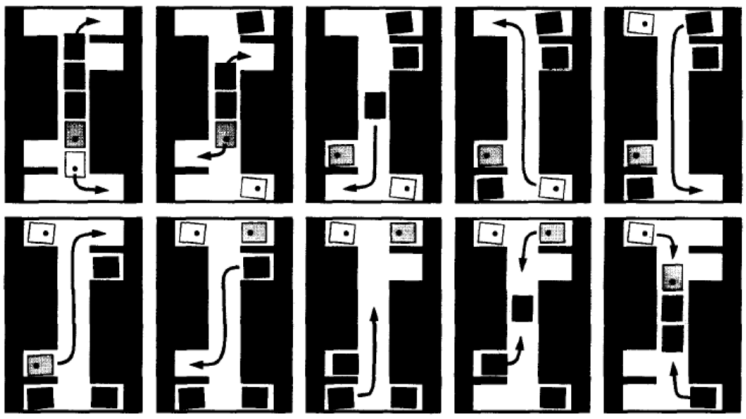
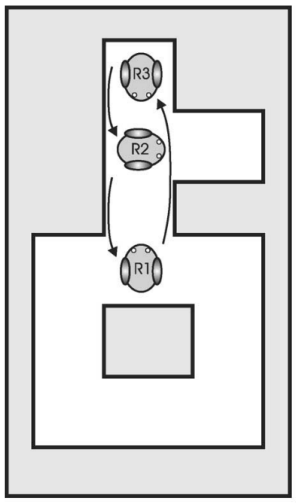


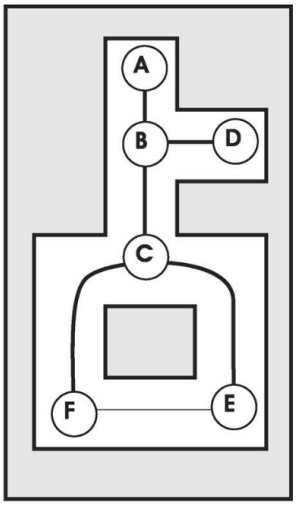
Figura 2.13: Ejemplo de secuencia de movimientos generada por la estrategia de Super-graphs. Figura tomada de [30].

Árbol de cubrimiento

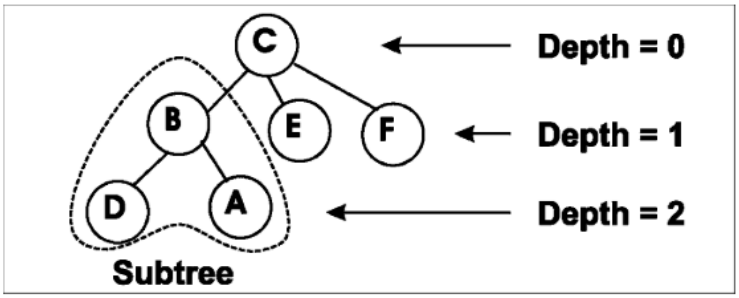
El método de Árbol de cubrimiento (*Spanning Tree*) presentado en [25] es un planificador para *roadmaps* que utiliza una representación de árbol de cubrimiento para crear y mantener caminos libres en el ambiente. Los nodos del grafo corresponden a las posiciones iniciales y las metas de los robots, y los ejes la conectividad entre estos nodos. Se crea una representación de árbol de cubrimiento de este grafo, que es un subconjunto del grafo original, con todos los nodos sin ciclos. La raíz del mismo es el nodo que esté más cerca del centro geográfico del mapa. En la Figura 2.14 se presenta un ejemplo de esta representación.



(a) Posiciones iniciales y finales de los 3 robots



(b) Representación en grafo del mapa



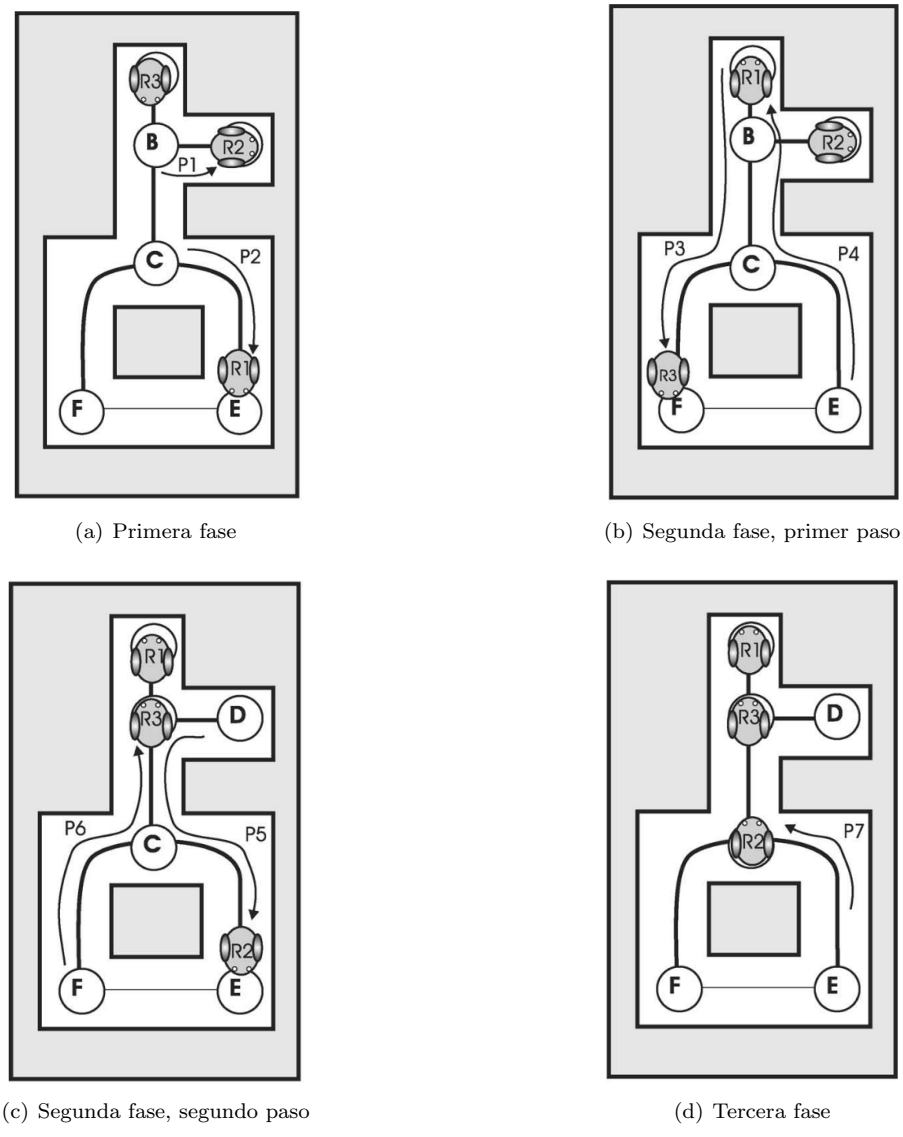
(c) Representación de árbol de cubrimiento

Figura 2.14: Ejemplo de representación de Árbol de cubrimiento para un problema con 3 robots. Figuras tomadas de [25].

En este ejemplo tenemos 3 robots. Queremos que el robot R1 llegue hasta la posición en que inicia el robot R3, que el robot R2 se mueva hasta la posición inicial del robot R1, y por último que el robot R2 termine en la posición en que inicia el robot R1.

En la primera fase, se genera un plan que mueve a todos los robots a las hojas del árbol de cubrimiento. En la segunda fase, los robots se mueven hacia posiciones desde las que pueden alcanzar sus metas sin obstruir los caminos de otros robots, esto es posible moviendo a los robots según la profundidad de sus metas en el árbol de cubrimiento. La tercera fase mueve los robots a las metas que no se hayan alcanzado todavía. Estas

fases son una secuencias de movimientos en los que se mueven de a un robot a la vez. La fase final del proceso busca mejorar la calidad del plan permitiéndole a los robots que se muevan simultáneamente cuando esto no genera colisiones. En la Figura 2.15 se muestra este proceso para el ejemplo definido en la Figura 2.14.



(a) Primera fase

(b) Segunda fase, primer paso

(c) Segunda fase, segundo paso

(d) Tercera fase

Figura 2.15: Ejemplo de solución generada por la estrategia de Árbol de cubrimiento. Figuras tomadas de [25]

Los autores también proponen utilizar este método en conjunto con un planificador desacoplado para encontrar caminos más cortos. Según sus resultados, el planificador desacoplado suele conseguir mejores resultados para equipos de 20 robots o menos, pero para equipos más grandes, el método acoplado resulta más eficiente.

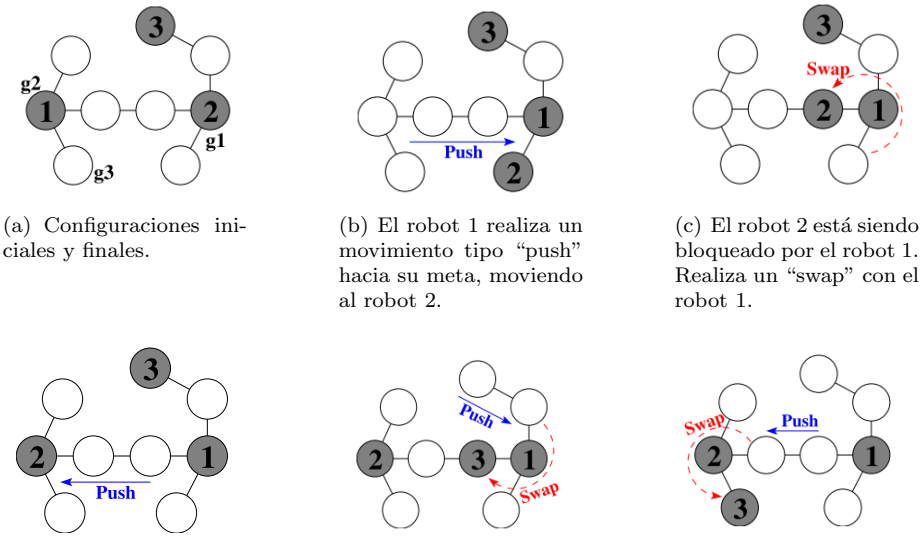
Path planning en un grafo

Dada una representación de grafo donde cada nodo es una configuración determinada para un robot y las aristas son caminos conocidos entre configuraciones, el problema de *Path Planning* para múltiples robots consiste en encontrar una serie de movimientos de forma que todos los robots lleguen desde el nodo que representa su posición inicial hasta el que representa su posición final por una secuencia de aristas sin pasar por un nodo ocupado por otro robot.

En [19], se presenta una estrategia acoplada para *Path Planning* en un grafo. La estrategia es computacionalmente eficiente y completa para gran cantidad de problemas (todas las instancias en las que haya $N-2$ robots o menos en un grafo con N vértices). La estrategia consiste en, secuencialmente, llevar a los robots desde sus posiciones iniciales a sus respectivas metas mediante el uso de dos primitivas: "push" y "swap".

La primera consiste en avanzar hacia la meta, "empujando" los robots con los que se encuentre que no estén en sus posiciones finales hacia afuera del camino, en caso de ser posible, o hacia adelante de lo contrario. La primitiva "swap" se utiliza cuando en el siguiente nodo del camino hay un robot que no puede ser empujado, ya sea porque se encuentra en su posición final, o porque no hay hacia donde empujarlo, y consiste en que cambien de posición, terminando el robot del otro lado del robot que tenía en el camino. Esta operación puede requerir que otros robots se muevan para dejar lugar para maniobrar, pero todos deben volver a las posiciones en las que se encontraban.
En el artículo se demuestra que si no es posible realizar un "swap" para dos robots, entonces no existe una solución al problema.

En la Figura 2.16 presentamos un ejemplo de esta estrategia para 3 robots. El nodo en que se encuentra cada robot en cada momento se representa con el numero del robot. En la Figura 2.16(a) se muestran además las metas a las que quieren llegar cada uno de los robots, por ejemplo, el robot 1 debe llegar al nodo del grafo marcado con "g1", y así para los demás robots. Las primitivas "push" y "swap" se representan con flechas azules y rojas respectivamente. Para los movimientos de tipo "push", la flecha indica la posición inicial y final del movimiento del robot. Para los movimientos de tipo "swap" la flecha indica la posición inicial y final del robot que intenta avanzar, y rodea al robot con el que debe intercambiar posiciones.



(a) Configuraciones iniciales y finales.

(b) El robot 1 realiza un movimiento tipo "push" hacia su meta, moviendo al robot 2.

(c) El robot 2 está siendo bloqueado por el robot 1. Realiza un "swap" con el robot 1.

(d) El robot 2 realiza un "push" hasta su meta.

(e) El robot 3 realiza un "push" y es bloqueado por el robot 1. Realiza un "swap" con el robot 1.

(f) El robot 3 realiza un "push" y es bloqueado por el robot 2. Realiza un "swap" con el robot 2 y llega a su meta.

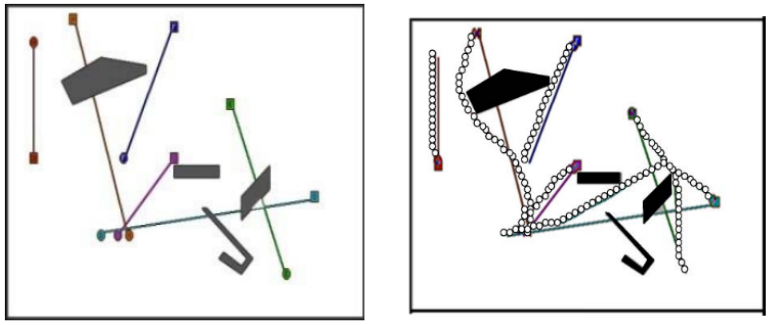
Figura 2.16: Ejemplo de solución para 3 robots utilizando las primitivas "push" y "swap". Figuras tomadas de [19].

Artificial Bee Colony Optimization

En [4], se propone un método que hace uso del algoritmo de optimización "Artificial Bee Colony (ABC)". Dada una función objetivo que depende de la distancia a recorrer hasta la meta y la distancia a obstáculos (incluyendo otros robots), y una velocidad constante a la que se mueve cada robot, en cada paso se busca elegir los ángulos en que se moverán los robots para minimizar el valor de la función objetivo.

El algoritmo ABC se basa en crear una población de soluciones aleatorias, evaluarlas, y luego buscar opciones mejores, primero buscando soluciones similares de las ya estudiadas, y luego de varios pasos de no encontrar mejoras, probando nuevas soluciones completamente aleatorias.

En la Figura 2.17 se muestra un ejemplo de los caminos generados por este algoritmo para 4 robots.



(a) Posiciones iniciales y finales de cada robot

(b) Caminos generados. Los puntos que forman los caminos representan la posición de los robots en cada paso del algoritmo.

Figura 2.17: Ejemplo de la aplicación de ABC para encontrar caminos libres de colisión para 4 robots. Figuras tomadas de [4].

Desacoplado

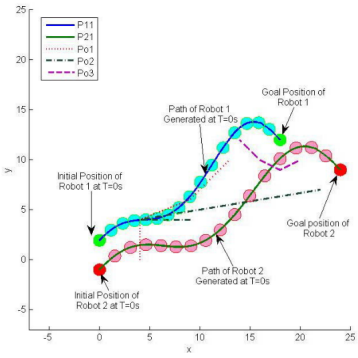
Las estrategias desacopladas separan el problema de path planning cooperativo en distintos componentes independientes. Estas estrategias no pueden garantizar completitud y pierden calidad de la solución, pero logran ser más eficientes que las acopladas. Un enfoque común es tener un primer paso en que se generan los caminos para cada robot, y luego un segundo paso en que se resuelven los posibles conflictos entre estos.

Prioritized Planning

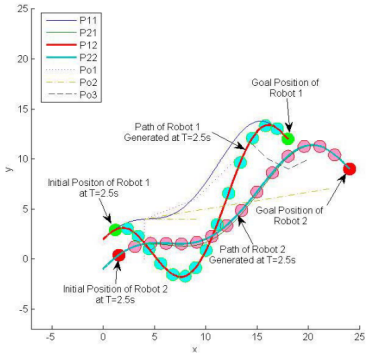
En esta clase de estrategias se les asignan prioridades a los robots según algún criterio determinado, y los caminos son computados en secuencia, de forma que los robots ignoren a los que tengan menor prioridad que ellos, pero tomen en cuenta las trayectorias planeadas por aquellos con mayor prioridad a la hora de generar sus caminos para evitar colisionar con ellos. La ventaja de este enfoque es que se reduce de un problema de alta dimensión, a una secuencia de problemas de dimensión más baja.

En [34], se presenta una estrategia de *Prioritized Planning* para equipos de robots en ambientes dinámicos, que toma en cuenta las limitaciones cinemáticas de los robots. Primero, cada robot aplica un método analítico para encontrar un camino. Luego, se utiliza un sistema de prioridades para realizar la cooperación. En cada intervalo de tiempo, los robots calculan sus valores de prioridad según la distancia recorrida y la distancia a la meta, se comunican con los otros robots para decidir el ranking, y realizan su *Path Planning*, ignorando los robots con menor prioridad que ellos, y tratando a los de mayor prioridad como obstáculos dinámicos. En la Figura 2.18 se muestra como cambian las trayectorias de 2 robots utilizando esta

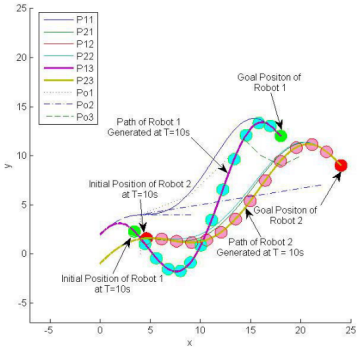
estrategia en un ambiente con 3 obstáculos dinámicos.



(a) T=0s. P11 y P21 son los caminos generados por los robots 1 y 2 respectivamente en T=0s. Po1, Po2 y Po3 son las trayectorias de los obstáculos dinámicos.



(b) T=2.5s. P12 y P22 son los caminos generados por los robots 1 y 2 respectivamente en T=2.5s. Al cambiar los estados de los obstáculos, el robot 1 replanifica su camino para evitar al obstáculo 1.



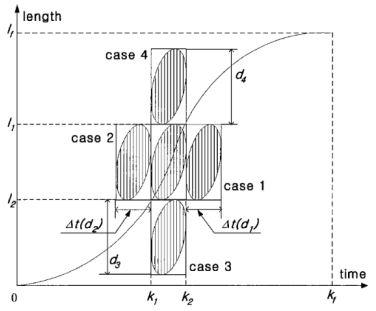
(c) T=10s. P13 y P23 son los caminos generados por los robots 1 y 2 respectivamente en T=10s. Se detecta un conflicto entre los robots. De acuerdo a la ecuación utilizada para el cálculo de prioridad, el robot 1 tiene más prioridad que el robot 2, por lo que lo ignora y continua con el mismo camino. El robot 2 replanifica su camino para evitar al robot 1.

Figura 2.18: Ejemplo de evolución de las trayectorias al avanzar el tiempo en un caso con 2 robots y 3 obstáculos dinámicos. Ejemplo tomado de [34].

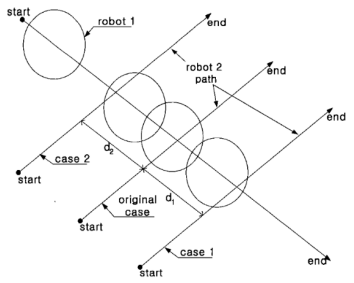
Analytical Method on Collision Maps

En [22], se presenta un método para evitar colisiones utilizando el mapa de colisiones. Se utiliza un método analítico basado en desplazar la región de

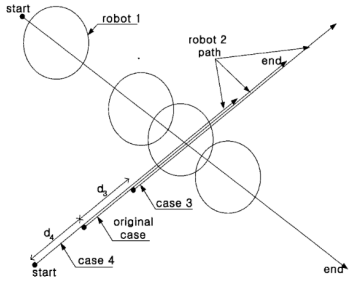
colisión en el mapa de colisiones. El método toma en cuenta la velocidad a la que se mueven los robots para encontrar los mínimos desplazamientos del camino que permitan evitar la colisión. La estrategia permite tener prioridades entre los robots, ya que el de mayor prioridad mantiene su camino original y el otro es el que desplaza su camino para evitar la colisión. En la Figura 2.19 se presenta un ejemplo de los desplazamientos posibles para evitar la colisión en un caso con dos robots.



(a) Mapa de colisiones para el robot 2. El eje vertical representa el largo del camino recorrido, mientras que el horizontal representa el tiempo. En el centro se ve la región de colisión original, y a los costados los posibles desplazamientos mínimos.



(b) Desplazamientos de la trayectoria del robot 2 para los casos 1 y 2



(c) Desplazamientos de la trayectoria del robot 2 para los casos 3 y 4

Figura 2.19: Ejemplo de evitación de colisión desplazando la región de colisión en un caso con 2 robots.

RRT Path Planning

En [6], se presenta un algoritmo desacoplado. Se basa en que cada robot calcule iterativamente el camino a su destino utilizando RRT [18] (*Rapidly-Exploring Random Trees*), obteniendo la potencial mejora del camino más corto (PPI) y en base a esto se va seleccionando el robot con mejor PPI

para actualizar.

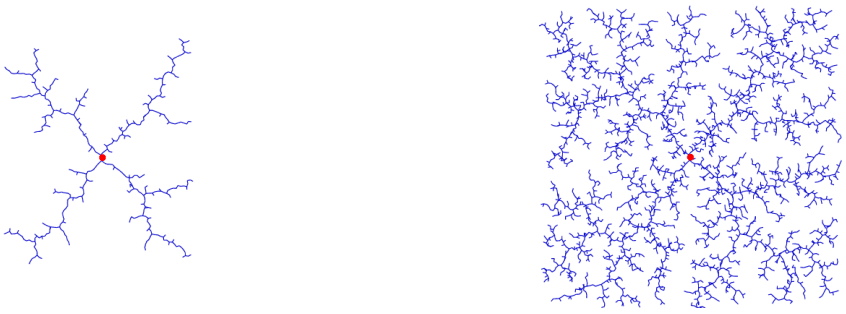


Figura 2.20: Generación de Roadmap a partir del algoritmo RRT

Cuando un robot es seleccionado actualiza su camino, y en base a ciertos criterios efectúa cambios en el resto de los robots de manera de mejorar los costes globales. Luego esto se comunica al resto de los robots.

Path Coordination

En las estrategias pertenecientes a esta categoría, el problema se descompone en *Path Planning* y *Velocity Planning*. Primero se generan independientemente los caminos para cada robot, y luego se planifica la velocidad con la que recorrerán esos caminos de forma de evitar colisiones. La incorporación del planeamiento de velocidades implica que se agrega una dimensión de tiempo a los cálculos.

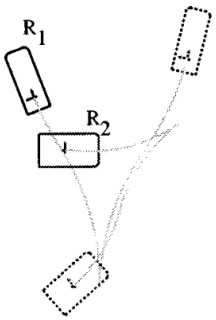
En [29], se presenta una estrategia para *Velocity Planning* que se basa en utilizar *Path Finding* sobre un **diagrama de coordinación**. Esta estrategia se le aplica a robots que se mueven por caminos tipo SA (formados por segmentos rectos y arcos de círculo).

Los diagramas de coordinación son diagramas que representan las posiciones de los robots a lo largo de los caminos en las que pueden ocurrir conflictos. Cada eje se asocia al camino que recorre uno de los robots. Es decir, avanzar respecto a un eje es equivalente a que el robot avance en su camino. En el diagrama se representan como áreas de conflicto las secciones de las trayectorias en que podrían ocurrir colisiones.

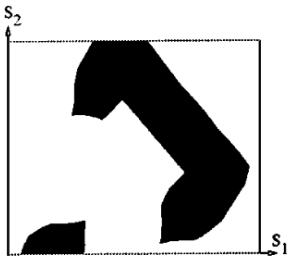
La coordinación luego puede verse como un problema de *Path Finding* sobre el diagrama de coordinación, donde se busca llegar desde el punto (0,0), que representa que ambos robots se encuentran en su posición inicial, al punto (1,1) que representa que ambos robots se encuentran en su posición final, evitando obstáculos (áreas de conflicto).

Para facilitar los cálculos se simplifica la representación de los obstáculos en el diagrama de coordinación utilizando una representación de bounding box (mediante figuras rectangulares). El procedimiento consiste en

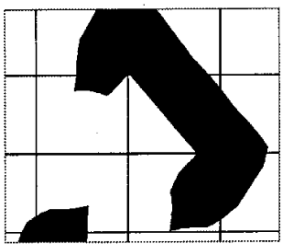
descomponer los caminos en sus segmentos de recta y arcos que los forman, luego se divide el espacio utilizando una grilla y dentro de cada celda se aproxima el obstáculo contenido utilizando una figura rectangular. En la Figura 2.21 presentamos un ejemplo de diagrama de coordinación.



(a) Caminos de dos robots.



(b) Diagrama de coordinación.



(c) Partición resultante de la descomposición de los caminos.

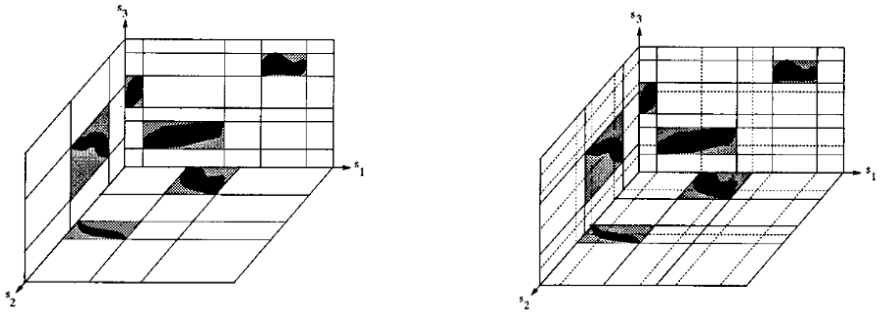
(d) Representación de bounding box de las áreas de conflicto y un camino posible.

Figura 2.21: Ejemplo de diagrama de coordinación tomado de [29].

Esta representación da lugar a una descomposición en células (cell decomposition) del diagrama de coordinación. Luego, se puede utilizar un algoritmo de búsqueda, por ejemplo A^*[11], para hallar un camino libre de colisiones en el diagrama de coordinación. En la Figura 2.21(d) se muestra un camino libre de colisión que representa una posible solución al problema de coordinación para el ejemplo.

Generalizando, para n robots, el diagrama de coordinación generalizado es un prisma n-dimensional con un eje por cada uno de los robots. En los planos definidos por cada par de ejes, se representa el diagrama de coordinación para el par de robots asociados. El prisma se subdivide en celdas respecto a la descomposición en celdas de los planos, para esto, la descomposición en células de cada plano se utiliza para refinar las descomposiciones de los demás planos. En la Figura 2.22 se muestra un ejemplo de

como se refina la descomposición en células combinando la descomposición de todos los planos.



(a) Descomposición original de cada plano

(b) Descomposición refinada

Figura 2.22: Ejemplo de refinación de la descomposición en células. Tomado de [29].

Estas celdas se consideran ocupadas si al menos un par de robots tiene un conflicto. Los obstáculos en el espacio n-dimensional se definen a partir del producto cartesiano de una región de conflicto en una de las caras con el resto de los ejes, por lo tanto, se puede afirmar que la topología del diagrama de coordinación generalizado está dada por la topología de los diagramas de coordinación de cada par de ejes. En la Figura 2.23 se muestra como los obstáculos se proyectan desde cada plano al resto del diagrama n-dimensional.

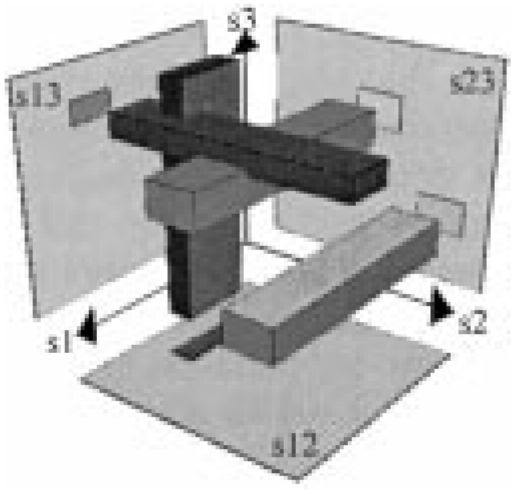


Figura 2.23: Estructura de los obstáculos en el diagrama de coordinación.

Análogamente al caso de dos robots, el problema de coordinación se puede plantear como la búsqueda de un camino libre de colisiones desde el punto inicial (0, 0, ..., 0), en que cada robot se encuentra en su posición inicial hasta el punto final (1, 1, ..., 1) en que todos los robots terminaron sus recorridos.

El algoritmo es completo para una gran cantidad de problemas. En el artículo se muestra que resuelve eficientemente problemas con grandes flotas de robots (10 robots en situaciones de peor caso y más de 100 en situaciones prácticas).

Un problema de las estrategias desacopladas es que pueden llevar a *deadlocks* aún cuando existen soluciones posibles. En estos casos se puede implementar una solución acoplada-desacoplada, que utilice un método acoplado sobre una pequeña parte del problema.

Acoplado-Desacoplado
Las estrategias de esta categoría buscan potenciarse obteniendo las mejores características de las estrategias acopladas y desacopladas. Es decir que por medio de algoritmos que combinen ambas estrategias, buscan lograr soluciones óptimas y completas en tiempos de ejecución razonables y viables.

Subdimensional expansion

En [32] se presenta una estrategia en la que se planea cada robot individual por separado, es decir con un enfoque desacoplado y definiendo un espacio de búsqueda de una dimensión para cada robot. Solo se centralizan los conjuntos de robots luego de que se encuentra una colisión entre ellos,

aumentando la dimensión del espacio de búsqueda localmente para asegurarse de encontrar un camino alternativo. De esta manera minimiza la dimensión del espacio de búsqueda siempre que sea posible, agrandándolo solo cuando sea necesario.
Si bien en el peor caso el espacio de búsqueda es el espacio de configuración completo, para la mayoría de los problemas se pueden resolver con espacios de búsqueda de baja dimensión.

En la Figura 2.24 tenemos un ejemplo visual espacio de búsqueda. El cubo representa la región del espacio de búsqueda en qué el conjunto de colisiones contiene 3 robots. El cuadrado representa la región del espacio de búsqueda en qué el conjunto de colisiones contiene 2 robots. El círculo es la representación de los destinos. Para ambos espacios de búsqueda se tuvo que centralizar los conjuntos de robots y generar un enfoque acoplado.

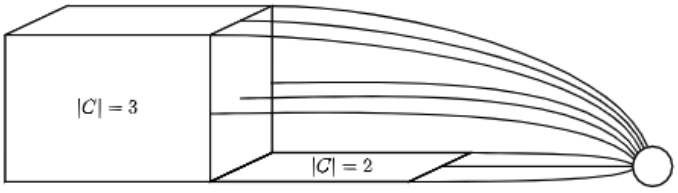


Figura 2.24: Visualización geométrica del espacio de búsqueda embebido en el espacio de configuración. Tomado de [32]

Aunque crezca el espacio de búsqueda a cubrir todo el espacio de configuración en el peor escenario, en la mayoría de las ocasiones se puede solucionar con un espacio de búsqueda de dimensión baja que permite una eficiente computación y un camino de calidad.

Este enfoque utiliza el algoritmo M^*, basado en el algoritmo A^*. Es completo y óptimo.

2.3. Ambiente de desarrollo

2.3.1. ROS

ROS (Robot Operating System)[28] es un meta sistema operativo utilizado en aplicaciones de robótica. Provee servicios útiles incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comúnmente usadas, envío de mensajes entre procesos y mantenimiento de paquetes. También provee herramientas y librerías para obtener, construir, escribir y ejecutar código en distintas máquinas.

Conceptos Básicos

Nodo: Es un proceso que realiza cierta funcionalidad. El software del robot puede descomponerse en nodos. Cada uno de estos cumple un cometido diferente. Esto tiene la ventaja de modularizar los componentes, pero también genera el desafío de sincronizar a los distintos nodos.

Nodo Master: El nodo *roscore* provee registro de nodos y búsqueda de los mismos. Este nodo siempre tiene que estar activo para que el resto de los nodos puedan interactuar entre sí.

Tópico: Los tópicos son una herramienta para la transmisión de mensajes. Las operaciones que se realizan son:

- Publish (para enviar un mensaje)
- Subscribe (para recibir un mensaje)

Los tópicos proveen comunicación asincrónica no bloqueante Many to Many. Se identifican mediante su nombre, y pueden ser accedidos desde cualquier nodo. Los mensajes que se envian a un tópico deben tener un tipo esperado (Int16, Float16, Twist, etc.)

Servicios: Es la forma de comunicación entre nodos cuando se precisa obtener una respuesta. Son sincrónicos bloqueantes.

Parámetros: Son parámetros mantenidos por el nodo master, que se pueden modificar y obtener por cualquier nodo.

2.3.2. Simulador

A continuación se detallan las características mínimas requeridas por el simulador para el correcto desarrollo del proyecto de grado.

Permitir robots móviles.

Permitir reproducir un experimento N veces consecutivas a partir de una situación inicial.

Permitir acelerar el tiempo.

Facilitad de integración con Python y C/C++.

Facilidad de integración con ROS.

(Opcional) Consumir lo menos posible (CPU, memoria, disco)

Opciones posibles
Webots [33]

- Utilizado por marcas reconocidas mundialmente (Huawei, Nasa, Honda, etc).
- Fast Protoyping
- Contiene una gran variedad de simulaciones de diferentes robots para ambientes tanto exteriores como interiores.
- Permite integración con Python, C, C++, ROS.
- Posee una extensa documentación con ejemplos y guías para mejorar la curva de aprendizaje.
- Existen muchas formas de aumentar/disminuir la velocidad de simulación y reproducir un experimento N veces.
- Permite detectar colisiones.

Gazebo [9]

- Permite simular una amplia variedad de robots en ambientes exteriores e interiores.
- Permite aumentar/disminuir las velocidad de las simulación, así como reproducir un experimento N veces.
- Permite integración con Python, C, C++, ROS.
- No tan performante en comparación con otras opciones.
- Permite detectar colisiones.

CoppeliaSim (ex V-REP) [5]

- También cumple con los requisitos mínimos.
- Publicado en 2019.
- Permite simular una amplia variedad de robots en ambientes exteriores e interiores

Rendimiento

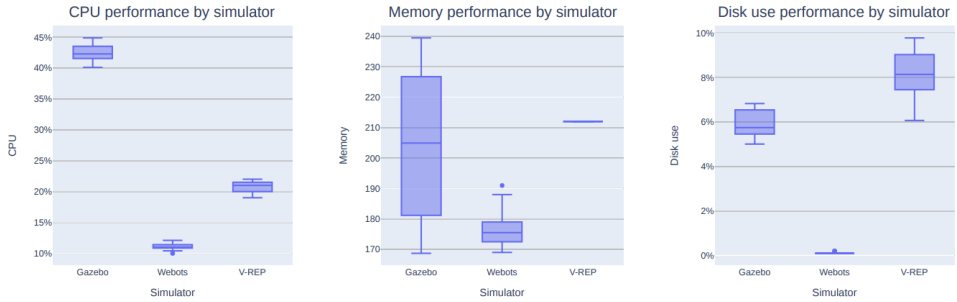


Figura 2.25: Comparación en CPU, memoria y uso de disco entre los distintos simuladores. Tomado de [3]

Según el estudio **A Comparison of Humanoid Robot Simulators [3]**, podemos concluir que Webots es la mejor opción como simulador para realizar el proyecto.

El estudio compara Webots, Gazebo y V-REP (ahora CoppeliaSim) en uso de CPU, uso de memoria y uso de disco, ejecutando 20 veces el escenario de llegar a una posición específica esquivando un obstáculo.
Todos los simuladores cumplen con los requisitos mínimos necesarios mencionados para la correcta realización del proyecto. Sin embargo, Webots destaca en términos de rendimiento de CPU, memoria y disco, por lo que optamos designarlo como el simulador a utilizar para el desarrollo del proyecto. Más específicamente se utilizó la versión Webots 2021a.

2.3.3. Máquina Virtual

Se decidió utilizar una máquina virtual para la realización del proyecto. De esta forma se permite la portabilidad del proyecto, se asegura que todos tengan el mismo ambiente y así se minimizan posibles errores de configuración.

Se eligió VMWare[31] como Software de Virtualización por las siguientes razones:

- Fácil de Usar
- Soporte para Windows y Linux
- Gratuito para uso no comercial
- Soporte de OpenGL 3.1

En particular el último punto es vital para el correcto funcionamiento del simulador en la máquina virtual, ya que sin una versión 3.1 o superior, funciona muy lento. Es por este punto que se descartó el Software de Virtualización VirtualBox.

2.3.4. Descripción del robot

Dentro del hardware disponible para Webots, decidimos utilizar como robot el Khepera IV [17] (Figura 2.26). Se optó por utilizar este robot debido a que es popular, moderno y tiene dos ruedas centradas, facilitando movimientos como girar sobre sí mismo.
El robot Khepera IV es un robot diferencial, ya que el movimiento esta basado en una tracción diferencial, lo que implica que utiliza dos ruedas controladas individualmente.

Figura 2.26: Robot Khepera IV

2.3.5. Cinemática de robot

Surge el problema de hacer que el robot se mueva a un punto objetivo (Figura 2.27). Para esto se calcula el ángulo θ entre el *heading* del robot y el punto deseado. En base a θ se puede calcular la velocidad angular w. Sabiendo también la velocidad lineal requerida se puede aplicar las ecuaciones para un robot diferencial (2.4) y obtener la velocidad de cada rueda.

$$v_r = \frac{2v - wL}{2R} v_l = \frac{2v + wL}{2R} \tag{2.4}$$

Siendo:

- v: velocidad lineal.
- w: velocidad angular.
- L: distancia entre ruedas.

- R: Radio de ruedas.

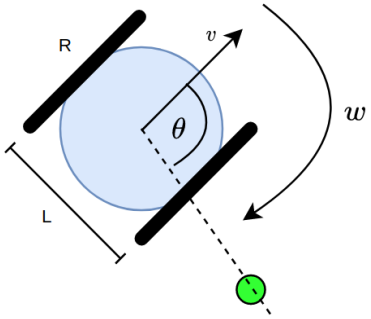


Figura 2.27: Ángulo a destino

Para resolver la cinemática del robot, se define la velocidad lineal v conocida, por lo que para obtener las velocidades de las ruedas solamente es necesario calcular w. Para calcular w se optó por utilizar un control PID (Sección 2.3.6). El timestep se definió como 16ms, ya que observamos que proveía una buena capacidad de respuesta, sin degradar mucho el rendimiento en comparación a 8ms.

2.3.6. Control PID

El control PID [12] es un mecanismo de control muy utilizado en muchas áreas que permite regular un parámetro deseado a través de un bucle de retroalimentación. En la Figura 2.28 se ilustra el flujo del control.

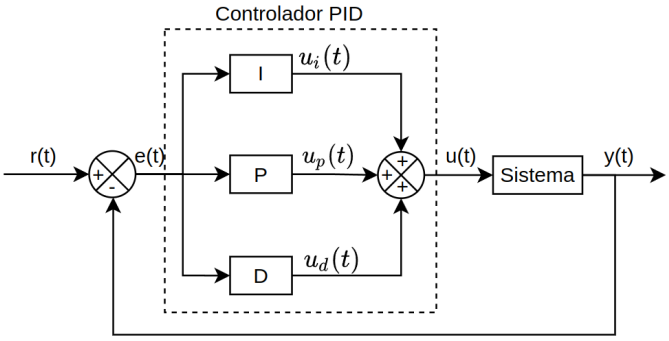


Figura 2.28: Controlador PID

El control consiste en medir la diferencia entre la variable real obtenida y la variable deseada y aplicar una serie de operaciones al error actual para obtener

un nuevo valor.

Los componentes potencial, integral y derivativo se definen en (2.5), (2.6), (2.7) respectivamente.

$$u_p(t) = K_p e(t) \tag{2.5}$$

$$u_i(t) = K_i \int_0^t e(\tau) d\tau \tag{2.6}$$

$$u_d(t) = K_d \frac{de}{dt} \tag{2.7}$$

El componente proporcional (2.5) depende del error actual, puede generar sobre-oscilaciones y no puede resolver sistemas donde existe un error estacionario (ya que error 0 implica salida 0). El control integral (2.6) depende de los errores pasados, soluciona el problema del error estacionario y suele acelerar la convergencia. El componente derivativo (2.7) sirve para ajustar el control a posibles errores futuros. Estos valores son sumados para ajustar el proceso (2.8), pero no garantizan un control óptimo del sistema.

$$y(t) = u_p(t) + u_i(t) + u_d(t) \tag{2.8}$$

Para la cinemática de un robot terrestre diferencial, descartamos el control derivativo, ya que los controles proporcional e integral fueron suficientes para resolver el problema. La simulación realizada se aplica bajo un tiempo discreto con determinado *timestep* Δt, por lo que aplicar el control integral requiere calcular las áreas de los errores de manera discreta. Esto resulta en (2.9).

$$y(t) = K_p e(t) + K_i \sum_{i=1}^{t} e(i) \Delta t \tag{2.9}$$

Para ajustar los parámetros K_p, K_i, se utilizó un procedimiento experimental [26] que consiste en ir duplicando un valor pequeño de K_p sin utilizar el control integral hasta obtener oscilaciones, en dicha situación se divide el valor y se repite el procedimiento con K_i.

Capítulo 3

Implementación de las estrategias

Contenido

3.1. Restricciones y asunciones

En base a las recomendaciones del tutor, se utilizó ROS y el control PID como restricciones del proyecto y se definió implementar una estrategia independiente reactiva y una estrategia cooperativa jerárquica para comparar los desempeños. El tercer algoritmo implementado no fue parte de la definición inicial del proyecto, y se realizó para combinar las estrategias utilizando las ventajas de cada una de ellas.

3.2. Estrategias

Se implementaron 3 estrategias, una con enfoque cooperativo y dos con enfoque independiente. La estrategia cooperativa ejecuta de manera desacoplada y los robots se mueven siguiendo el paradigma jerárquico (sección 2.1.1). Entre las estrategias independientes una utiliza el paradigma reactivo y la otra el paradigma híbrido a la que se le agrega un paso previo de planificación del camino. Se visualiza la comparación en el Cuadro 3.1.

Descripción	**Paradigma**	**Tipo de Estrategia**
Campos Potenciales	Reactivo	Independiente
Path Coordination	Jerárquico	Cooperativo desacoplado
A^* con Campos Potenciales	Híbrido	Independiente

Cuadro 3.1: Estrategias implementadas

En el Capítulo 4 se muestran los resultados para poder evaluar sus fortalezas y debilidades en diferentes escenarios. El código correspondiente a la implementación de las estrategias se encuentra disponible en el repositorio Gitlab [1] de Facultad de Ingeniería, conteniendo además instrucciones y documentación del código.

3.2.1. Estrategia Reactiva

Se diseñó un algoritmo basado en Campos Potenciales (Sección 2.2.1) en donde cada robot se mueve independientemente y actúa según la información local que obtiene a través de sus sensores. Se utilizaron las fuerzas atractiva, repulsiva y tangencial, ya que para aplicar las fuerzas uniforme y perpendicular sería necesario conocer el entorno.

3.2.2. Movimiento

Para esta estrategia se definió la velocidad lineal constante, por lo que las fuerzas indican solamente la dirección a donde debe dirigirse el robot. Como consecuen-

[1] https://gitlab.fing.edu.uy/lucas.corazza/proyecto-grado

cia en cada ciclo de robot se sensa el entorno y se obtiene el ángulo θ que debe rotar el robot para llegar al punto objetivo.
Sea θ_t el ángulo formado entre el *heading* (vector a donde apunta el robot) del robot y el punto destino en determinado instante t. La velocidad angular w_{t+1} objetivo se calcula utilizando un control proporcional (Sección 2.3.6) con coeficiente K_p (3.1).

$$w_{t+1} = K_p \theta_t \tag{3.1}$$

El control integral no es utilizado debido a que cada punto a alcanzar se calcula en cada ciclo de robot, aprovechando al máximo la velocidad de respuesta. Si se utilizara el control integral sería necesario mantener fijo un punto destino durante n ciclos de robot, perdiendo reactividad, y como consecuencia incrementando la probabilidad de choque.

3.2.3. Sensado

Para sensar el entorno, dentro de los recursos de hardware disponibles en Webots utilizamos el lidar Robotis LDS-01 [27] (Figura 3.1). Un lidar (Laser Imaging Detection and Ranging) es un dispositivo que permite determinar la distancia desde un emisor láser a un objeto o superficie utilizando un láser que emite luz en forma de pulsos. El modelo utilizado posee un rango de lecturas entre 0.12m y 3.5m de distancia, una amplitud de 360° y una resolución de 1°. Se utilizó este sensor ya que permite una lectura del entorno precisa, con amplitud configurable y con un máximo de distancia adecuado al tamaño del robot. En particular comparando con los sensores del Khepera IV, éste permite obtener una mayor cantidad de lecturas.

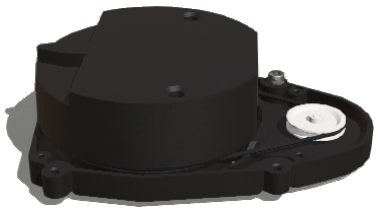


Figura 3.1: Robotis LDS 01

Con el fin de flexibilizar el testeo del algoritmo se decidió parametrizar la cantidad de medidas del lidar s_{cant} y su amplitud s_{amp}.
Dada una amplitud de sensado s_{amp}, se obtienen s_{cant} lecturas del sensor equiespaciadas centradas en el *heading*. En la Figura 3.2 se despliega un ejemplo donde $s_{amp} = 90°$ y $s_{cant} = 4$, de modo que se calculan 4 ángulos a leer del sensor espaciados por 30°.

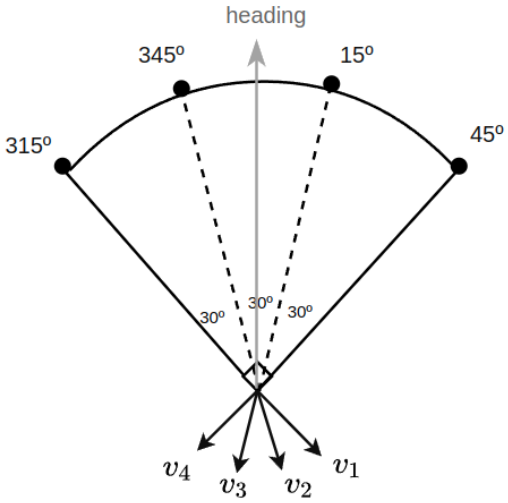


Figura 3.2: Ejemplo de lecturas del lidar con vectores repulsivos

Durante este proceso también se aprovecha a calcular vectores repulsivos normalizados correspondientes a cada lectura del sensor para utilizar durante el cálculo de las fuerzas repulsivas. En cada ciclo del robot se calcula la distancia registrada en cada lectura, en base a esto se calcula una fuerza, y se multiplica al vector correspondiente. Luego los vectores resultantes pueden ser sumados para obtener una fuerza repulsiva resultante, que debe ser rotada según la orientación del robot.

3.2.4. Análisis de fuerzas

En las Figuras 3.3, 3.4 y 3.5 se ilustran los campos potenciales actuando por separado y en conjunto, donde el punto atractivo está ubicado en $(x, y) = (9, 9)$ y los obstáculos ubicados en los puntos $(x, y) = (4, 5)$ y $(x, y) = (8, 5)$ los cuales generan un campo repulsivo de una unidad de radio.

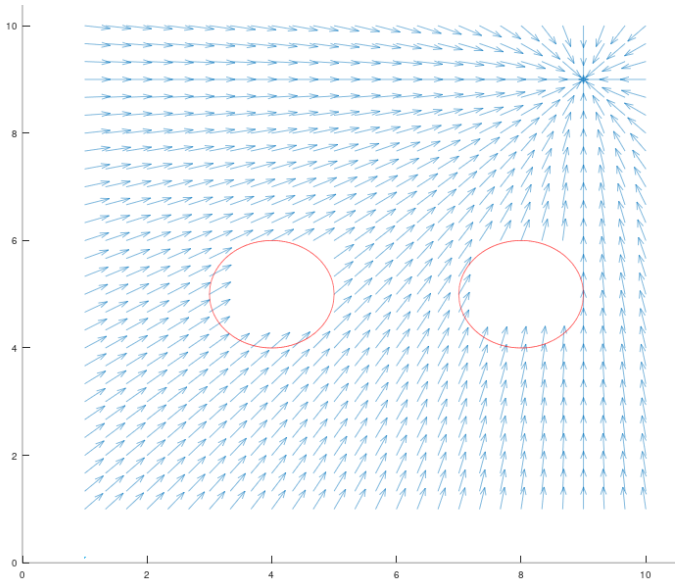


Figura 3.3: Campo de fuerza atractiva generada por punto (9,9)

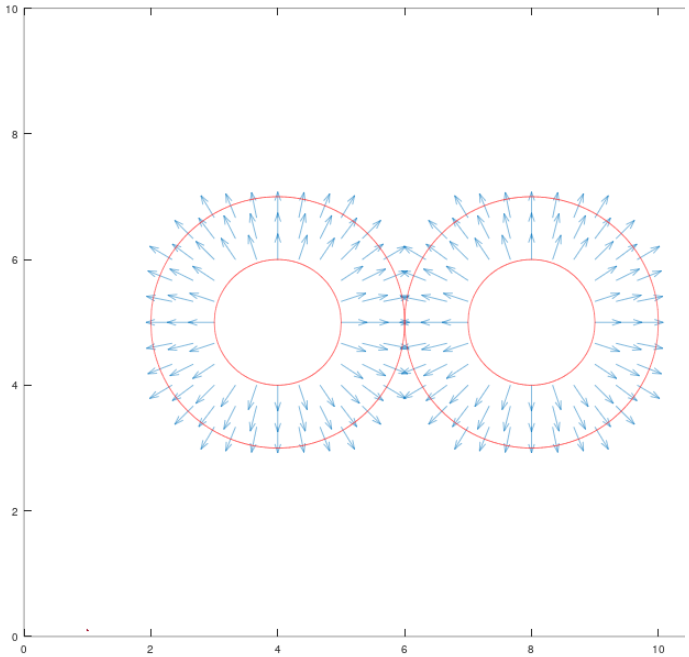


Figura 3.4: Campo de fuerza repulsiva generado por obstáculos en (4,5) y (8,5)

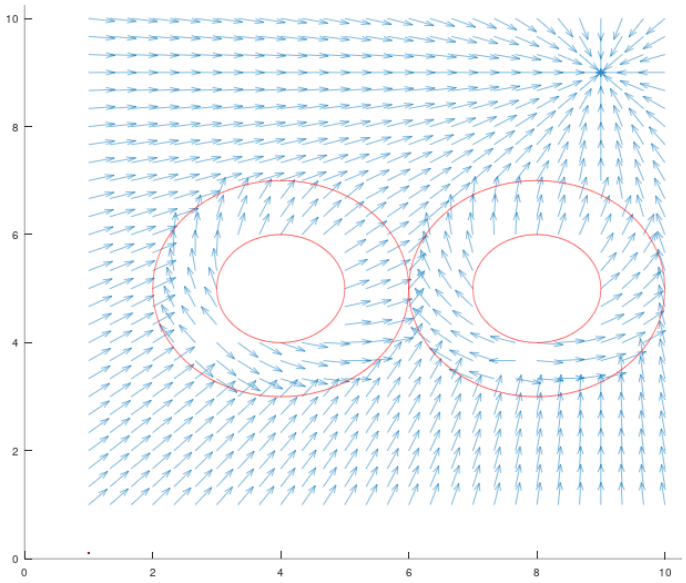
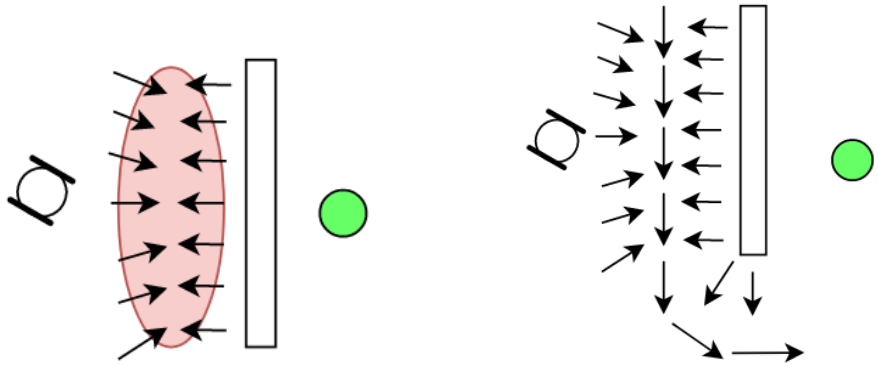


Figura 3.5: Campo potencial resultante de fuerzas atractivas y repulsivas

Evaluando varios escenarios se observó que utilizando solamente las fuerzas atractivas y repulsivas el robot no logró resolver muchos casos básicos. Esto ocurría debido a que el robot no lograba escapar de mínimos locales (Sección 2.2.1). Con el objetivo de resolver este conflicto, se agrega la fuerza tangencial, la cual actúa solamente entre d_{rep} y d_{tg} de distancia del obstáculo, permitiendo que éste rodee obstáculos o paredes.
En las figuras 3.6(a) y 3.6(b) se presenta un ejemplo donde el obstáculo (pared) se encuentra entre el robot y el punto de llegada (indicado en color verde). La posición inicial del robot se ubica a la izquierda del obstáculo.



(a) Ejemplo sin fuerzas tangenciales

(b) Ejemplo con fuerzas tangenciales

Figura 3.6: Ejemplo de fuerza tangencial

En el caso de la figura 3.6(a) el robot probablemente quede atrapado dentro de la zona roja. Sin embargo en el caso de la figura 3.6(b), añadiendo fuerzas tangenciales, hay posibilidades de que el robot llegue hasta el contorno inferior del obstáculo (o por el superior dependiendo de la orientación del robot) y pueda llegar al punto destino. Como resultado, en la Figura 3.7 se describen las fuerzas ejercidas por un obstáculo, donde d_{min} es la distancia mínima a la que el objeto puede ser detectado, d_{rep} es la distancia máxima de aplicación de la fuerza repulsiva y d_{tg} es la distancia máxima de aplicación de la fuerza tangencial.

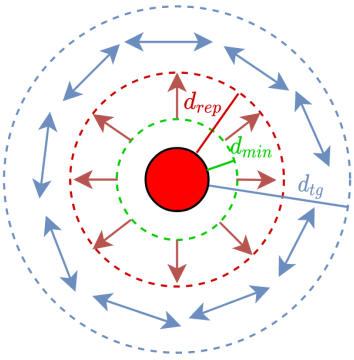


Figura 3.7: Fuerzas "generadas" por obstáculo según rangos de distancia

Para detectar si un robot llega a su destino se utiliza un margen de distancia $d_{llegada}$. A su vez, para facilitar la detención del robot en el punto destino, se utiliza un margen de distancia d_{vel} en el cual el robot desciende la velocidad hasta detenerse completamente (Figura 3.8).

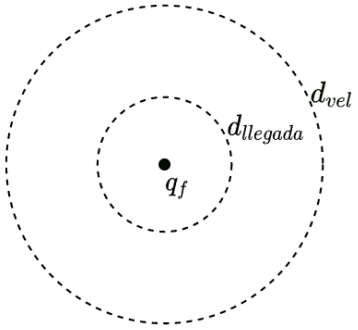


Figura 3.8: Margen de llegada al destino q_f ($d_{llegada}$) y margen para reducir la velocidad (d_{vel})

Sea (x_r, y_r) la posición del robot, se describen las fuerzas utilizadas:

Fuerza atractiva

Sea (x_g, y_g) la posición del destino y k_a una constante positiva se define $\vec{v_{at}}$ (3.2).

$$\vec{v_{at'}} = \begin{bmatrix} x_g - x_r \\ y_g - y_r \end{bmatrix}$$

$$\vec{v_{at}} = k_a * \frac{\vec{v_{at'}}}{||\vec{v_{at'}}||} \tag{3.2}$$

Como resultado, se obtiene un vector de magnitud constante que va desde la posición del robot al punto destino.

Fuerza repulsiva

Sea k_r una constante positiva y d la distancia al obstáculo sensado, se define la magnitud de la fuerza repulsiva f_{rep} a partir de la ecuación repulsiva de campos potenciales formulada en [13].

$$f_{rep}(d) = \begin{cases} k_r * (\frac{1}{d - d_{min}} - \frac{1}{d_{rep}})^2 \text{ si } d_{min} < d < d_{rep} \\ 0 \text{ sino} \end{cases}$$

Sea $\vec{v_{rep}}'$ un vector normalizado que va desde el obstáculo a la posición del robot, se obtiene el vector repulsivo $\vec{v_{rep}}$ (3.3).

$$\vec{v_{rep}} = f_{rep} * \vec{v_{rep}}' \tag{3.3}$$

Como resultado se obtiene un vector de magnitud inversamente proporcional a la distancia siempre que la misma sea menor a d_{rep} y mayor a

d_{min}. En la Figura 3.9 se observa un ejemplo de la curva creada por la magnitud de la fuerza repulsiva.

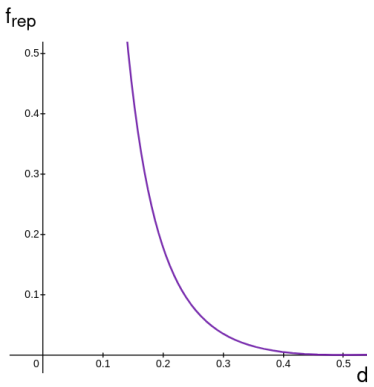


Figura 3.9: Gráfica de fuerza respulsiva. $d_{rep} = 0.5$, $k_r = 0.1$, $d_{min} = 0.12$

La utilidad de la función utilizada es que la curva crece lentamente mientras se aleja de la distancia máxima (0.5), pero al acercarse a la distancia mínima esta crece asintóticamente. En la teoría esto permite que obstáculos "lejanos" que estén dentro del radio repulsivo puedan efectuar leves correcciones a la dirección del robot, mientras que los obstáculos cercanos sensados puedan ejercer una corrección drástica a la dirección del robot.

Fuerza tangencial

Sea $\vec{v_o}$ un vector que va desde la posición del robot hasta el obstáculo sensado,

$$\vec{v_o} = \begin{bmatrix} x_o \\ y_o \end{bmatrix}$$

se definen $\vec{v_\perp}$ y $-\vec{v_\perp}$ las rotaciones de $\vec{v_o}$ de 90° en sentido horario y antihorario respectivamente:

$$\vec{v_\perp} = \begin{bmatrix} -y_o \\ x_o \end{bmatrix}$$

Para elegir el vector perpendicular más cercano al *heading*, se calcula el ángulo ϕ entre el *heading* del robot y $\vec{v_\perp}$ (3.4).

$$\phi = arcos(\frac{\vec{v_\perp} * \vec{v_h}}{||\vec{v_\perp}|| * ||\vec{v_h}||}) \tag{3.4}$$

En la Figura 3.10 se ilustra un escenario en donde se debe eligir el vector perpendicular más cercano al *heading*. En dicho caso de debe elegir el vector $\vec{v_\perp}$.

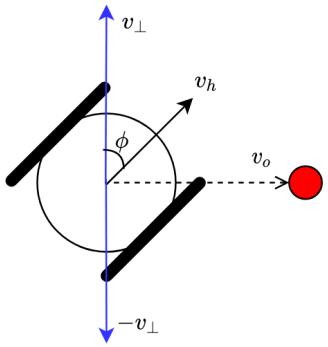


Figura 3.10: Elección de fuerza tangencial

Sea k_{tg} una constante positiva, se obtiene la fuerza tangencial expresada en (3.5).

$$\vec{v_{tg}} = \begin{cases} k_{tg} * \vec{v_\perp} \text{ si } \phi \leq \pi/2 \\ -k_{tg} * \vec{v_\perp} \text{ sino} \end{cases} \tag{3.5}$$

Esta fuerza tiene una magnitud constante sobre el cálculo de la fuerza resultante cuando el objeto sensado se ubique en una franja de distancia determinada.

Para calcular el punto resultante a donde debe moverse el robot, se aplica (3.6).

$$\vec{v_{resultante}} = \vec{v_{at}} + \vec{v_{rep}} + \vec{v_{tg}} \tag{3.6}$$

Para calcular la fuerza repulsiva y tangencial es necesario obtener la distancia al obstáculo para cada sensor, por lo que el vector resultante también se puede definir como (3.7), siendo s_{cant} la cantidad de lecturas del sensor.

$$\vec{v_{resultante}} = \vec{v_{at}} + \sum_{i=1}^{s_{cant}} \vec{v_{rep_i}} + \sum_{i=1}^{s_{cant}} \vec{v_{tg_i}} \tag{3.7}$$

3.2.5. Arquitectura

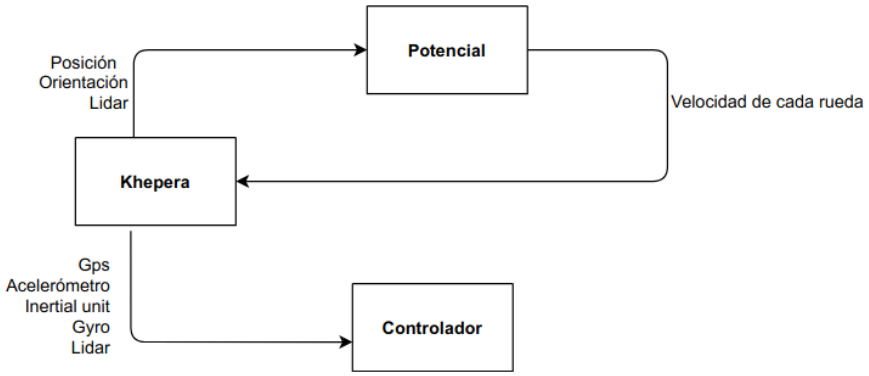


Figura 3.11: Aquitectura potencial

En la Figura 3.11 se muestra un diagrama con los nodos implicados en la arquitectura. El nodo *Khepera* es creado por Webots y es utilizado para interactuar con el robot a través de ROS. El nodo *Controlador* es el encargado de ejecutar el timestep, habilitar los sensores y aplicar configuraciones. El nodo *Potencial* es el encargado de procesar la información del lidar, calcular la velocidad angular objetivo, calcular velocidades de cada rueda y asignárselas al robot por medio de un tópico publicado por el nodo *Khepera*.

3.2.6. Algoritmo

En el Pseudocódigo 3.1 se resume el ciclo principal del flujo del robot.

Pseudocódigo 3.1: Algoritmo de Campo Potencial

```
1  vector<float> punto_final;
2  float d_vel;
3  while(ros::ok()) { // Mientras no se quiera terminar la
       ejecucion
4      esperarTimestep();
5      info_sensada = sensar();
6      angulo_actual = obtenerAnguloActual();
7      punto_actual = obtenerPuntoActual();
8      if (distancia(punto_actual, punto_final) < d_llegada) {
9          break;
10     }
11     if (distancia(punto_actual, punto_final < d_vel)) {
12         bajarVelocidadRobot();
13     } else {
14         subirVelocidadRobot();
```

```
15      }
16      siguiente_punto = obtenerPunto(punto_actual,
            angulo_actual, punto_final, info_sensada);
17      moverseHaciaPunto(siguiente_punto);
18  }
19  detenerRobot();
```

Notar que la función *moverseHaciaPunto* no espera a llegar al punto deseado, sino que envía la actuación a los motores y el loop se vuelve a ejecutar en el siguiente timestep. Además el robot solamente se detiene si llega al punto destino ejecutando la función *detenerRobot*, en caso contrario sigue moviéndose.
En el Pseudocódigo 3.2 se describe a grandes rasgos la función *obtenerPunto* para el cálculo de los puntos intermedios a calcular a partir de las fuerzas potenciales.

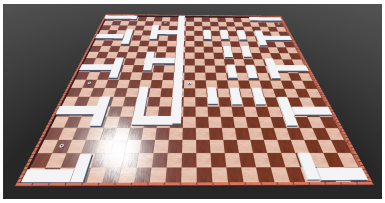
Pseudocódigo 3.2: Función obtenerPunto para calcular punto resultante de fuerzas potenciales

```
1   vector<float> obtenerPunto(
2       vector<float> punto_actual,
3       float angulo_actual,
4       vector<float> punto_final,
5       vector<Data> info_sensada)
6
7   {
8       v_res = {0, 0};
9       for (int i = 0; i < info_sensada.size(); i++) {
10          if (info_sensada[i].distancia < d_rep) {
11              v_res += obtenerVector(i) *
                    fuerzaRepulsiva(info_sensada[i].distancia);
12          } else if (sensores[i].distancia < d_tg) {
13              v_res += obtenerVector(i) *
                    fuerzaTangencial(info_sensada[i].distancia) *
                    rotacion(90);
14          }
15      }
16      v_res = v_res * rotacion(angulo_actual);
17
18      v_atraccion = (punto_final - punto_actual);
19      v_atraccion = v_atraccion * fuerzaAtraccion();
20
21      v_res = v_res + v_atraccion;
22      v_res = v_res + punto_actual;
23
24      return v_res;
25  }
```

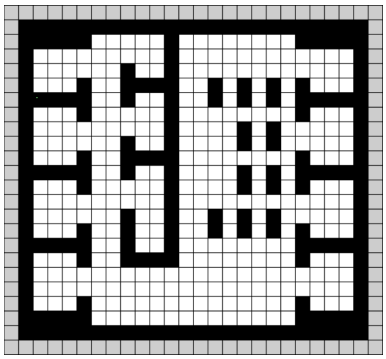
La función *obtenerVector* obtiene el vector correspondiente al sensor de la iteración, calculado inicialmente cuando se configuran los ángulos de sensado (Sección 3.2.3).

3.2.7. Estrategia Cooperativa

La estrategia cooperativa implementada se diseñó a partir de una de las estrategias de *Path Coordination* presentada en la Sección 2.2.2, basada en [29]. La estrategia consiste en la coordinación del movimiento de los robots por sus respectivos caminos, los cuales son planificados independientemente, y se mantienen fijos en la coordinación. Cada robot planifica un camino propio, luego se buscan las secciones de los caminos en los que ocurren conflictos y se genera un diagrama de coordinación, donde los ejes representan la posición de cada robot a lo largo de sus respectivos caminos. Para llevar a cabo la coordinación, se busca un camino entre el punto inicial del diagrama (en el que todos los robots se encuentran en su posición inicial) y el punto final (donde todos los robots han terminado su recorrido), evitando las áreas de conflicto, que son tratadas como obstáculos. La diferencia con dicha estrategia, radica en que se decidió trabajar con espacios discretos en lugar de espacios continuos, tanto para la planificación de los caminos, como para la coordinación del recorrido por los mismos de manera de simplificar el problema.
En particular, para la representación del ambiente por el que se mueven los robots, se utiliza una grilla de ocupación bidimensional. En esta representación el ambiente se divide en cuadrantes de tamaño constante, para los cuales se conoce si están libres u ocupados por algún obstáculo. Los caminos se planifican como una serie cuadrantes que conectan al cuadrante inicial con el cuadrante final.



(a) Ambiente de ejemplo en el simulador Webots



(b) Representación en grilla de ocupación

Figura 3.12: Representación de un mapa

Para la coordinación de los robots, se consideran bloques de tiempo que llamamos "turno", cuya duración se asocia a la cantidad de tiempo que tarda un robot en atravesar la diagonal de un cuadrante moviéndose a velocidad máxima. Este valor se eligió ya que recorrer una celda de punta a punta por la diagonal es el trayecto más largo que puede realizar un robot al atravesar un cuadrante.

3.2.8. Arquitectura

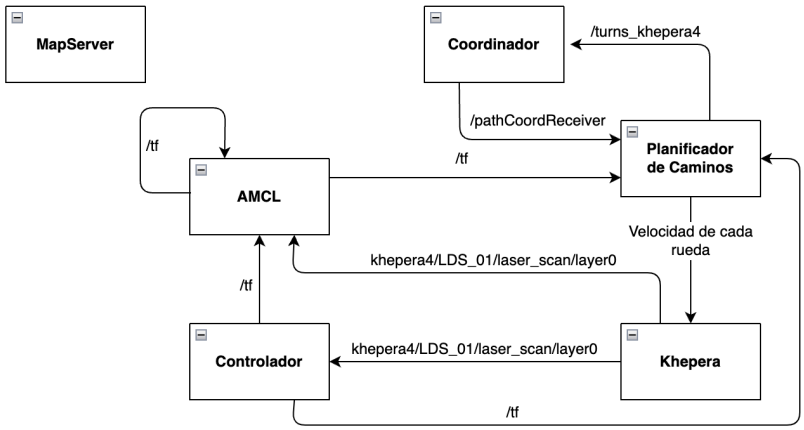


Figura 3.13: Arquitectura Cooperativo

En la Figura 3.13 se muestra un diagrama con los nodos que forman la arquitectura de la estrategia cooperativa. Las funciones de cada uno de estos serán explicadas en las siguientes secciones.

3.2.9. Paquetes Utilizados

Gmapping [10] [2]

Provee datos de localización y cartografía en línea (SLAM - *Simultaneous Localization and Mapping*) basado en lásers como un nodo de ROS llamado *slam gmapping*. Esto permite, mediante los valores obtenidos por un sensor lidar colocado sobre un robot móvil y su ubicación, generar un mapa de grilla de ocupación en 2D.

Map Server [20]

Provee la información de los mapas como un servicio de ROS. También provee el nodo *map saver* el cual permite persistir los mapas generados por el paquete *gmapping*, generando los archivos de terminación *pgm* y *yaml*.

AMCL [2] Provee un sistema de localización probabilístico para los robots moviéndose en ambientes bidimensionales. Implementa el método de localización de Monte Carlo adaptativo [7] que utiliza un filtro de partículas

[2] Los mapas generados tenían un error de corrimiento de los obstáculos respecto a la realidad, problema que no se pudo concluir si era de configuración ó de paquete, por lo que se optó por corregir dicho corrimiento modificando manualmente los archivos generados.

para determinar la posición de un robot respecto a un mapa conocido.

En el momento de ejecutar la estrategia, los archivos generados por el nodo *map saver* se cargan utilizando el nodo *map server* que se encarga de publicar tópicos y servicios para que los demás nodos puedan acceder a la información del mapa.

Para ubicar a los robots en el mapa, se utiliza el paquete *amcl*, el cual requiere como parámetros la posición y rotación inicial del robot, el tópico en que se publican las medidas del sensor lidar del robot, y los nombres del marco de odometría y del marco base.

3.2.10. Tópicos, Servicios y Mensajes

A continuación se listan los mensajes y tópicos utilizados para el proceso de planificación de caminos y coordinación de los robots.

Servicios

/static_map: Mensajes de tipo **nav_msgs::GetMap**. Recibe la matriz de ocupación y los meta-datos del mapa.

Tópicos

/model_name: Mensajes de tipo String. Recibe el nombre de cada robot del sistema.

/pathCoordReceiver: Mensajes de tipo **path_with_id**. Se utiliza para comunicarle los caminos planificados por cada robot al nodo de coordinación.

/turns_<RobotName>: Mensajes de tipo **turns**. Existe uno de estos tópicos por cada robot. Se utiliza para enviar desde el nodo coordinador a los controladores de cada robot tanto los índices de los caminos coordinados como también cuantos turnos tiene que llevarles recorrer cada trayecto.

/map_metadata: Mensajes de tipo **nav_msgs::MapMetaData**. Recibe los meta-datos del mapa.

3.2.11. Generación de los Caminos

Para cada robot se inicializa un nodo llamado *PlanificadorDeCamino* que se encarga de la lógica de navegación del robot, incluyendo planificar el camino, comunicarse con el nodo coordinador y mover al robot hasta cada punto del camino.

Recibe como parámetros:

robot_name: identificador del robot.

base_frame_id: nombre del marco de referencia de la base del robot.

pathDestinationX: valor del eje x del cuadrante destino del robot.

pathDestinationY: valor del eje y del cuadrante destino del robot.

unique_launch_file: booleano que define si debe esperar **node_start_delay** segundos para ejecutar el nodo.

node_start_delay: tiempo a esperar para que se ejecute el nodo si la variable **unique_launch_file** está apagada.

Se obtienen los datos del mapa utilizando el servicio publicado por el nodo *map server* y se obtienen las coordenadas del robot en el marco de mapa utilizando la funcionalidad *lookupTransform* del paquete *tf*. Con estos datos, se puede obtener el cuadrante de la grilla en que se encuentra el robot, restándole a las coordenadas del mismo la posición del origen del mapa, y dividiendo entre el valor de resolución (es decir, el valor del tamaño del lado de los cuadrantes).

Podemos pensar en la grilla como un grafo de forma que cada cuadrante es un nodo, y cada nodo se comunica con los 8 nodos que lo rodean (los cuadrantes a los 4 lados, y los 4 que se encuentran en diagonal).
Los nodos se consideran libres u ocupados de acuerdo al valor de ocupación del cuadrante del mapa. Es decir, si en el nodo existe un obstáculo o un robot, el mismo se considera como ocupado. De lo contrario se lo considera libre.

Tomando a modo de ejemplo la Figura 3.14 con los 4 movimientos básicos (horizontales y verticales), los cuadrantes superior, lateral izquierdo y lateral derecho del nodo que contiene al robot están libres, y el robot podrá moverse a esas posiciones. No es el caso de la posición inferior, puesto que ahi se observa un obstáculo, lo que impide el movimiento y se marca a la celda como ocupada.

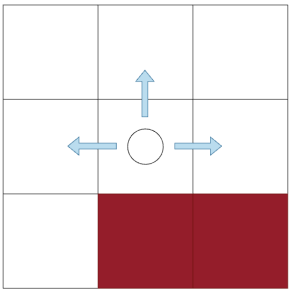


Figura 3.14: Los 4 movimientos a los lados del robot

Para poder realizar un movimiento en diagonal, se debe revisar que estén libres ambos cuadrantes a los costados del movimiento en diagonal. Si uno de estos está bloqueado por un obstáculo, se considerará bloqueado el movimiento en diagonal también (Figura 3.15). Esto se debe a que el robot no es un punto, sino que tiene un cierto diámetro, por lo que no puede pasar por el punto en que se tocan las esquinas de los cuadrantes sin ocupar parte de los cuadrantes a los costados. Si bien el hecho de que un cuadrante esté marcado como ocupado no implica que toda su área esté ocupada, no se puede garantizar que la parte por la que pasaría el robot esté libre, por lo que optamos por evitar éste tipo de movimientos, y así poder asegurar que no ocurran colisiones.

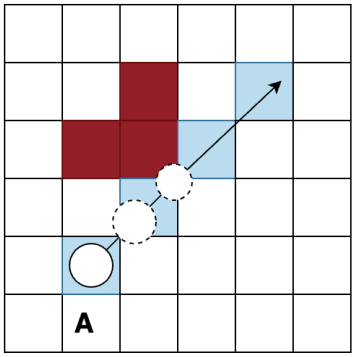


Figura 3.15: Colisión entre Robot A y el obstáculo. Como el Robot A tiene diámetro, no puede pasar por la esquina sin pasar por arriba de un cuadrante marcado como obstáculo, por lo que no se puede asegurar que no ocurran colisiones.

De esta forma, podemos aplicar un algoritmo de *Path Finding* que permita encontrar un camino en el grafo entre el nodo inicial (el nodo asociado al cuadrante en que se encuentra el robot) y el nodo final (el nodo asociado al cuadrante indicado por las coordenadas que se recibieron como parámetro).

Se optó por utilizar el algoritmo A^* para encontrar el camino, ya que es una opción flexible y popular en la comunidad para encontrar caminos en grafos. El algoritmo A^* es un algoritmo de búsqueda en grafos de tipo heurístico que encuentra, siempre que se cumplan condiciones específicas, el camino de menor costo entre 2 nodos de un grafo ponderado [1].

El algoritmo A^* devuelve la secuencia de cuadrantes que se deben visitar para llegar desde el cuadrante inicial hasta el cuadrante final por el camino más corto.

En el Pseudocódigo 3.3 se presenta cómo se realiza la planificación del camino, desde el momento inicial del sistema al momento de enviar el camino al nodo

coordinador.

Pseudocódigo 3.3: Algoritmo de robot para coordinar caminos estrategia Cooperativa

```
1       obtenerParametros();
2       obtenerNombresControladores();
3
4       mapa = obtenerMapa();
5       generador = setearMundo(mapa);
6       camino = generarCamino(generador, puntoInicial,
            puntoFinal);
7
8       enviarPathACoordinador(camino);
```

3.2.12. Coordinación

La coordinación de los robots es llevada a cabo por el nodo coordinador. Este es el encargado de revisar los caminos de los robots para detectar potenciales conflictos y crear un plan que los evite.

El nodo coordinador recibe los siguientes parámetros:

- **robot_amount**: Cantidad de robots del caso.
- **select_tile_logic_automatically**: Permite que el sistema, permita o bloquee movimientos que causen que dos robots coincidan en un mismo cuadrante durante un trayecto (uno puede entrar mientras el otro se está yendo del cuadrante) según el tamaño del robot y el tamaño de los cuadrantes al momento de encontrar el camino a la meta.
- **use_large_tile_logic**: Siempre trata al sistema como si hubiera espacio dentro del cuadrante para coincidir dos robots en el mismo.
- **use_small_tile_logic**: Siempre trata al sistema como si no hubiera espacio dentro del cuadrante para coincidir dos robots en el mismo.
- **node_start_delay**: Tiempo de espera para ejecutar el nodo en tiempo de ejecución.
- **unique_launch_file**: Indica si el sistema se ejecuta desde un único archivo de launch o varios.

Una vez generados los caminos, los nodos de PlanificadorDeCamino publican los caminos, junto con el identificador del robot asociado, en el tópico **/pathCoordReceiver** (Figura 3.16).

El nodo coordinador recibe los caminos e identificadores a través del tópico **/pathCoordReceiver** y espera a recibir todos los caminos.

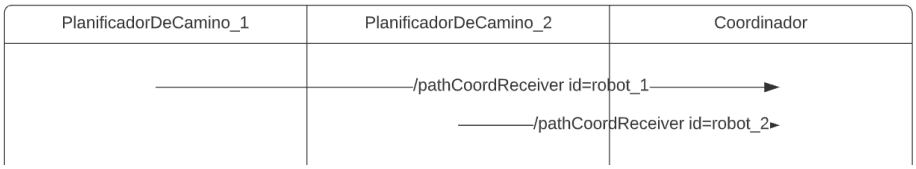
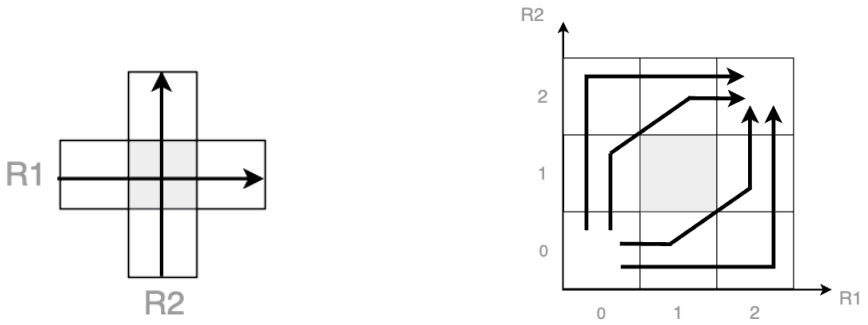


Figura 3.16: Diagrama de flujo de planificación de caminos

Diagrama de coordinación

Al igual que en la estrategia presentada en la sección 2.2.2 de **Path Coordination** [29], se utiliza la información de los conflictos para construir un diagrama de coordinación, donde cada eje representa la posición de un robot en su respectivo camino. La diferencia en esta estrategia consiste en que los caminos no se representan como continuos, sino como una lista de cuadrantes por los que pasa el robot. Por lo tanto, el valor en un eje indica el índice en la lista de cuadrantes de la grilla que forman su camino, en el que se posiciona el robot. Esto lleva a que el diagrama de coordinación se pueda ver como una cuadrícula.



(a) Caminos de los robots

(b) Diagrama de coordinación. La celda gris representa la celda conflictiva. Las flechas indican posibles soluciones al problema de coordinación.

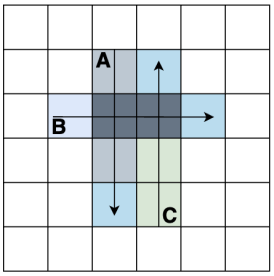
Figura 3.17: Ejemplo de colisión en Diagrama de Coordinación de los Robots R1 y R2

Consideremos el ejemplo presentado en la Figura 3.17 en que tenemos dos robots R_1 y R_2. Los caminos de ambos robots son de tres cuadrantes de largo (incluyendo el cuadrante inicial y el cuadrante final) y coinciden en el segundo cuadrante, por lo que existe una posible colisión en el punto (1,1) del diagrama de coordinación, ya que nuestro objetivo es que cada robot llegue al final de su camino. Por lo tanto, podemos tratar la coordinación de los robots como un problema de *Path Finding* en el que buscamos un camino entre el punto (0,0)

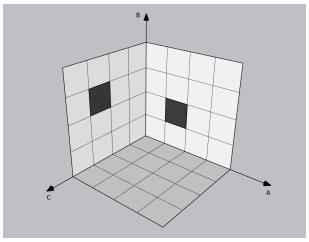
y el (2,2), evitando el conflicto que se encuentra en el punto (1,1).

Si en determinado camino de un diagrama de coordinación alguna coordenada disminuye, esto implica retroceder en el camino del robot. En dichos casos se implementó un movimiento en reversa, ya que girar sobre sí mismo generaba un retraso que afectaba la coordinación.

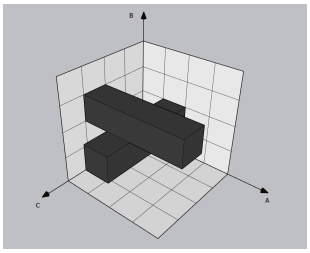
Para n robots, el diagrama de coordinación generalizado toma la forma de un prisma n-dimensional dividido en cubos n-dimensionales. Los cuadrantes marcados como obstáculos en los diagramas bi-dimensionales definidos por cada par de robots se convierten en obstáculos n-dimensionales al proyectarlos mediante el producto cartesiano del cuadrante con el resto de los ejes. En la Figura 3.18 se presenta un ejemplo de diagrama de coordinación para 3 robots, y como los conflictos se proyectan desde los planos al resto del espacio.



(a) Caminos de los robots. Las flechas indican los caminos a seguir por los robots. Las celdas de color gris oscuro representan los cuadrantes compartidos de sus caminos entre al menos dos robots.



(b) Zonas de conflicto en los planos del diagrama de coordinación entre las celdas compartidas de los robots A y B, y B y C respectivamente.



(c) Diagrama de coordinación con las zonas de conflicto proyectadas en el ambiente

Figura 3.18: Ejemplo de diagrama de coordinación para tres robots

Una solución al problema de coordinación es un camino libre de colisiones desde

la celda inicial (0, 0, ..., 0) hasta la celda $(l_1, l_2, ..., l_n)$ donde l_i es la cantidad de pasos en el camino del robot i.

Detección de conflictos

Al igual que en [29], esta búsqueda se realiza de a pares de caminos. Es decir para el camino de cada robot, se buscan conflictos con los caminos asociados a los otros $n-1$ robots, siendo n la cantidad total de robots. Como cada camino es una lista de cuadrantes por los que pasará el robot, lo que se busca son cuadrantes que aparezcan en ambos caminos. Estos conflictos se guardan como una tupla, indicando los identificadores de los robots y los índices del cuadrante en que puede ocurrir el conflicto entre ambos caminos.

(a) Caminos de los robots

(b) Diagrama de coordinación

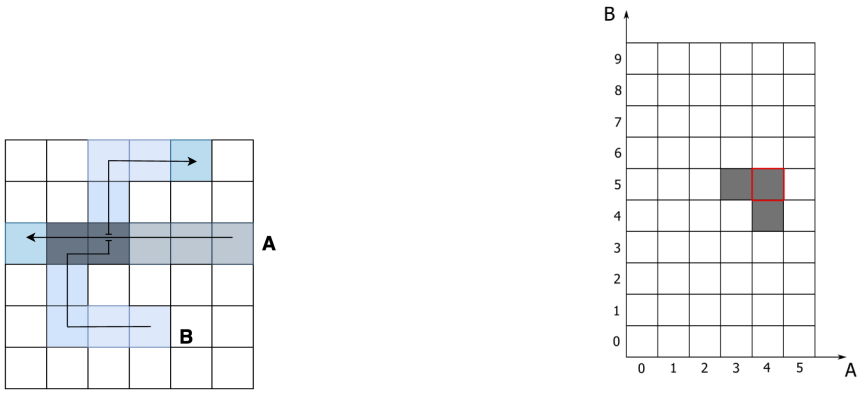
Figura 3.19: Ejemplo de conflicto entre robot A y robot B.

En el ejemplo presentado en la Figura 3.19(a) podemos ver que se da un conflicto entre el robot A y el robot B en la posición (4, 4). La tupla que se guarda es de la forma <idRobot1, idRobot2, posRobot1, posRobot2>. Entonces en este caso se guardaría la tupla <A, B, 4, 4>, pues el conflicto se da en ambos casos en el cuadrante 4 de su camino (el primer cuadrante del diagrama de coordinación es el índice 0).

También se considera como conflicto el caso de "intercambio de posiciones". Dados dos robots A y B, si al evaluar el cuadrante A_i del camino del robot A con el cuadrante B_j del camino del robot B, se detecta que el cuadrante A_i coincide con la posición del cuadrante B_{j-1} y el cuadrante B_j coincide con la posición del cuadrante A_{i-1}, se debe registrar un conflicto.
Tenemos dos objetos que ocupan celdas contiguas en la grilla que quieren intercambiar posiciones y eso físicamente no se puede resolver porque al moverse por un mismo camino en direcciones opuestas en algún momento van a colisionar.

Teniendo esto en cuenta optamos por marcar la posición resultante de si ambos intercambiaran posiciones como conflicto.

Tomemos el ejemplo de la Figura 3.20. Se dará esta situación de que estén contiguos en la grilla y quieran intercambiar posiciones cuando el robot A esté en la posición 3 en el Diagrama de Coordinación, y el robot B esté en la posición 4 (3.20(b)).



(a) Caminos de los robots

(b) Diagrama de coordinación

Figura 3.20: Ejemplo de conflicto entre robot A y robot B en 2 cuadrantes, con sentidos contrarios. En (b) aparece marcado con rojo el conflicto definido por el "intercambio de posiciones".

La estrategia de marcar esta configuración como conflicto se eligió por su simplicidad, pero se debe destacar que genera una limitación de la estrategia ya que la configuración en si no es conflictiva, sino que solo el movimiento que lleva a ella lo es. Los robots podrían hipotéticamente llegar a dicha configuración de otra manera, si uno o ambos robots llegaran hasta esa posición retrocediendo. Esto puede llevar a que haya casos en los que exista una solución para la coordinación, pero esta no se encuentre ya que no se consideran posibles estos movimientos.

Al igual que ocurrió en la búsqueda de caminos, se deben tomar en cuenta los cuadrantes involucrados al realizar un movimiento en diagonal (Figura 3.21). La primera opción que se decidió implementar fue marcar como conflicto cuando en un camino se detecta un movimiento en diagonal, y otro camino intersecta a uno de los cuadrantes por los que pasaría el robot. Sin embargo, esta opción limita innecesariamente los movimientos posibles. Es solo el movimiento diagonal el que no se puede realizar si el otro robot se encuentra en uno de esos cuadrantes, mientras que el otro robot sí está libre de moverse al cuadrante antes o después

de que se realice el movimiento en diagonal.

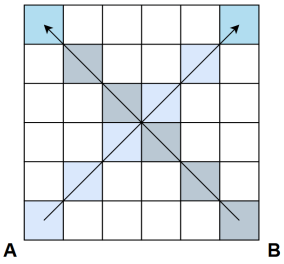
(a) Caminos de los robots

(b) Diagrama de coordinación

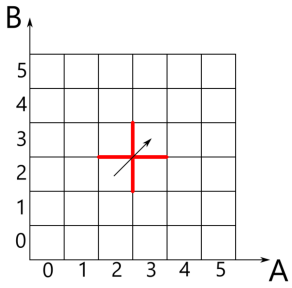
Figura 3.21: Al realizar un movimiento diagonal, el robot A pisaría uno de los cuadrantes del camino del robot B. Por lo que se define un obstáculo de borde que previene que el robot A se mueva desde el cuadrante 2 al cuadrante 3 de su camino mientras el robot B esté en el cuadrante 2 de su propio camino.

Por lo tanto se optó por implementar "obstáculos de borde" (Figura 3.21), que únicamente previene el movimiento del robot que realiza el desplazamiento en diagonal. Estos se guardan en una tupla que además de contener los identificadores de los dos robots, contienen los índices de los cuadrantes antes y después del robot que realiza la diagonal, y el índice del cuadrante del otro robot.

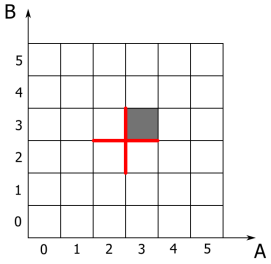
Por último, se debe considerar el caso de "cruce de diagonales", que se da cuando los trayectos diagonales de dos caminos se intersecan en un mismo vértice. Al igual que se hizo para el caso de "intercambio de posiciones", se optó por marcar la configuración resultante de dichos movimientos como conflicto, si bien esto nuevamente limita el espacio de soluciones. En el ejemplo presentado en la Figura 3.22, vemos que se da este caso cuando los robots quieren pasar desde el segundo cuadrante de su respectivo camino al tercero, por lo que se marca como conflicto el cuadrante del diagrama de coordinación que representa la configuración en que ambos robots se encuentran en el tercer cuadrante de sus respectivos caminos.



(a) Caminos de los robots



(b) Diagrama de coordinación. La flecha muestra el movimiento que no se debe permitir, ya que genera una colisión.



(c) Diagrama de coordinación con obstáculo agregado para evitar el movimiento problemático.

Figura 3.22: Ejemplo de conflicto de tipo "cruce de diagonales". El mismo ocurre al cruzarse los caminos de los robots en el vértice entre sus respectivos cuadrantes 2 y 3. Se marca como conflicto la configuración en que ambos robots se encuentran en el tercer cuadrante de su camino.

Búsqueda de caminos en el diagrama de coordinación

Al igual que en la búsqueda de caminos para los robots, se utilizó el algoritmo A^* para la búsqueda del camino en el diagrama de coordinación generalizado.

Como se mencionó previamente, el valor en cada eje indica la posición del robot respecto a su camino planificado. Por lo tanto, avanzar un paso respecto a un eje es equivalente a que el robot asociado se mueva al siguiente cuadrante de su camino.

Se permiten los movimientos en diagonal, ya que moverse en diagonal en el diagrama de coordinación significa mover a más de un robot a la vez. Nuevamente se debe tomar una decisión respecto a los movimientos diagonales que pasen junto a un obstáculo (que puede ser tanto una celda completamente bloqueada

al asociarse a un conflicto, o un "obstáculo de borde" asociado a un conflicto en un movimiento diagonal). Estos casos son equivalentes a que un robot llegue al borde del cuadrante conflictivo al mismo tiempo que el otro sale del mismo.

Los robots tardan un cierto tiempo en trasladarse, por lo que pueden llegar a ocurrir colisiones en la ventana de tiempo en que ambos robots se encuentran en el mismo cuadrante. Dado que no se permite que un robot entre y otro salga al mismo tiempo por el mismo punto borde del cuadrante, este caso resulta menos probable cuanto más grandes sean los cuadrantes en comparación al tamaño de los robots. Por esta razón, se puede configurar si permitir o no estos movimientos con las variables **use_large_tile_logic**, **use_small_tile_logic**, y **select_tile_logic_automatically**.

Las variables **use_large_tile_logic** y **use_small_tile_logic** niegan o permiten estos movimientos **siempre**. En cambio la variable **select_tile_logic_automatically** permite o no los movimientos basándose en el tamaño del robot y el tamaño de los cuadrantes. Si el radio del robot es menor a un cuarto del largo del lado de los cuadrantes, se permiten los movimientos mencionados. Esto se debe a que los casos en que los robots pasan más cerca entre sí, son aquellos donde un robot pasa por el centro de un lado, y el otro pasa por una de las puntas del cuadrante asociadas a ese mismo lado. Como se puede ver en la Figura 3.23, al encontrarse en los puntos de cruce, los centros de los robots se encuentran a medio largo de lado de distancia, por lo que para que no ocurra un choque, el radio de los robots debe ser más chico que un cuarto del largo del lado.

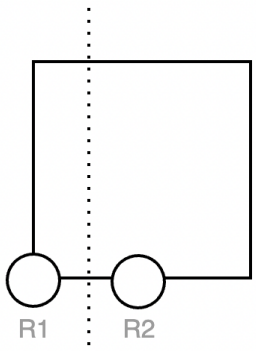


Figura 3.23: Permitir movimientos basándose en el tamaño de los cuadrantes y robot

El camino obtenido del diagrama de coordinación representa una lista de turnos, indicando el índice del cuadrante en el camino de cada robot en cada turno. Esta lista se procesa para obtener, por cada robot, una lista que indica el orden en que se visitan estos cuadrantes (lo cual es importante en casos en los que un

robot tiene que retroceder en su camino) y una lista indicando la cantidad de turnos que debe pasar en cada cuadrante, siendo un turno el valor constante de tiempo durante el cual un robot debe permanecer en una celda caso de no haber conflictos.
Si tomamos como ejemplo el caso presentado en la Figura 3.17, tenemos que un camino posible (asumiendo que se permiten los movimientos en diagonal rozando obstáculos) es:

$$(0,0) \longrightarrow (1,0) \longrightarrow (2,1) \longrightarrow (2,2)$$

Luego se obtiene para el robot R1 las siguientes listas:

- Índices: $0 \longrightarrow 1 \longrightarrow 2$
- Turnos: $1 \longrightarrow 1 \longrightarrow 2$

y para el robot R2 se obtiene:

- Índices: $0 \longrightarrow 1 \longrightarrow 2$
- Turnos: $2 \longrightarrow 1 \longrightarrow 1$

Para ambos robots se obtienen listas de índices iguales, ya que la coordinación no requiere que ninguno de los dos deba dar marcha atrás en ningún momento, y recorrerán sus caminos en orden. En cuanto a las listas de turnos, podemos ver que el robot R1 avanza en el primer turno al segundo cuadrante de su camino, y en el siguiente turno al tercero, por lo que pasa un turno en el primer y segundo cuadrante, y luego permanece en el cuadrante final hasta que el robot R2 complete su recorrido, es decir dos turnos. En cambio el robot R2 debe esperar a que el robot R1 salga del cuadrante conflictivo antes de poder ingresar al mismo, por lo que debe pasar dos turnos en el primer cuadrante, y luego avanzar por su camino hasta llegar a su meta final.
Estas listas se envían a través de un tópico identificado con los identificadores de los robots a los nodos **PlanificadorDeCamino** que se encargarán de la ejecución del plan.

Con el fin de optimizar el proceso de coordinación, se separaron los robots en grupos de coordinación para reducir la dimensión del espacio en que se realiza la búsqueda del camino. Los robots se agrupan entre los que tienen conflictos entre sí. A los robots que no tienen conflictos con ningún otro se les comunica que avancen por sus caminos a una velocidad de un cuadrante por turno. Por otro lado, para cada uno de los grupos de coordinación, se aplica el proceso de coordinación descrito previamente (sin considerar a los robots que no pertenezcan al grupo).

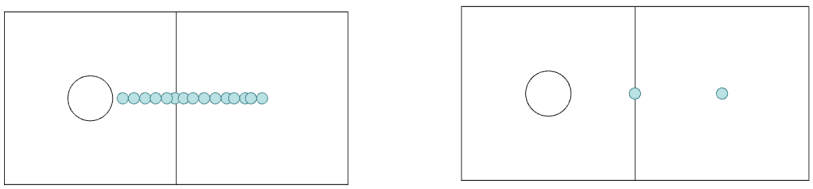
En el Pseudocódigo 3.4 se describe la coordinación entre los distintos caminos de los robots a sus respectivos destinos.

Pseudocódigo 3.4: Algoritmo de coordinación de caminos para estrategia Cooperativa

```
1     caminos = obtener_caminos();
2     grupos, colisiones = buscar_colisiones(caminos);
3     for (int i = 0; i < grupos.size(); i++)
4     {
5         grupo = grupos[i]
6         if (grupo.size() == 1)
7         {
8             robot = grupo[0];
9             indices, turnos = generar_listas_max_vel(robot);
10            enviar_listas(robot, indices, turnos);
11        }
12        else
13        {
14            camino_coordinacion = buscar_camino(grupo,
                  colisiones[grupo])
15            for (int j = 0; j < grupo.size(); j++)
16            {
17                robot = grupo[j];
18                indice, turnos = generar_listas(robot,
                      camino_coordinacion);
19                enviar_listas(robot, indices, turnos);
20            }
21        }
22    }
```

3.2.13. Ejecución del Camino

Para calcular el movimiento a realizar y alcanzar las metas, se utiliza el control PI (proporcional e integral), el cual se puede utilizar debido a que los puntos intermedios y final son constantes y no se debe calcular ningún nuevo punto en cada ciclo del robot. Utilizando el mismo, se observó una pequeña mejora en el movimiento del robot. Por otra parte, no es posible utilizarlo en la estrategia potencial ya que los puntos son muy cercanos unos con otros y son calculados en cada iteración (Figura 3.24(a) y Figura 3.24(b)).



(a) Los puntos en la estrategia independiente son calculados en cada iteración y, por consecuente, son cercanos entre sí

(b) Los puntos en la estrategia cooperativa son previamente calculados a distancias mayores ya definidas

Figura 3.24: Metas de algoritmo potencial y cooperativo

Luego que el nodo de coordinación determina los distintos caminos con sus respectivos índices y turnos (*indexTilesList y turnsList*), los caminos son enviados a los nodos **PlanificadorDeCamino** de cada robot.

indexTilesList contiene los índices de los cuadrantes del camino en el orden en que deben ser recorridos, mientras que *turnsList* contiene la cantidad de turnos que debe demorar el robot en ir desde su posición a la siguiente meta.

El robot avanza en el camino según el índice del cuadrante objetivo y el índice del cuadrante actual (obtenidos en *indexTilesList*), siendo las metas las intersecciones entre los cuadrantes (Figura 3.25).

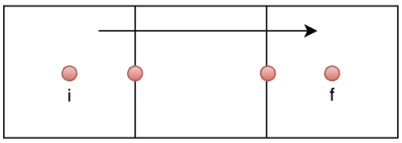
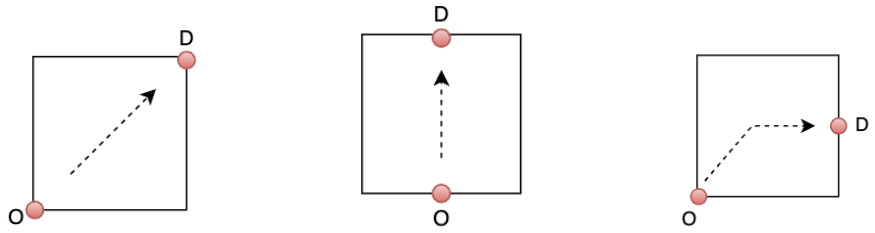


Figura 3.25: Ejemplo de Camino con meta inicial i, metas intermedias y meta final f.

Se utilizan las intersecciones de los cuadrantes con el fin de facilitar el control de en dónde encuentra el robot, y la cantidad de turnos que este debe permanecer en el mismo. Ubicando las metas en las intersecciones podemos saber que al llegar a una meta se da el cambio de cuadrante, dejando libre el anterior y ocupando el nuevo. Además, conociendo la cantidad de turnos que debe permanecer en el nuevo cuadrante, se puede ajustar la velocidad hasta la siguiente meta, en la que sabemos que lo abandonará. De esta forma se asegura que el robot tarda la cantidad de tiempo correspondiente en realizar los trayectos.
La primera meta es la posición central del cuadrante en el que se inicializa el robot. Esto sirve para que independientemente de la posición y rotación con que inicie el robot, llegue al siguiente cuadrante por el trayecto esperado. Para

terminar el camino, la última meta se define en el centro del cuadrante final.

La velocidad del robot durante cada tramo del recorrido está definida por la cantidad de turnos que debe esperar en el mismo y el lugar por el que debe entrar y salir del cuadrante (Figura 3.26). No es lo mismo recorrer la diagonal del cuadrante que cruzarlo verticalmente, ya que en la diagonal se recorre una mayor distancia. En el caso del cuadrante inicial, también se debe tomar en cuenta la posición desde la que inicia el robot, ya que no recorrerá el cuadrante completo, sino que se trasladará desde algún punto hasta el centro, y desde ahí hacia la siguiente meta. Para el último cuadrante, se avanza a velocidad máxima pues el recorrido no afecta a la coordinación.



(a) Robot entra al cuadrante por una diagonal y también se retira por una diagonal

(b) Robot entra al cuadrante por el centro de un lado y también se retira por el centro de un lado

(c) Robot entra al cuadrante por una diagonal y se retira por el centro de un lado

Figura 3.26: Diferentes posiciones por las cuales puede entrar y salir el robot del cuadrante, recorriendo diferentes distancias.

Cuando el sistema detecta que se alcanza una meta, se llama a la función **obtenerNuevoGoal** que retorna la siguiente meta. A este punto se le agrega la distancia del radio del robot dividido 2, para que efectivamente el robot cambie de cuadrante y no genere colisiones al momento de la cooperación (Figura 3.27). Luego este punto se utiliza como argumento en la función **goToGoal** para moverse hasta él.

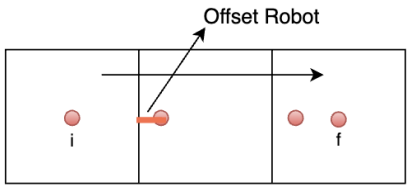


Figura 3.27: Ejemplo de Camino con meta inicial i, metas intermedias y meta final f con offset del tamaño del robot

Bajo ciertas situaciones dentro de la coordinación, es posible que algún robot tenga que volver hacia atrás y luego retomar su camino. Estas situaciones son indicadas por el array **indexTilesList** pues el siguiente elemento puede ser un índice del camino ya recorrido. Una vez que se detecta este escenario, se debe agregar una meta en el centro del cuadrante al que se llegó, para que el robot vaya hasta el centro del cuadrante antes de volver hacia atrás, con el fin de que permanezca en el cuadrante durante la cantidad de turnos apropiada.

Veamos el ejemplo de la Figura 3.28 en el cual un robot tiene que ir del punto inicial I al punto final F y recibe el siguiente array **indexTilesList**: **[0, 1, 2, 1, 2, 3]**. La flecha indica la meta a la cual está yendo el robot en cada iteración.

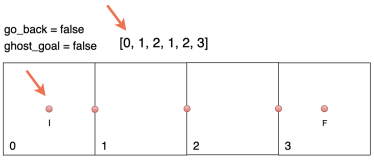
En la Figura 3.28(a), el robot va desde su posición inicial en el cuadrante al centro del mismo. Luego avanza a sus siguientes metas 1 y 2 (Figura 3.28(b) y 3.28(c) respectivamente).

Al llegar a este punto, se detecta que debe retornar al cuadrante anterior (1). Para mantener la coordinación el robot debe permanecer en el cuadrante al que llegó durante una cierta cantidad de turnos. Dado que el punto por el que debe abandonar el cuadrante es el mismo punto por el que arribó, no tendría que recorrer el mismo, por lo que lo pasaría inmediatamente al siguiente. Para prevenir esto, definimos que el robot debe ir primero hasta el centro del cuadrante y luego dar la vuelta. Para lograr esto, se activa la bandera **ghost_goal** y el robot obtiene una "meta fantasma" que se ubica en el centro del cuadrante (Figura 3.28(d)).

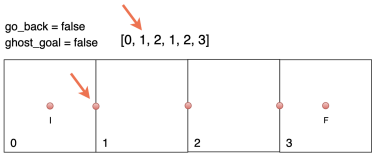
Al llegar a la meta fantasma, como la bandera **ghost_goal** está activada, no aumenta el iterador de la lista de índices del camino, y se asigna la bandera **go_back** con el valor contrario al que tenia. Como estaba desactivada, queda activada. También se desactiva la bandera **ghost_goal** porque llegó a su "meta fantasma". Obtiene como meta la intersección entre el cuadrante anterior y el actual (1 y 2 respectivamente), y como la bandera de **go_back** está activada, el robot va hacia la meta en reversa (Figura 3.28(e)).

Al llegar a su meta (Figura 3.28(f)), el sistema vuelve a identificar que debe generar una "meta fantasma" porque se detecta que la siguiente meta es la intersección entre el cuadrante anterior y el cuadrante actual y se vuelve a activar la bandera **ghost_goal**. Como la bandera **go_back** sigue activada el robot continúa en reversa hacia el centro del cuadrante 1.

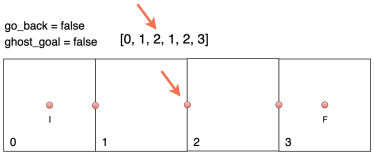
Al llegar, se asignan **go_back** y **ghost_goal** como desactivadas, pues llegó a su meta. Luego se establece como destino la intersección entre el cuadrante 1 y 2 (Figura 3.28(g)).
En las últimas dos figuras (Figura 3.28(h) y 3.28(i)), se observa como el robot llega al último indice de **indexTilesList**, yendo a su cuadrante final y luego al centro del mismo.



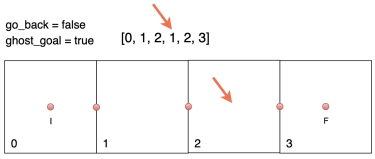
(a) Paso inicial



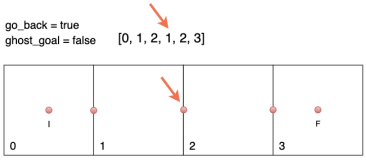
(b) Robot avanza al primer destino



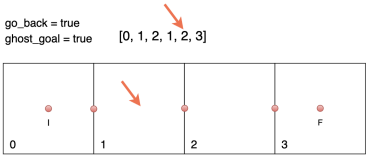
(c) Robot avanza al segundo destino



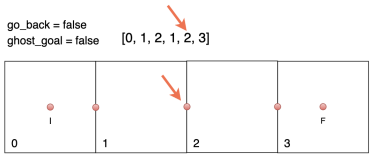
(d) Se detecta que debe retornar al cuadrante anterior, se activa la bandera ghost_goal y se setea como destino el centro del cuadrante



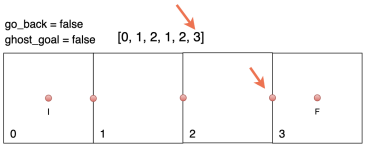
(e) Se activa la bandera go_back y se desactiva la bandera ghost_goal. Se asigna como destino la intersección entre el cuadrante 1 y 2



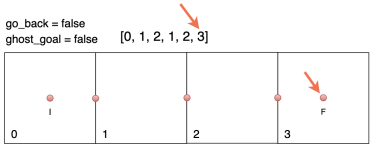
(f) Se detecta que la siguiente meta es la intersección entre el cuadrante anterior y el actual y se activa la bandera ghost_goal. Como la bandera go_back sigue activada el robot navega al centro del cuadrante anterior



(g) Se desactivan las banderas go_back y ghost_goal. Se asigna como destino la intersección entre el cuadrante 1 y 2



(h) Robot avanza al último índice de indexTilesList



(i) Robot avanza al centro del último índice de indexTilesList

Figura 3.28: Ejemplo de camino en el cual el robot debe ir para atrás

En el Pseudocódigo 3.5 se ilustra la subsección Ejecución del Camino. En particular, se observa como se itera por las listas que indican el camino a seguir, como se obtiene la siguiente meta, y la lógica para moverse hasta la misma, respetando la velocidad y dirección en que debe moverse.

Pseudocódigo 3.5: Ejecución del plan para estrategia Cooperativa

```
1     void obtenerNuevaGoal() {
2         if (!goal_fantasma) {
3             iterador++;
4         } else {
5             irParaAtras = !irParaAtras;
6         }
7
8         if (recorriTodosLosIndices) {
9             cuadrante = centroDeUltimaTile();
10        } else {
11            cuadrante =
                  camino[listaIndicesCuadrantes[iterador]];
12        }
13
14        if (!goal_fantasma && detectoCambioSentidoIndices()) {
15            goal_fantasma = true;
16            cuadrante = centroCuadranteActual();
17        } else {
18            goal_fantasma = false;
19        }
20
21        return coordenadas(cuadrante);
22    }
23
24    iterador = -1;
25
26    if (iterador != listaIndicesCuadrantes.size()) {
27        goal = obtenerNuevaGoal();
28        IrAPunto(goal);
29    }
30
31    while (ros::ok()) { // Mientras no se quiera terminar la
          ejecucion
32        punto_actual = obtenerPunto();
33        distancia = distanciaAGoal();
34
35        if (distancia < distancia_tolerada) {
36            if (iterador != listaIndicesCuadrantes.size()) {
37                goal = obtenerNuevaGoal();
38            } else {
39                detenerRobot();
40            }
41        } else {
```

```
42              if (iterador == 0) {
43                  reducidorVelocidad = listaTurnos[iterador];
44              } else {
45                  reducidorVelocidad = listaTurnos[iterador -
                        1];
46              }
47
48              if (iterador != 0 && iterador != 1 && iterador !=
                    listaIndicesCuadrantes.size())
49              {
50                  reducidorVelocidad =
                        calcularVelocidadBasadoEnOrigenYDestino(
51                      cuadranteAnterior,
52                      cuadrenteActual,
53                      cuadranteFuturo,
54                      reducidorVelocidad
55                  )
56              }
57
58              modificarVelocidad(reducidorVelocidad);
59
60              if (irParaAtras) {
61                  irAPuntoParaAtras(goal);
62              } else {
63                  irAPunto(goal);
64              }
65          }
66      }
```

3.2.14. Estrategia Híbrida

La estrategia híbrida surge de unificar las estrategias anteriores para evaluar el desempeño utilizando las ventajas de cada algoritmo: utilizar las fuerzas potenciales para evitar colisiones, y tener un conocimiento del mapa para poder calcular el mejor camino. Por lo tanto, en este algoritmo cada robot calcula el camino más corto a su destino de la misma manera que en la estrategia cooperativa pero en vez de coordinarse entre los robots, cada uno se mueve según lo sensado localmente como en la estrategia potencial.
Tiene como cometido transportarse a través de una secuencia de puntos atractores intermedios para llegar a su destino.
Como consecuencia, el paradigma utilizado es una combinación del paradigma reactivo y jerárquico. El Movimiento y Sensado coincide con lo utilizado para la estrategia de Campos Potenciales (Sección 3.2.2 y 3.2.3).

3.2.15. Arquitectura

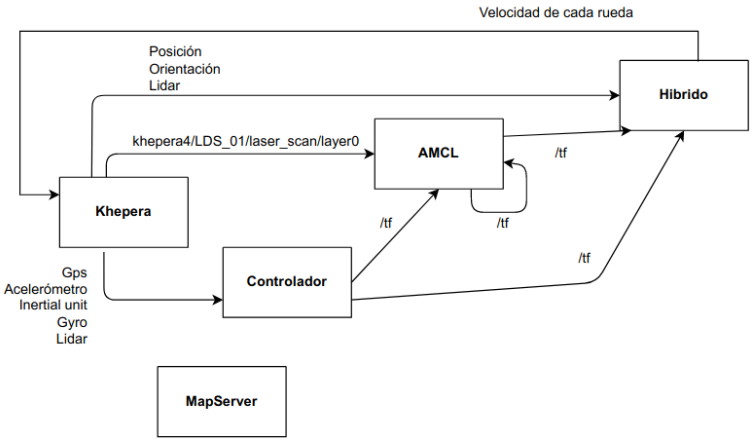
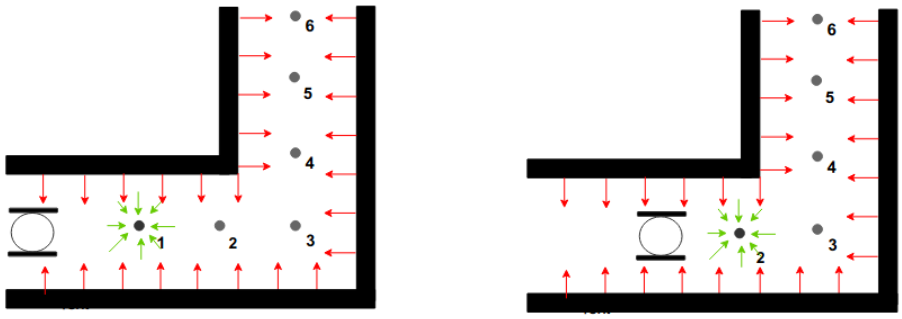


Figura 3.29: Arquitectura Híbrida

Al igual que la estrategia cooperativa, cada robot calcula su camino de manera independiente y el algoritmo cuenta con el nodo AMCL para determinar la posición del robot respecto al mapa ya conocido. Se evalúa el camino y se envía al nodo Híbrido. Este nodo es el encargado de ejecutar el algoritmo A^* para el destino deseado. Una vez obtenidos los puntos intermedios es responsable de obtener la información del sensor y aplicar la estrategia de Campos Potenciales para llegar a cada punto intermedio y al punto final.

3.2.16. Algoritmo

En la Figura 3.30 se muestra un ejemplo del comportamiento del algoritmo. Primero se calculan todos los puntos a recorrer utilizando el algoritmo A*.



(a) Instancia 1 (b) Instancia 2

Figura 3.30: Ejemplo del Algoritmo Híbrido. Se ilustran los puntos a recorrer, punto actual y fuerzas repulsoras y atractivas

Una vez obtenidos los puntos a recorrer, el robot debe ir por cada uno de ellos secuencialmente utilizándolos como puntos atractores.

En la Figura 3.30(a) se muestra el primer movimiento del robot, donde utiliza el punto 1 como punto atractor y se calculan las fuerzas atractivas y repulsivas para que el robot logre llegar al mismo. Una vez que llega a él (Figura 3.30(b)), toma como punto atractor al punto 2 y vuelve a calcular las fuerzas respectivas para alcanzarlo. Así realiza sucesivamente en cada punto hasta llegar al punto destino (punto 6) donde se detiene.

Al utilizar las fuerzas repulsivas, el robot se mantiene alejado de las paredes y logra calcular el movimiento hacia su próximo punto atractor destino.

En este algoritmo se utilizan tolerancias diferentes según el punto atractor al que se debe llegar. Para los puntos atractores intermedios, el margen de llegada es mayor respecto al margen asignado en el último punto atractor. Esto se realiza para poder graduar la flexibilidad en el recorrido, manteniendo la tolerancia deseada al punto final. Si se utiliza una tolerancia grande degenera en el algoritmo reactivo.

El algoritmo híbrido evita las colisiones al igual que la estrategia reactiva. A su vez, a diferencia del algoritmo cooperativo, con el uso de las fuerzas repulsivas se logra evitar posibles choques con otros robots en caso que interfieran en el recorrido. Esto se debe a que los mismos generan fuerzas repulsivas que hacen desviar al robot del recorrido para evitar la colisión.

En el Pseudocódigo 3.6 se presenta el funcionamiento del algoritmo.

Pseudocódigo 3.6: Algoritmo híbrido

```
1 float d_llegada;
```

```
2  float d_llegada_final;
3  float d_vel;
4  bool es_ultimo_atractor;
5
6  while(ros::ok()) { // Mientras no se quiera terminar la
       ejecucion
7      esperarTimestep();
8      info_sensada = sensar();
9      angulo_actual = obtenerAnguloActual();
10     punto_actual = obtenerPuntoActual();
11
12     if (distancia(punto_actual, atractor_actual) < d_llegada)
           {
13         if (es_ultimo_atractor) {
14             detenerRobot();
15             break;
16         } else {
17             atractor_actual = obtenerSiguienteAtractor();
18             if (es_ultimo_atractor) {
19                 d_llegada = d_llegada_final;
20             }
21         }
22     }
23
24     if (es_ultimo_atractor) {
25         if (distancia(punto_actual, punto_final < d_vel)) {
26             bajarVelocidadRobot();
27         } else {
28             subirVelocidadRobot();
29         }
30     }
31     vector<float> siguiente_punto =
           obtenerPunto(angulo_actual, punto_actual,
           atractor_actual, info_sensada)
32     moverseHaciaPunto(siguiente_punto);
33
34 }
```

La función *obtenerPunto* es la misma función utilizada para la estrategia reactiva en la Sección 3.2.6. La función *obtenerSiguienteAtractor* obtiene el siguiente punto en una lista de atractores calculados en base al algoritmo A^*. Si se obtiene el último atractor se asigna *true* en la variable *es_ultimo_atractor*.

Capítulo 4

Experimentación

Contenido

Con el objetivo de probar los algoritmos implementados, se analiza la capacidad de resolución de diversos casos de prueba, los que consisten en que cada robot alcance su meta en un tiempo acotado sin colisionar contra obstáculos u otros robots.

Para cada caso de prueba, se decide repetir la ejecución para obtener métricas más fiables, ya que variables como error del lidar, la actuación del motor y los tiempos de ejecución podrían afectar el resultado de un caso de prueba para dos repeticiones distintas. Al mismo tiempo son de interés las siguientes métricas:

Porcentaje de éxito ($\%e$):

$$\% \frac{\text{repeticiones satisfactorias}}{\text{repeticiones totales}}$$

Las repeticiones satisfactorias corresponden a las repeticiones donde el robot llega al destino sin colisionar y en un tiempo menor al limite especificado.

Porcentaje de casos de choque ($\%c$):

$$\% \frac{\text{repeticiones con choques}}{\text{repeticiones totales}}$$

Tiempo de recorrido promedio (μ_t): Corresponde al tiempo promedio necesario para que los robots lleguen a su destino.

Desviación estándar de tiempo de recorrido (σ_t)

Distancia recorrida promedio (μ_d): Corresponde a la distancia promedio necesaria para que cada robot alcance su destino.

Tiempo total promedio (μ_T): Corresponde al tiempo promedio desde que se ejecuta el comando para el caso de prueba hasta que los robots alcanzan su destino.

Tiempo de procesamiento promedio (μ_{tp}): $\mu_T - \mu_t$. Corresponde al tiempo promedio necesario para inicializar los robots y procesar los cálculos para que los mismos arranquen con sus caminos.

Porcentaje de tiempo de procesamiento promedio($\%\mu_{tp}$): $\% \frac{\mu_{tp}}{\mu_T}$. Corresponde al porcentaje de la proporción de tiempo sobre el tiempo total, en el que los robots están ejecutando sin moverse.

4.1. Automatización de pruebas y obtención de métricas

Para llevar a cabo una automatización de la ejecución de los casos de pruebas y con el fin de recolectar métricas relevantes para analizar el desempeño de los algoritmos se implementaron 2 nodos adicionales.

4.1.1. Nodo Statistics

Este nodo tiene el objetivo de monitorear los robots deseados y sincronizar el comienzo de sus recorridos. Al mismo tiempo recopila el recorrido del robot, detecta colisiones, detecta la llegada al destino, la distancia recorrida y el tiempo que demora en llegar al destino.
Para la detección de colisiones se optó por recubrir los robots con un cuerpo de forma cilíndrica para detectar el tacto como se muestra en la siguiente Figura 4.1.

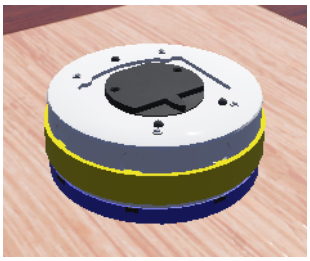


Figura 4.1: Robot Khepera con sensor de tacto

4.1.2. Nodo Tests

Este nodo tiene como cometido ejecutar un conjunto de casos de prueba descritos en un archivo *yaml*. El funcionamiento del nodo consiste en ejecutar Webots, los archivos *.launch* correspondientes a la estrategia y comunicarse con el nodo Statistics para sincronizar los robots y luego obtener las métricas. Este proceso ejecuta las repeticiones deseadas. Una vez ejecutados los casos de pruebas se escriben los resultados en un archivo de texto.

4.2. Casos de prueba

Cada caso de prueba se ejecutó 10 veces en una máquina con las características detalladas en el Cuadro 4.1.

OS	Ubuntu 20.04
CPU	Intel Pentium G4560 3.50GHz
Memoria	16GiB
Versión de ROS	noetic
Versión de Webots	2021a

Cuadro 4.1: Entorno de experimentación

Cada caso de prueba se ejecuta hasta que los robots llegan a destino o se cumple el máximo de tiempo especificado.

Se consideran distintos escenarios evaluando los siguientes criterios:

- Llegar al destino en un tiempo limitado.
- Evitar colisiones con otros robots.
- Evitar colisiones con paredes.
- Encontrar el camino más corto.
- Capacidad de evitar que los robots se interpongan entre sí.
- Resolución del problema en un entorno real.
- Evitar aglomeraciones.

Para evaluar el desempeño de los algoritmos con componente potencial, en cada escenario específico se decidió probar manualmente y encontrar intuitivamente un conjunto de variables que hagan resolver en lo posible el problema. Estas variables son seleccionadas principalmente para lograr que los robots recorran las paredes, utilicen una fuerza repulsiva mayor o para sensar objetos con más alcance.

Para la ejecución de los escenarios del algoritmo cooperativo se utiliza una grilla de 0.5m, excepto en los casos en que se quiere comparar con cuadrantes de menor tamaño, los cuales se explicitará directamente.

Si bien la velocidad lineal máxima del robot Khepera IV es de 1 m/s, se utiliza el 30 % de la misma para las pruebas. Esto es porque a mayor velocidad se dificulta la maniobrabilidad, y además aumenta el margen de tiempo y espacio que requiere esquivar un obstáculo.

A continuación se muestran los casos de prueba realizados, donde cada robot se identifica en la figura con un cuadrado de determinado color. Se presentan dos figuras donde la figura izquierda simboliza la configuración inicial, y la figura de la derecha simboliza la configuración final deseada.
En los primeros casos de prueba se utilizan escenarios con un único robot para evaluar la resolución de problemas particulares como son los mínimos locales, pasillos estrechos, y la toma de un camino mas corto hacia el punto destino. Luego se evalúan escenarios con varios robots a la vez, donde cada caso de prueba presenta alguna dificultad particular para las estrategias.

4.2.1. Caso 01 - Mínimo local

(a) Configuración inicial (b) Configuración final

Figura 4.2: Robot atraviesa un mínimo local (marcado con óvalo rojo)

Este caso evidencia la capacidad de la estrategia reactiva para escapar de un mínimo local (óvalo rojo en 4.2(a)) utilizando las fuerzas tangenciales. La estrategia híbrida y cooperativa resuelven el problema satisfactoriamente ya que calculan el camino hacia la meta.

Métrica	**Reactiva**	**Híbrida**	**Cooperativa**
$\%e$	100	100	100
$\%c$	0	0	0
μ_d (m)	8.5	6.1	5.8
μ_t (s)	31.0	21.2	20.8
σ_t	7.73	0.40	0.40
$\%\mu_{tp}$	24	48	53

Cuadro 4.2: Métricas caso de prueba 1

4.2.2. Caso 02 - Mínimo local con pasillos

(a) Configuración inicial (b) Configuración final

Figura 4.3: Robot atraviesa un mínimo local (marcado con círculo rojo) en una estructura con pasillos

La estrategia reactiva falla en llegar a destino por quedarse atrapado en un mínimo local (circulo rojo en 4.3(a)). Incluso en los casos donde logra escapar de éste, le es imposible ingresar al pasillo final debido a los parámetros seleccionados, ya que las paredes generan fuerzas repulsivas que impiden el ingreso del robot al mismo. En cambio, la estrategia híbrida al conocer el camino a realizar puede resolver el escenario satisfactoriamente al igual que la cooperativa.
Si bien en el caso de prueba 01 se logran obtener parámetros que resuelven el problema de mínimos locales, cuando los mismos se combinan con pasillos estrechos es difícil obtener parámetros que resuelvan el escenario en conjunto.

Métrica	**Reactiva**	**Híbrida**	**Cooperativa**
$\%e$	0	100	100
$\%c$	0	0	0
μ_d (m)	-	5.0	5.1
μ_t (s)	-	17.8	19.2
σ_t	-	0.40	0.39
$\%\mu_{tp}$	-	46	52

Cuadro 4.3: Métricas caso de prueba 2

4.2.3. Caso 03 - Camino más corto

(a) Configuración inicial

(b) Configuración final

Figura 4.4: Robot decide el camino más corto

La estrategia potencial tiene un porcentaje de éxito del 20 % pues en algunas ocasiones las mismas fuerzas que son necesarias para que el robot pueda salir de mínimos locales (óvalos rojos en 4.4(a)) lo hacen colisionar. Sin embargo cuando lo logra toma el camino más largo, lo que evidencia la incapacidad de encontrar el mejor camino. La estrategia híbrida (similar al caso 02) puede resolver de manera muy similar a la estrategia cooperativa.

Métrica	Reactiva	Híbrida	Cooperativa
$\%e$	20	100	100
$\%c$	70	0	0
μ_d (m)	27.2	10.3	10.4
μ_t (s)	95.0	35.0	36.6
σ_t	3.00	0.10	0.49
$\%\mu_{tp}$	13	31	37

Cuadro 4.4: Métricas caso de prueba 3

4.2.4. Caso 04 - Pasillo estrecho

(a) Configuración inicial

(b) Configuración final

Figura 4.5: Robot cambia de zona al pasar por un pasillo

La única estrategia que llega a destino sin colisionar en todas las repeticiones es la cooperativa. Esto se debe a que en las estrategias con componente reactivo los robots llegan al pasillo en el mismo momento y quedan repeliéndose entre ellos, sin poder avanzar en el pasillo. Sin embargo la estrategia reactiva logra en 3 ocasiones resolver el problema. En dichas instancias hay dos opciones posibles, la primera es que un robot logre sacar a otro del pasillo, y la segunda es que un robot demore al entrar al pasillo. Esto no ocurrió en la estrategia híbrida, lo cual podría ser por tener caminos menos flexibles a realizar o por los parámetros seleccionados.

Métrica	Reactiva	Híbrida	Cooperativa
$\%e$	30	0	100
$\%c$	70	70	0
μ_d (m)	14.0	-	2.7
μ_t (s)	51.8	-	12.8
σ_t	28.0	-	0.46
$\%\mu_{tp}$	35	-	69

Cuadro 4.5: Métricas caso de prueba 4

4.2.5. Caso 05 - Intercambio de posiciones

(a) Configuración inicial

(b) Configuración final

Figura 4.6: Robots deben ir a la posición del robot enfrente suyo

En este caso de prueba los robots deben ir a la posición inicial del robot que se encuentra enfrente suyo, por lo tanto los caminos óptimos entre los robots que se enfrentan son los mismos. De este modo la estrategia cooperativa falla, ya que estos caminos están contenidos entre sí. El conocimiento de cuadrantes libres son los que están dentro de su camino y no es posible saber si los cuadrantes contiguos serán o no serán utilizados por otros robots. Esto imposibilita la resolución [29].

Las estrategias reactiva e híbrida resuelven el 100 % de los casos. En la estrategia reactiva, los robots al llegar al centro del mapa tienen la capacidad de repelerse entre sí y rotar al mismo tiempo para luego continuar al punto final. Para el algoritmo híbrido es más difícil ya que los robots deben pasar por un punto cercano al centro del mapa. De esta manera se evidencia que el algoritmo reactivo tiene una resolución más rápida y eficiente.

Métrica	Reactiva	Híbrida	Cooperativa
$\%e$	100	100	0
$\%c$	0	0	0
μ_d (m)	6.1	9.3	-
μ_t (s)	23.3	33.0	-
σ_t	8.5	14	-
$\%\mu_{tp}$	46	48	-

Cuadro 4.6: Métricas caso de prueba 5

4.2.6. Caso 06 - Cruce entre robots

(a) Configuración inicial

(b) Configuración final

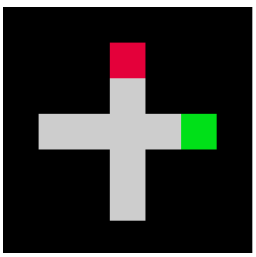
Figura 4.7: Robots deben cruzarse entre ellos

Este caso de prueba es similar al anterior ya que los robots deben cruzarse entre ellos, pero los caminos generados no están contenidos entre si. Los algoritmos reactivo y cooperativo resuelven el escenario en el 100 % de los casos. El algoritmo híbrido tiene un porcentaje de éxito del 90 %, esto es debido a un choque entre robots.

Métrica	Reactiva	Híbrida	Cooperativa
$\%e$	100	90	100
$\%c$	0	10	0
μ_d (m)	3.2	2.6	1.7
μ_t (s)	13.1	9.8	7.4
σ_t	1.4	0.63	0.61
$\%\mu_{tp}$	62	72	80

Cuadro 4.7: Métricas caso de prueba 6

4.2.7. Caso 07 - Camino estrecho sin colisiones



(a) Configuración inicial

(b) Configuración final

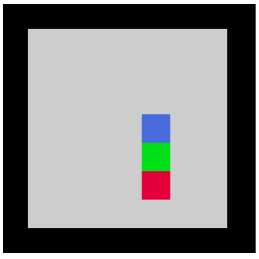
Figura 4.8: Robots deben ir por un camino estrecho sin colisionar

La dificultad de este caso radica en evitar una colisión en la intersección de 2 pasillos angostos, los cuales dificultan la maniobrabilidad. La estrategia reactiva tiene un 80 % de éxito presentando choques en los casos sin éxito. La estrategia híbrida obtiene un 100 % de casos de éxito al igual que la estrategia cooperativa.

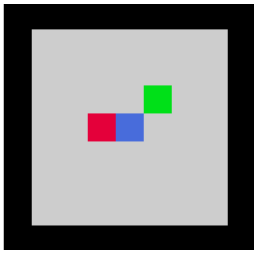
Métrica	**Reactiva**	**Híbrida**	**Cooperativa**
$\%e$	80	100	100
$\%c$	20	0	0
μ_d (m)	2.9	2.8	1.9
μ_t (s)	12.7	10.5	8.4
σ_t	0.70	0.67	0.37
$\%\mu_{tp}$	51	64	75

Cuadro 4.8: Métricas caso de prueba 7

4.2.8. Caso 08 - Dejar paso a otros robots



(a) Configuración inicial



(b) Configuración final

Figura 4.9: Robots en cuadrantes contiguos deben dejar pasar a otros

Este es un caso específico utilizado para validar la marcha hacia atrás en los caminos de algunos robots en la estrategia cooperativa. En este caso, el robot azul deberá dejar pasar al robot verde para que el mismo llegue a su posición, luego vuelve a su destino dejándole el camino libre al robot rojo para que tome la diagonal, para al final de todo ir a su meta.
Todas las estrategias resuelven este caso el 100 % de las repeticiones con tiempos similares.

Métrica	**Reactiva**	**Híbrida**	**Cooperativa**
$\%e$	100	100	100
$\%c$	0	0	0
μ_d (m)	0.9	0.9	0.8
μ_t (s)	4.9	3.9	4.3
σ_t	0.30	0.37	0.13
$\%\mu_{tp}$	76	84	87

Cuadro 4.9: Métricas caso de prueba 8

4.2.9. Caso 09 - Dejar paso a otros robots 2

(a) Configuración inicial

(b) Configuración final

Figura 4.10: Robots en cuadrantes contiguos deben dejar pasar a otros con obstáculos

Caso análogo al Caso 08 en cuanto a las dificultades para la estrategia cooperativa.
Las estrategias híbrida y reactiva poseen un éxito del 100 %.
En la estrategia cooperativa utilizando cuadrantes de 0.25m los robots no llegan nunca a destino, pues el robot rojo no puede realizar el movimiento diagonal ya que pasa al lado del robot azul, y nunca podrá llegar a destino.

Métrica	Reactiva	Híbrida	Cooperativa
$\%e$	100	100	100
$\%c$	0	0	0
μ_d (m)	0.9	0.9	0.9
μ_t (s)	4.8	3.6	4.5
σ_t	0.20	0.39	0.45
$\%\mu_{tp}$	77	86	87

Cuadro 4.10: Métricas caso de prueba 9

4.2.10. Caso 10 - Caminos incluidos entre robots

(a) Configuración inicial

(b) Configuración final

Figura 4.11: Un camino contenido en otro

Para los casos con componente potencial este caso tiene la dificultad que los robots se trasladan en direcciones opuestas, por lo que se verifica la repulsión para evitar choques y que se interpongan entre sí. La estrategia reactiva presenta un porcentaje de éxito del 100 % y la híbrida del 70 %, teniendo choque en un caso. La dificultad para la estrategia híbrida radica en que el robot verde debe seguir un camino, y parte del camino queda obstaculizado por el robot rojo, ya que éste llega y se detiene en su destino. Para la estrategia cooperativa no se puede resolver, ya que un camino contiene a otro.

Métrica	Reactiva	Híbrida	Cooperativa
$\%e$	100	70	0
$\%c$	0	10	0
μ_d (m)	2.3	3.2	-
μ_t (s)	9.9	11.9	-
σ_t	0.50	4.80	-
$\%\mu_{tp}$	61	67	-

Cuadro 4.11: Métricas caso de prueba 10

4.2.11. Caso 11 - Cruce diagonal de caminos

(a) Configuración inicial (b) Configuración final

Figura 4.12: Cruce de caminos en diagonal

La complejidad de este caso radica en poder realizar un cruce en diagonal sin colisionar. Este caso es resuelto el 100 % de las repeticiones para las 3 estrategias.

Métrica	**Reactiva**	**Híbrida**	**Cooperativa**
$\%e$	100	100	100
$\%c$	0	0	0
μ_d (m)	3.0	2.8	1.9
μ_t (s)	12.2	10.6	7.4
σ_t	1.40	0.44	0.27
$\%\mu_{tp}$	52	65	77

Cuadro 4.12: Métricas caso de prueba 11

4.2.12. Caso 12 - Cruce diagonal de caminos 2

(a) Configuración inicial (b) Configuración final

Figura 4.13: Cruce diagonal en el final

Este caso presenta una dificultad similar al caso 11, sin embargo finalizan el recorrido inmediatamente luego del cruce. Las estrategias reactiva e híbrida tienen un 100 % de éxito.
La estrategia cooperativa no resuelve este escenario, debido a que el punto final

está marcado como conflicto por ser un cruce de diagonales y al ejecutar el algoritmo A^* con el grupo de colisiones no se obtiene un camino.

Métrica	**Reactiva**	**Híbrida**	**Cooperativa**
$\%e$	100	100	0
$\%c$	0	0	0
μ_d (m)	2.9	2.4	-
μ_t (s)	15.1	9.4	-
σ_t	6.00	1.00	-
$\%\mu_{tp}$	48	66	-

Cuadro 4.13: Métricas caso de prueba 12

4.2.13. Caso 13 - Dejar paso en cuadrantes contiguos

(a) Configuración inicial (b) Configuración final

Figura 4.14: Robot debe dejar pasar a otro en cuadrantes contiguos

Las 3 estrategias resuelven el escenario el 100 % de los casos con éxito.
Cabe destacar que la única forma en que la estrategia cooperativa encuentre una solución es si se permite que todos los robots avancen al mismo tiempo, ya que para los 3 robots el siguiente cuadrante está ocupado. Por esto, solo se encuentra la solución si se permite que los robots avancen mientras el robot que estaba en el cuadrante objetivo se retira del mismo. En este caso, como los cuadrantes miden 0.5 metros y el robot 0.13m de diámetro, hay espacio suficiente para que no ocurran colisiones si se habilita esta opción. Es por este mismo motivo que la estrategia cooperativa siempre llega a destino.

Por otro lado, si usáramos cuadrantes de 0.25m, como ya no existe espacio para que 2 robots estén al mismo tiempo en un cuadrante, el problema no tendría solución.

Métrica	Reactiva	Híbrida	Cooperativa
$\%e$	100	100	100
$\%c$	0	0	0
μ_d (m)	0.6	0.6	0.4
μ_t (s)	3.8	2.8	2.7
σ_t	0.30	0.43	0.38
$\%\mu_{tp}$	78	88	90

Cuadro 4.14: Métricas caso de prueba 13

4.2.14. Caso 14 - Aglomeración de pasillo

(a) Configuración inicial

(b) Configuración final

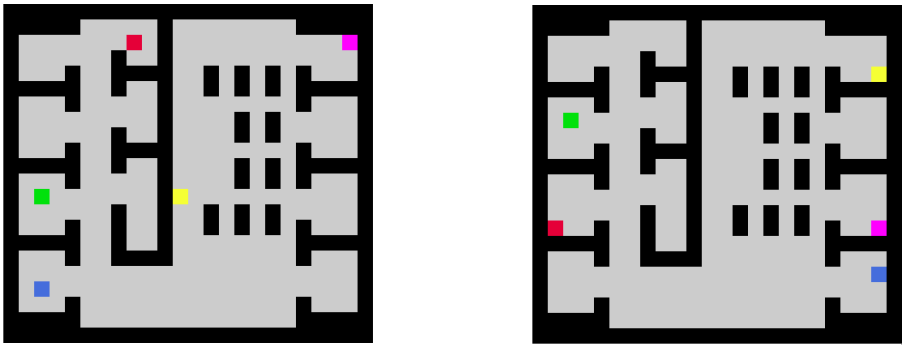
Figura 4.15: Aglomeración de pasillo

La dificultad de este caso radica en la generación de una aglomeración, en donde los robots deben atravesar un pasillo estrecho y los 3 están a una distancia muy similar. Las estrategias reactiva e híbrida tienen un 60 % de casos de éxito. La estrategia reactiva presenta choques en los casos de falla y la estrategia híbrida presenta 2 casos de choque y 2 casos en donde los robots quedan trancados entre sí. Esto ocurre porque los robots compiten por pasar primero. La estrategia cooperativa resuelve el problema el 100 % de las repeticiones.

Métrica	Reactiva	Híbrida	Cooperativa
$\%e$	60	60	100
$\%c$	40	20	0
μ_d (m)	7.2	4.0	3.6
μ_t (s)	26.8	14.2	14.3
σ_t	3.11	0.37	0.43
$\%\mu_{tp}$	56	63	68

Cuadro 4.15: Métricas caso de prueba 14

4.2.15. Caso 15 - Oficina



(a) Configuración inicial (b) Configuración final

Figura 4.16: Caso de prueba realista

Este caso de prueba es un entorno más realista que los anteriores, donde hay varias habitaciones. Se dispone de 5 robots existiendo 2 posibles cruces entre 2 pares de robots. Las estrategias reactiva e híbrida obtuvieron un porcentaje de éxito de 0 % y 10 % respectivamente. El problema yace en la presencia de un problema similar al presentado en el caso de prueba 1, donde 2 robots quedan atrapados en un pasillo.
La estrategia cooperativa fue capaz de resolver el caso el 100 % de las repeticiones.

Métrica	**Reactiva**	**Híbrida**	**Cooperativa**
$\%e$	0	10	100
$\%c$	70	40	0
μ_d (m)	-	8.2	8.0
μ_t (s)	-	28.2	28.3
σ_t	-	-	0.23
$\%\mu_{tp}$	-	63	64

Cuadro 4.16: Métricas caso de prueba 15

4.3. Análisis de resultados

En los cuadros 4.17 y 4.18 se detallan los resultados obtenidos de la experimentación.
Cuando la suma de $\%e$ y de $\%c$ no es del 100 %, se debe a los casos en que el robot no colisionó pero no llegó al punto destino.

Caso	Reactiva			Híbrida			Cooperativa		
	$\%e$	$\%c$	μ_d (m)	$\%e$	$\%c$	μ_d (m)	$\%e$	$\%c$	μ_d (m)
1	100	0	8.5	100	0	6.1	100	0	5.8
2	0	0	-	100	0	5.0	100	0	5.1
3	20	70	27.2	100	0	10.3	100	0	10.4
4	30	70	14.0	0	70	-	100	0	2.7
5	100	0	6.1	100	0	9.3	0	0	-
6	100	0	3.2	90	10	2.6	100	0	1.7
7	80	20	2.9	100	0	2.8	100	0	1.9
8	100	0	0.9	100	0	0.9	100	0	0.8
9	100	0	0.9	100	0	0.9	100	0	0.9
10	100	0	2.3	70	10	3.2	0	0	-
11	100	0	3.0	100	0	2.8	100	0	1.9
12	100	0	2.9	100	0	2.4	0	0	-
13	100	0	0.6	100	0	0.6	100	0	0.4
14	60	40	7.2	60	20	4.0	100	0	3.6
15	0	70	-	10	40	8.2	100	0	8.0

Cuadro 4.17: Porcentaje de éxito, porcentaje de casos con choques, distancia recorrida promedio

Caso	Reactiva			Híbrida			Cooperativa		
	μ_t (s)	σ_t	$\%\mu_{tp}$	μ_t (s)	σ_t	$\%\mu_{tp}$	μ_t (s)	σ_t	$\%\mu_{tp}$
1	31.0	7.73	24	21.2	0.40	48	20.8	0.40	53
2	-	-	-	17.8	0.40	46	19.2	0.39	52
3	95.0	3.00	13	35.0	0.10	31	36.6	0.49	37
4	51.8	28.0	35	-	-	-	12.8	0.46	69
5	23.3	8.5	46	33.0	14	48	-		-
6	13.1	1.4	62	9.8	0.63	72	7.4	0.61	80
7	12.7	0.70	51	10.5	0.67	64	8.4	0.37	75
8	4.9	0.30	76	3.9	0.37	84	4.3	0.13	87
9	4.8	0.20	77	3.6	0.39	86	4.5	0.45	87
10	9.9	0.50	61	11.9	4.80	67	-	-	-
11	12.2	1.40	52	10.6	0.44	65	7.4	0.27	77
12	15.1	6.00	48	9.4	1.00	66	-	-	-
13	3.8	0.30	78	2.8	0.43	88	2.7	0.38	90
14	26.8	3.11	56	14.2	0.37	63	14.3	0.43	68
15	-	-	-	28.2	-	63	28.3	0.23	64

Cuadro 4.18: Promedio de tiempo de llegada, desviación estándar de tiempo de llegada y porcentaje de tiempo de procesamiento

4.3.1. Reactiva

De los resultados obtenidos, podemos observar que la estrategia reactiva no logra resolver todos los escenarios, y además en varios casos colisiona. Entre los puntos débiles de esta estrategia se destacan:

Pasillos estrechos: Las paredes de los pasillos generan fuerzas repulsivas hacia los costados de los robots, haciendo que los mismos se muevan ineficientemente. Además, a pesar de que un pasillo tenga la medida para que 2 robots puedan transitarlo al mismo tiempo, en la práctica no fue posible lograrlo debido a las fuerzas repulsivas generadas entre los robots y las paredes.

Necesidad de coordinación: Dado que el algoritmo no planifica, los robots no son capaces de ceder paso entre sí, por lo que varios mapas con sectores estrechos no se pueden resolver.

Mínimos locales: A pesar de haber implementado una fuerza tangencial con el fin de mitigar este problema, cuando un mismo mapa contiene además de mínimos locales pasillos estrechos es difícil encontrar parámetros que puedan resolver el escenario.

Eficiencia de trayectoria: El algoritmo no es capaz de encontrar los caminos más cortos.

Aglomeraciones: El algoritmo no es capaz de evitar las aglomeraciones de robots, siendo vulnerable a que estos queden trancados entre sí.

Robots posicionados de manera simétrica: Utilizando sensores sin errores significativos, los escenarios donde los robots se deben cruzar simétricamente son difíciles de resolver, ya que el problema tiende a hacer que los robots se tranquen entre sí.

4.3.2. Cooperativa

Comparando con la estrategia reactiva, se observa que la estrategia cooperativa obtiene un mejor tiempo de llegada y menor distancia recorrida en todos los casos de prueba. La desviación estándar también es significativamente más pequeña, por lo que se deduce que las soluciones obtenidas son más parecidas entre sí. Por otro lado, esta estrategia presenta un tiempo de procesamiento 38 % mayor.
La debilidad más importante de esta estrategia es la incapacidad de resolver escenarios donde algún camino planificado esté contenido en otro. En el resto de los casos de prueba, este algoritmo no obtiene casos de choque.

4.3.3. Híbrida

Comparando con la estrategia reactiva, se observa una mejor capacidad de resolución de mínimos locales, incluso cuando existen pasillos al mismo tiempo.

Además esta estrategia tiene la capacidad de que los robots encuentren el camino más corto a su destino. También en la mayoría de los escenarios obtiene soluciones con menor desviación estándar. Por otro lado, se evidencia una desmejora en algunos casos donde los robots se aglomeran, ya que los caminos tienen menor flexibilidad respecto a la estrategia reactiva. Al igual que en la estrategia reactiva, surge la necesidad de asignar prioridades o tener algún tipo de coordinación para resolver problemas donde los robots deben ceder paso. El tiempo de procesamiento es en promedio 27 % mayor respecto a la estrategia reactiva.

Comparando con la estrategia cooperativa, se observa que el mejor tiempo de resolución depende del caso de prueba específico. Con respecto a la distancia recorrida, se obtienen peores resultados para la estrategia híbrida. La desviación estándar es generalmente menor en la estrategia cooperativa, por lo que se deduce que las soluciones obtenidas son más parecidas entre sí. En cuanto al tiempo de procesamiento, la estrategia cooperativa en promedio es 8 % mayor.

Capítulo 5

Conclusiones y trabajo futuro

Contenido

En esta sección se presentan las conclusiones a las que se llegaron respecto a las estrategias presentadas, ventajas y desventajas de las mismas y casos a los que podrían ser aplicables.
También se presentan ideas de como proseguir en caso de buscarse mejorar estas estrategias.

5.1. Conclusiones

Se implementaron y se realizaron pruebas a 3 estrategias utilizando distintos paradigmas. En base a los análisis realizados no podemos concluir que una estrategia sea mejor que otra, pero sí podemos afirmar que algunas estrategias son más adecuadas según lo que se espera del comportamiento de los robots, las restricciones del problema y características del entorno. En el Cuadro 5.1 se describen algunas características y restricciones de las estrategias implementadas.

Descripción	**Reactiva**	**Híbrida**	**Cooperativa**
Actúa sin conocer el entorno	Si	No	No
Evita obstáculos no conocidos	Si	Si	No
Resuelve problemas de coordinación	No	No	Si
Robusto a entornos complejos	No	No	Si
Lógica completamente distribuida	Si	Si	No
Optimalidad en distancia recorrida	No	No	No
Completitud	No	No	No

Cuadro 5.1: Características de las estrategias

5.1.1. Reactiva

Esta estrategia de rápida ejecución, se puede implementar sin utilizar lógica compleja y es capaz de resolver una cantidad de escenarios con distintos desafíos, sin embargo la simpleza de esta estrategia trae como consecuencia la dificultad de encontrar un conjunto único de parámetros que trasladen a los robots a sus respectivos destinos de forma satisfactoria en distintos tipos de escenarios, en particular, las cualidades de evitar colisiones, salir de mínimos locales y pasar por pasillos estrechos son muy difíciles de equilibrar. Por ello no es de mucha utilidad en entornos donde existan espacios reducidos. Por otro lado, esta estrategia tiene la ventaja de ser aplicable sin conocer el entorno, por lo tanto los robots pueden evitar obstáculos dinámicos y desconocidos. Al mismo tiempo esta estrategia siempre posee el mismo estado, por lo que los robots podrían apagarse y prenderse o cambiarse de lugar sin necesidad de reconfigurar el robot.

5.1.2. Cooperativo

Esta estrategia tiene la restricción de conocer el entorno donde se ejecuta. En los casos donde los robots no deben moverse hacia atrás, se encuentra la solución óptima dentro de la cuadrícula, pero no la solución óptima del camino. Esto es debido a que la cuadrícula es una abstracción del mundo y es un espacio discreto en lugar de un espacio continuo. Es aplicable a entornos con bajo espacio de maniobrabilidad ya que se resuelven problemas de cooperación entre robots.
Sin embargo, esta estrategia no resuelve escenarios en donde haya obstáculos no conocidos. Además esta estrategia es vulnerable a la desincronización de los robots y necesita un nodo maestro que gestione la cooperación.

5.1.3. Híbrido

Esta estrategia tiene como restricción conocer el entorno donde se aplica el algoritmo, sin embargo a diferencia de la estrategia cooperativa, tiene la capacidad de reaccionar y evitar colisiones ante obstáculos no conocidos. Al igual que la estrategia reactiva, no es aplicable en entornos donde existen espacios reducidos. En cuanto a la calidad de soluciones, es capaz de encontrar soluciones cercanas a la óptima cuando no se presentan problemas de coordinación entre robots.

5.2. Trabajo Futuro

En esta sección se proponen ideas para continuar mejorando las estrategias presentadas.

5.2.1. Estrategia Reactiva

Recuperación de mínimos locales
: Existen escenarios donde el robot no puede resolver el problema y queda atrapado en un mínimo local, esto se debe a que las fuerzas tangenciales no son suficientes para escapar del mismo. Una mejora a realizar es poder detectar cuando el robot está atrapado en un mínimo local y accionar un comportamiento para escapar del mismo.

5.2.2. Estrategias Reactiva e Híbrida

Velocidad variable
: La velocidad lineal utilizada es constante. Una posible mejora es que el robot varíe su velocidad según el entorno. Por ejemplo, el algoritmo podría ser más robusto y evitar choques si el robot disminuyera la velocidad al detectar objetos cercanos.

Auto ajuste de parámetros
: Dado que los escenarios requieren distintos parámetros para encontrar soluciones, sería interesante que los robots tengan la capacidad de ajustar

los parámetros durante el recorrido para adaptarse a la dificultad del sector del mapa.

Asignación de parámetros por robot en *script* de pruebas

En la automatización de pruebas se creó un diseño donde se asignan las mismas variables de ejecución a todos los robots de un mismo caso de prueba. La posibilidad de asignar distintas variables a cada robot, por ejemplo distintas fuerzas repulsivas, también podría ayudar a solucionar problemas de coordinación, logrando crear una prioridad según dicha fuerza entre los robots (el robot con mayor fuerza repulsiva, tendría menor prioridad).

5.2.3. Estrategia Cooperativa

Velocidad Máxima

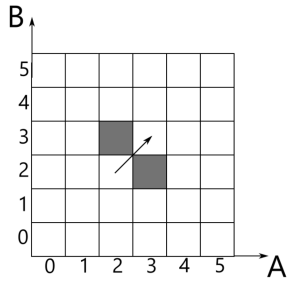
Una mejora del algoritmo es permitir que los robots que no tengan conflictos se muevan a velocidad máxima. En la implementación presentada, cuando un robot no detecta posibles conflictos continúa restringiendo la velocidad según la posición del robot del cuadrante al que va a ir y el cuadrante actual, por ejemplo, cuando un robot cruza un cuadrante de lado a lado o de punta a punta.

Esto no es necesario, debido a que el tiempo que tarda en cruzar los cuadrantes no es relevante ya que no se requiere coordinar con ningún otro robot. En su lugar, se podría hacer que estos robots se muevan siempre a velocidad máxima, para disminuir el tiempo que tardan en llegar a la meta.

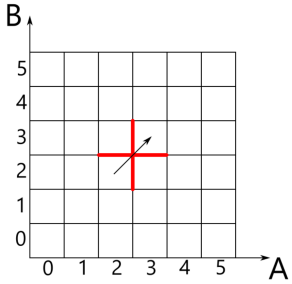
Resolución de casos conflictivos especiales

Como se mencionó en la sección 3.2.12, la estrategia elegida para prevenir colisiones en los casos de "intercambio de posiciones" y "cruce de diagonales" genera una limitación en el espacio de soluciones, ya que debido a un conflicto en uno de los movimientos con los que se puede llegar a una configuración, se marca esta como conflictiva, aunque no lo sea en si misma, causando que se descarten posibles soluciones que pasen por dicha configuración, pudiendo llegar a la misma de otras formas no conflictivas. Un ejemplo de este error ocurre en el Caso de Prueba 11 - Cruce diagonal 2, en el cual no se puede encontrar solución ya que la configuración final se marca como conflicto.

Una alternativa es que al permitirse los movimientos en diagonal que pasan por el vértice de un obstáculo en el diagrama de coordinación, se prohíban aquellos movimientos diagonales que pasen por un vértice tal que se encuentren 2 obstáculos de borde, o un obstáculo completo a ambos lados del camino diagonal, siempre y cuando los obstáculos sean proyectados desde un mismo plano, es decir, generados por un mismo par de robots. En la figura 5.1 se muestran ejemplos de los movimientos no permitidos.



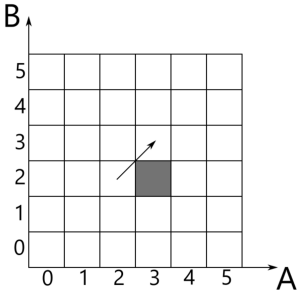
(a) Intercambio de posiciones



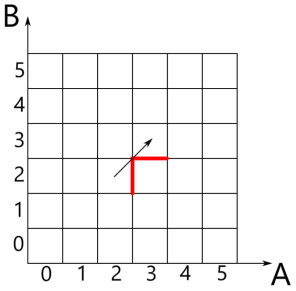
(b) Cruce de diagonales

Figura 5.1: Ejemplos de movimientos no permitidos.

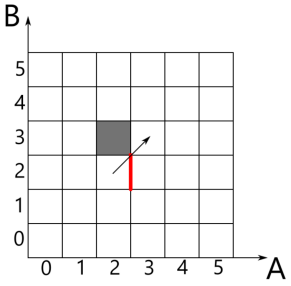
Sí se deben permitir en cambio los casos en que se encuentra un obstáculo completo a un lado y solo un obstáculo de borde en el otro. En la Figura 5.2 se muestran ejemplos de movimientos que estarían permitidos.



(a) Un obstáculo completo



(b) Obstáculos de borde a un solo lado



(c) Un obstáculo completo y un obstáculo de borde

Figura 5.2: Ejemplos de movimientos permitidos.

Prioridades

Implementar la posibilidad de asignar distintas prioridades a los robots. Una forma de lograr esto sería modificando el algoritmo utilizado para realizar la búsqueda del camino en el diagrama de coordinación de forma que priorice avanzar respecto a un eje por sobre los otros, por ejemplo, modificando el cálculo de distancia a la meta de los puntos intermedios del grafo.

Inicio de Robots

La estrategia requiere que todos los robots se inicien simultáneamente, para que lleven a cabo la coordinación antes de empezar a moverse. Esto no es una opción práctica si se quisiera utilizar en un ambiente real, ya que se podrían querer agregar nuevos robots mientras los otros se mueven, o un robot querer moverse a una nueva meta luego de llegar a su propuesta inicial, mientras los otros robots aún se están moviendo.
Para hacer esto se tendrían que frenar los robots que se encuentran en movimiento para volver a iniciar el proceso de coordinación, o que los mismos avancen hasta su meta actual y luego detenerlos. Otra opción es tratar a los robots en movimiento como obstáculos, de tal forma que los robots que comienzan a moverse eviten los caminos de los mismos para planificar sus caminos.

Subconjunto de Caminos

En el caso en que el camino de un robot B esté totalmente incluido en el camino del robot A, el robot A no podrá llegar nunca a su destino ya que no tiene posibles celdas para trasladarse, pues comparten todas las celdas con el robot B (Figura 5.3).

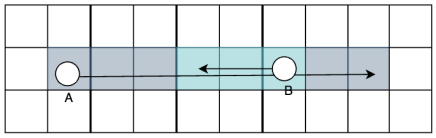


Figura 5.3: El camino del Robot B es un subconjunto del camino del Robot A

La solución a este problema sería tener en cuenta distintas celdas a las del camino inicial, en particular las celdas adyacentes, para poder así esquivar al robot A y llegar a destino.

Creación de Mapas

Para la generación de mapas se utilizó Gmapping, con un error conocido que es el desfasaje de 1 cuadrante en los obstáculos, el cual se solucionó modificando la imagen de grises manualmente. Las posibles soluciones incluyen desde encontrar el error para el correcto funcionamiento, o buscar otro paquete que cumpla la misma función de manera correcta.

Desincronización

Durante la ejecución del algoritmo, se asume que los robots se mueven de la manera esperada, sin embargo en la realidad esta restricción no es aplicable, ya que los robots podrían tener distintas demoras y ejecución de sus motores. Por ello es necesario un mecanismo de control de movimiento (como el paquete presentado en la sección 3.2.9, AMCL) para corroborar los movimientos de los robots o un mecanismo de resincronización.

Bibliografía

[1] *A* Algorithm*. URL: https://en.wikipedia.org/wiki/A*_search_algorithm.

[2] *AMCL ROS Package*. URL: http://wiki.ros.org/amcl.

[3] Angel Ayala y col. «A Comparison of Humanoid Robot Simulators: A Quantitative Approach». En: oct. de 2020, págs. 1-6. DOI: 10.1109/ICDL-EpiRob48136.2020.9278116.

[4] Preetha Bhattacharjee y col. «Multi-robot path-planning using artificial bee colony optimization algorithm». En: *Proceedings of the 2011 3rd World Congress on Nature and Biologically Inspired Computing, NaBIC 2011*. 2011, págs. 219-224. ISBN: 9781457711237. DOI: 10.1109/NaBIC.2011.6089601.

[5] *CoppeliaSim Homepage*. URL: https://www.coppeliarobotics.com.

[6] Vishnu R. Desaraju y Jonathan P. How. «Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees». En: *2011 IEEE InternationaConference on Robotics and Automation*. 2011, págs. 4956-4961. DOI: 10.1109/ICRA.2011.5980392.

[7] Dieter Fox y col. «Monte Carlo Localization: Efficient Position Estimation for Mobile Robots». En: ene. de 1999, págs. 343-349.

[8] Atsushi Fujimori y col. «Cooperative collision avoidance between multiple mobile robots». En: *Journal of Robotic Systems - J ROBOTIC SYST* 17 (jul. de 2000). DOI: 10.1002/1097-4563(200007)17:73.0.CO;2-A.

[9] *Gazebo Homepage*. URL: http://gazebosim.org.

[10] *Gmapping ROS Package*. URL: http://wiki.ros.org/gmapping.

[11] Peter Hart, Nils Nilsson y Bertram Raphael. «A Formal Basis for the Heuristic Determination of Minimum Cost Paths». En: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), págs. 100-107. DOI: 10.1109/tssc.1968.300136. URL: https://doi.org/10.1109/tssc.1968.300136.

[12] Joseph L. Hellerstein y col. «PID Controllers». En: *Feedback Control of Computing Systems*. John Wiley y Sons, Ltd, 2004. Cap. 9, págs. 293-335. ISBN: 9780471668800. DOI: https://doi.org/10.1002/047166880X.ch9. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/047166880X.ch9. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/047166880X.ch9.

[13] Howie Choset Kevin M. Lynch Seth Hutchinson George A. Kantor Wolfram Burgard Lydia E. Kavraki y Sebastian Thrun. *Principles of Robot Motion*. 2005.

[14] Bruno Siciliano Oussama Khatib. *Springer Handbook of Robotics*. 2016.

[15] Bruno Siciliano Oussama Khatib. *Springer Handbook of Robotics Chapter 7.4*. 2016.

[16] O. Khatib. «Real-time obstacle avoidance for manipulators and mobile robots». En: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2. 1985, págs. 500-505. DOI: 10.1109/ROBOT.1985.1087247.

[17] *Khepera IV*. URL: http://www.k-team.com/khepera-iv.

[18] Steven M. LaValle. «Rapidly-exploring random trees : a new tool for path planning». En: *The annual research report* (1998).

[19] Ryan Luna y Kostas E. Bekris. «Efficient and complete centralized multi-robot path planning». En: *Proceedings of the 4th Annual Symposium on Combinatorial Search, SoCS 2011*. 2011. ISBN: 9781577355373. DOI: 10.1109/iros.2011.6095085.

[20] *MapServer ROS Package*. URL: http://wiki.ros.org/map_server.

[21] Robin R. Murphy. «Introduction to AI Robotics». En: (ene. de 2000).

[22] Seung-Hwan Park y Beom-Hee Lee. «A New Analytical Representation to Robot Path Generation with Collision Avoidance through the Use of the Collision Map». En: *International Journal of Control, Automation and Systems* 4.1 (2006), págs. 77-86.

[23] Lynne E. Parker. «Multiple Mobile Robot Teams, Path Planning and Motion Coordination in Multiple Mobile Robot Teams». En: *Encyclopedia of Complexity and Systems Science*. Springer New York, 2009, págs. 5783-5800. DOI: 10.1007/978-0-387-30440-3_344.

[24] David Parsons y John Canny. «A motion planner for multiple mobile robots». En: (1990), págs. 8-13. DOI: 10.1109/ROBOT.1990.125937.

[25] Mike Peasgood, Christopher M. Clark y John McPhee. «A complete and scalable strategy for coordinating multiple robots within roadmaps». En: *IEEE Transactions on Robotics* 24.2 (abr. de 2008), págs. 283-292. ISSN: 15523098. DOI: 10.1109/TRO.2008.918056. URL: https://www.researchgate.net/publication/3450577_A_Complete_and_Scalable_Strategy_for_Coordinating_Multiple_Robots_Within_Roadmaps.

[26] *PID Tuning*. URL: https://www.crossco.com/resources/technical/how-to-tune-pid-loops/.

[27] *Robotis LDS-01*. URL: https://www.robotis.us/360-laser-distance-sensor-lds-01-lidar/.

[28] *ROS*. URL: https://www.ros.org/.

[29] Thierry Siméon, Stéphane Leroy y Jean Paul Laumond. «Path coordination for multiple mobile robots: A resolution-complete algorithm». En: *IEEE Transactions on Robotics and Automation* 18.1 (feb. de 2002), págs. 42-49. ISSN: 1042296X. DOI: 10.1109/70.988973.

[30] Petr Švestka y Mark H. Overmars. «Coordinated path planning for multiple robots». En: *Robotics and Autonomous Systems* 23.3 (abr. de 1998), págs. 125-152. ISSN: 0921-8890. DOI: 10.1016/S0921-8890(97)00033-X.

[31] *VMWare*. URL: https://www.vmware.com/latam/products/workstation-player.html.

[32] Glenn Wagner y Howie Choset. «Subdimensional expansion for multirobot path planning». En: *Artificial Intelligence* 219 (2015), págs. 1-24. DOI: 10.1016/j.artint.2014.11.001. URL: www.elsevier.com/locate/artint.

[33] *Webots Homepage*. URL: https://cyberbotics.com.

[34] Wentao Yu y col. «A cooperative path planning algorithm for a multiple mobile robot system in a dynamic environment». En: *International Journal of Advanced Robotic Systems* 11.1 (ago. de 2014). ISSN: 17298814. DOI: 10.5772/58832.