INSTITUTO DE COMPUTACIÓN, FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA, MONTEVIDEO, URUGUAY

Informe de Proyecto de Grado Ingeniería en Computación

Planificación de Trayectorias Óptimas

Valentín Llambías, Pedro Osimani
Tutores: Facundo Benavides (InCo), Pablo Monzón (IIE)
Mayo 2022

Resumen

Un robot no-holonómico es aquel que necesita rotar para modificar su dirección. Esto se debe a que no es capaz de controlar el total de sus grados de libertad. Dentro de los robots no-holonómicos se encuentran los robots con ruedas no omnidireccionales. El problema abordado es la planificación del control de robots no-holonómico, de dos ruedas, con tracción diferencial, en particular el Khepera III. La planificación del control permitirá llevar el robot desde una posición inicial del entorno a una final, en el menor tiempo posible, sin colisionar con los obstáculos.

Se analizó el problema desde un punto de vista cinemático y se definieron los modelos matemáticos considerando los diferentes parámetros del Khepera III. Con los conocimientos adquiridos se diseñó e implemento una solución al problema de planificación de control. Esta solución consiste en primero realizar un modelado del entorno. Luego con el modelo obtenido se realizará una planificación del camino, que lleve al robot desde la posición inicial a la final. Por último, se construye una planificación de control a partir del camino obtenido anteriormente. Esta planificación de control será la que retorne los movimientos a aplicar al robot para recorrer dicho camino, de forma segura y cumpliendo el objetivo de que sea minimizando el tiempo.

Como forma de validar la planificación de control obtenida, se realizó una simulación en Webots. Para esto, se diseñó e implementó la comunicación del control obtenido en un simulador.

Los resultados obtenidos, se pudo observar que las planificaciones obtenidas por la implementación realizada, son mejores en tiempo de recorrido que un algoritmo básico de planificación de trayectorias. Además, se logró que el tiempo de procesamiento de dichos algoritmos, sea el menor posible, para no afectar negativamente la diferencia en recorrido obtenida con la planificación.

Queda como trabajo futuro la construcción de un módulo de corrección de movimiento para la simulación, para mitigar los errores introducidos en la ejecución de la planificación obtenida. Además, como trabajo futuro, se plantea ejecutar la planificación obtenida en un entorno real.

Palabras clave: No-holonómico, Khepera III, Quadtree, A*, Grid Search

Índice general

Índice de figuras

Capítulo 1

Introducción

La palabra robot proviene del checo robota o rabota que se traduce como trabajo forzado o servidumbre. El término se popularizó gracias a la obra de teatro R.U.R (Robots Universales Rossum), escrita en 1920 por el autor checo Karel Čapek. En ella, una fábrica construye humanos artificiales (robots) a partir de materia orgánica con el fin de servir al resto de las personas. Hoy por hoy, existen diferentes definiciones de lo que es un robot. Para el R.I.A (Robotics Institute of America) un robot es un dispositivo artificial programable y diseñado para realizar una variedad de tareas.

Un robot móvil es capaz de cambiar su ubicación por medio de la locomoción [16]. Los vehículos de guiado automático (fig. 1.1) , AGV por sus siglas en inglés, representan un vehículo que se mueve de manera automática y sin conductor. Son utilizados principalmente para el transporte de objetos a lo largo de rutas fijas previamente definidas. Debido a que los AGVs operan en entornos especialmente modificados, alterar su ruta es costoso y cualquier cambio imprevisto (como objetos bloqueando el camino) pueden impedir que la tarea llevada a cabo sea completada con éxito. Por lo tanto, una alternativa para sortear éste tipo de dificultades es el uso de robots móviles autónomos [16]. Un robot es verdaderamente autónomo si es capaz de llevar a cabo tareas en un entorno de forma independiente, sin control ni intervención del ser humano. Esto implica ser capaz de enfrentar cambios en el entorno, de aprender de la experiencia, y de representar internamente el entorno para llevar a cabo procesos computacionales necesarios para la navegación.

Un robot móvil necesita de un mecanismo de locomoción que lo habilite a moverse por todo su entorno. Existen múltiples formas de moverse y, hoy en día, existen robots capaces de caminar, saltar, correr, patinar, nadar, volar y rodar. Muchos de estos mecanismos fueron inspirados en animales, insectos y en el propio ser humano. Un mecanismo que no fue inspirado en la naturaleza es la rueda. La rueda además de ser simple, es altamente

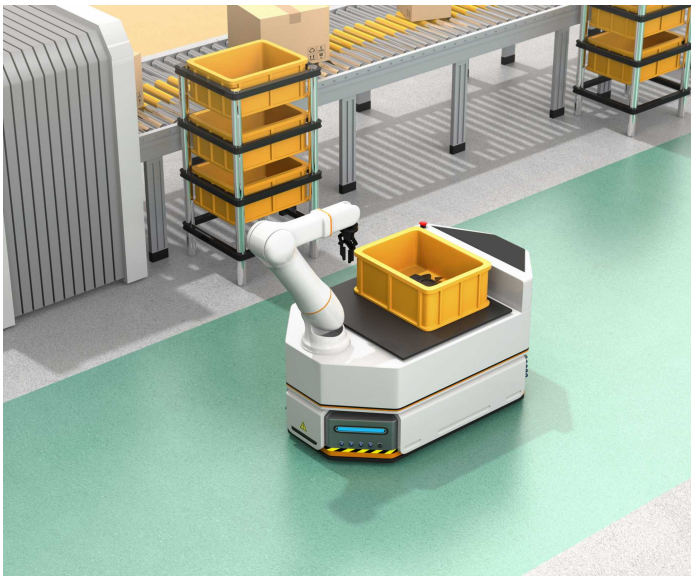


Figura 1.1: Un vehículo de guiado automático.

eficiente en terrenos planos. Es por esto que la rueda ha sido el mecanismo de locomoción más popular utilizado en la robótica y en la mayoría de los vehículos creados por el hombre.

Para resolver los problemas de locomoción de un robot, es necesario comprender aspectos de la mecánica, cinemática, dinámica y teoría de control. Por otro lado, la localización y navegación demandan conocimientos en algoritmos computacionales, teoría de la información e inteligencia artificial. En el correr de éste documento se detallarán algunos de estos conceptos.

La navegación es el problema de hallar los controles (o fuerzas) necesarios para llevar a un robot de una configuración (o estado) inicial a otra, sin que el robot colisione con ningún otro objeto, teniendo en cuenta las restricciones cinemáticas y dinámicas del robot. [5].

Para crear un plan de navegación es necesario ser capaz de especificar la posición exacta de cada punto del robot con el fin de asegurar que éste no colisione con ningún obstáculo. Esto se puede lograr con una conceptualización del entorno físico que tenga la capacidad de ser representado y procesado computacionalmente. Esta representación es fundamental para planificar el camino que lleve al robot a la configuración deseada y encontrar los movimientos necesarios que el robot debe realizar para poder recorrerlo exitosamente. Para encontrar los movimientos es necesario comprender las limitaciones cinemáticas del robot.

Antes de la década de los noventa, la planificación de caminos se direc-

cionaba a soluciones geométricas sin prestar atención a las limitaciones impuestas de los sistemas. Lozano-Pérez [15] trabajó la búsqueda de caminos en términos geométricos e introdujo el concepto de espacio de configuraciones. Este concepto define el espacio físico de trabajo del sistema, dentro del cual el robot está localizado por su posición y orientación. En la misma época se produjo la introducción de las restricciones de los sistemas en el espacio de configuraciones, lo que quiere decir que un camino encontrado no es siempre válido o ejecutable en referencia a los movimientos del robot (no holonomía).

Frente al problema de planificar una trayectoria para un sistema con restricciones cinemáticas, se plantea considerar un robot no-holonómico, en particular el Khepera III, desplazándose en un entorno conocido, con obstáculos conocidos y estáticos. El algoritmo deberá ser capaz de encontrar una trayectoria que minimice el tiempo que le llevaría al robot trasladarse desde un punto inicial al destino. A su vez, el usuario debe de poder configurar fácilmente el algoritmo para que se adecue a diferentes entornos.

La investigación realizada y los algoritmos construidos permitieron que el modelado del entorno, la planificación del camino y la planificación de trayectorias que optimicen el tiempo. De esta forma se lograron obtener los controles que se le deben aplicar al robot para llevarlo desde la posición inicial a la final. Esta implementación, puede adaptarse fácilmente a otro robot no-holonómico de similares características que el Khepera III. Para ello se deben ajustar los parámetros de configuración de entrada del algoritmo.

Para modelar el entorno se tomó como base el algoritmo de quadtree y la planificación del camino se basó en el algoritmo A*. Además, la implementación de la planificación de trayectorias, se basó en el algoritmo de Grid Search. Por último, se realizó una simulación de las soluciones obtenidas en Webots.

El resto del documento está estructurado de la siguiente forma. En el capítulo dos se desarrollará una descripción de los principales conceptos utilizados en éste proyecto. Luego capítulo tres se presenta la solución implementada para resolver el problema planteado. Después en el capítulo cuatro se detallará las pruebas realizadas sobre la solución implementada, además se realiza una comparativa contra un algoritmo básico como es el que denominamos Giro y Avanzo. Por último, en el capítulo cinco se presentará las conclusiones del proyecto y las principales líneas de trabajo futuro.

Capítulo 2

Revisión de antecedentes

Índice

2.1. Robots móviles

En las siguientes secciones se muestran los usos de los robots móviles, los diferentes tipos según su movilidad, y se hace una breve descripción de los robots diferenciales con dos ruedas junto a su modelo cinemático. Es en base a estos robots que en la sección 2.2 se describe el problema de la navegación.

2.1.1. Aplicaciones de los robots móviles autónomos

Un robot móvil autónomo es capaz de llevar a cabo tareas que involucran transporte, exploración, vigilancia, guía, inspección, etc. En particular, este tipo de robots son ideales para entornos inaccesibles u hostiles para los humanos. Algunos ejemplos son robots submarinos, planetarios, o robots operando en entornos contaminados.

El transporte es de las aplicaciones más básicas y la más empleada de los robots móviles autónomos. Estos pueden transportar productos dentro de un depósito o centros de envío incontables veces al día y en ciclos ininterrumpidos. Hoy día existen depósitos y centros de distribución masivos cuya área puede superar los ochenta mil metros cuadrados [1]. Es fácil imaginar lo bien que se adecuan los robots móviles autónomos en estos casos. Utilizar robots para tareas como el transporte, la carga y la descarga permite liberar a las personas y que estas se enfoquen en tareas más importantes, menos tediosas y con mayor valor agregado.

En el área de la salud, los robos móviles autónomos pueden ser una herramienta para el transporte de suministros y medicinas dentro de un centro de salud. Esto es aún mas importante cuando se trabaja en unidades potencialmente contaminadas por enfermedades infecciosas, previniendo así que el personal de salud esté en contacto frecuente con dichas enfermedades, y que el paciente no deje de recibir el tratamiento correcto. Además, si se equipa a los robots adecuadamente se pueden utilizar para descontaminar un espacio sin la exposición de personas a virus o bacterias.

Existe otra gran área en la cual se utilizan robots móviles que abarca la inteligencia artificial, la ciencia cognitiva y la psicología. Los robots móviles autónomos ofrecen un medio para probar hipótesis relacionadas al comportamiento, percepción y cognición.

Por último, la aplicación más difundida y conocida a nivel comercial es de la limpieza doméstica. En la actualidad, existen ya múltiples robots móviles autónomos diseñados por diferentes empresas capaces de aspirar y trapear los suelos. Estos robots son capaces de tomar decisiones en base a lo que perciben del entorno para poder llevar a cabo la limpieza de forma adecuada y segura, sin supervisión.

2.1.2. Robots holonómicos y no-holonómicos

El movimiento de un robot se puede clasificar según las limitaciones que lo restringen. Dadas las limitaciones de la cinemática del robot, el espacio de velocidad describe a los componentes del movimiento que el robot puede controlar. Por ejemplo, el espacio de velocidad de un uniciclo (fig. 2.1) puede ser representado con dos ejes, uno asociado a la velocidad lineal instantánea del uniciclo y el segundo para representar el cambio instantáneo en su orientación [22].

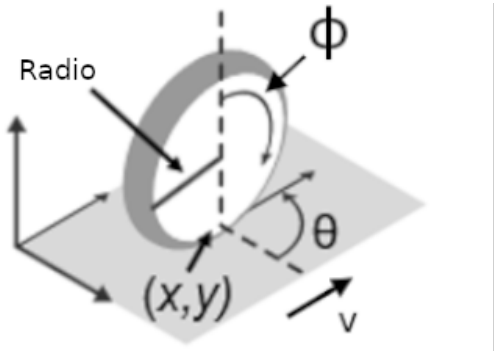


Figura 2.1: El uniciclo es una rueda que se desplaza sobre el plano horizontal.

Los grados de libertad (*DOF, degrees of freedom*) de un robot son la cantidad de coordenadas (o dimensiones) del espacio necesarias para describir posiciones alcanzables. Considerando un espacio de tres dimensiones y un robot capaz de volar libremente por el espacio, los grados de libertad son seis. Tres de ellos representan la posición (x, y, z) y tres su orientación $(roll, pitch, yaw)$ (fig. 2.2).

Un robot holonómico es capaz de controlar cada uno de sus grados de libertad, en otras palabras, es capaz de desplazarse en cada una de las coordenadas. Dicha característica le permite al robot ser capaz de modificar su dirección instantáneamente, sin necesidad de rotar o maniobrar. Algunos ejemplos de robot holonómico en un entorno de dos dimensiones, son el Uranus (fig. 2.4) y BB-8 de la saga de Star Wars (fig. 2.3). BB-8 está compuesto por una cabeza con forma de domo y un cuerpo esférico. Su cuerpo es capaz de rodar en cualquier dirección. El Uranus tiene tracción diferencial pero con unas ruedas especiales que le permiten la omnidireccionalidad.

En contraposición, un robot no-holonómico es aquel que necesita rotar para modificar su dirección ya que no puede controlar el total de sus grados de libertad. Dentro de los robots no-holonómicos se encuentran los robots con ruedas no omnidireccionales (p. ej.: ruedas estándar o ruedas orientables). En la figura 2.5 se pueden ver ejemplos de robots no holonómicos.

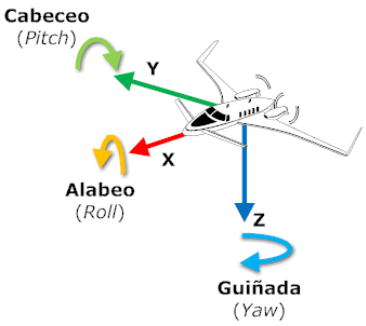


Figura 2.2: Los ángulos de rotación de una aeronave suelen estar definidos por los ejes x, y, z. El alabeo ($roll$), el cabeceo ($pitch$) y la guiñada (yaw) son las rotaciones definidas por los ejes x, y, z respectivamente.

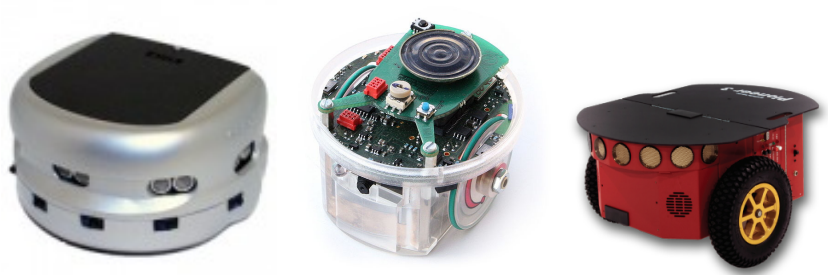
Figura 2.3: BB-8 de la saga de Star Wars es un robot del tipo holonómico.

Figura 2.4: El robot Uranus, de la Universidad Carnegie Mellon, es un robot impulsado por 4 ruedas independientes omnidireccionales. [3]

2.1.3. Robots diferencial con dos ruedas

Un robot de ruedas con tracción diferencial realiza su direccionamiento con la diferencia de velocidades de sus dos ruedas. De esta manera se puede conseguir que el robot avance en línea recta cuando los motores estén a la misma velocidad y que cuando se apliquen velocidades diferentes en los motores, este logre hacer un giro sobre si mismo. En el caso que se apliquen velocidades de igual magnitud, pero sentido opuesto, el robot girará sobre su propio eje.

Dentro de los robots diferenciales con dos rueda se encuentran los modelos Khepera (fig. 2.5a), desarrollados por la empresa K-Team [13], el robot móvil miniatura e-puck (fig. 2.5b) [9] y el Pioneer 3-DX (fig 2.5c) [21].



(a) Khepera III. (b) E-puck. (c) Pioneer 3-DX.

Figura 2.5: Tres de los robots diferenciales más populares en la investigación.

Modelo cinemático y dinámico

La cinemática del robot aplica la geometría al estudio del movimiento de los grados de libertad que este tiene [18]. El énfasis en la geometría significa que los vínculos del robot se modelan como cuerpos rígidos y se supone que sus articulaciones proporcionan rotación o traslación puras. Con la cinemática de robots se estudia la relación entre las dimensiones y la conectividad de las cadenas cinemáticas y la posición, velocidad y aceleración de cada uno de los eslabones del sistema robótico, con el fin de planificar y controlar el movimiento y calcular las fuerzas y pares de los actuadores.

El modelo cinemático estudia el movimiento del robot respecto a un sistema, o marco, de referencia fijo. Existen dos formas de describir el movimiento; a través de la cinemática directa o a través de la cinemática inversa. Para un robot móvil, la cinemática directa consiste en determinar la posición y la orientación finales del robot, conocidos los parámetros de control (p. ej. velocidad de las ruedas) y los parámetros geométricos de los elementos del robot (p. ej. distancia entre ruedas). El proceso inverso se conoce

como cinemática inversa y consiste en determinar los parámetros de control conociendo la posición y orientación final.

Esta sección se centrará únicamente en la descripción del modelo cinemático directo de robots diferenciales con dos ruedas, como los descriptos en la sección 2.1.3. En la formulación del modelo se han tomado las siguientes suposiciones con el objetivo de simplificarlo [4].

- El robot debe ser considerado como un cuerpo rígido.
- El robot se mueve en una superficie plana horizontal.
- El movimiento del robot es un rodamiento sin deslizamiento.
- El eje guía es perpendicular al plano.
- Durante el movimiento, el contacto de una rueda con el suelo se da en un único punto.

La posición del robot se describe en un marco referencial local $\{X_L, Y_L\}$, con centro el punto P, punto medio entre las dos ruedas, dentro de otro marco referencial global $\{X_G, Y_G\}$. Como se puede ver en la figura 2.6, la orientación del robot está definida por la diferencia angular entre el marco de referencia global y el local. Si θ es el ángulo de orientación del robot, la postura del mismo en el marco global $\{X_G, Y_G\}$ queda dada por los valores de (x, y, θ), tal como se muestra en la figura. 2.6. De esta forma, el movimiento del robot queda descrito por la traslación del punto P y la rotación del robot sobre el plano horizontal. Ambas acciones se pueden realizar controlando las velocidades angulares de la ruedas izquierda y derecha.

Se define el modelo cinemático directo, que permite predecir las velocidades del robot y su posición en función de los parámetros geométricos y las velocidades angulares de las ruedas. En este caso, los parámetros geométricos son el radio de cada rueda y la distancia entre las mismas.

Formalmente, dado un robot diferencial con dos ruedas, tal como se muestra en la figura 2.7, es posible aproximar la velocidad lineal del robot, V_{robot}, y su velocidad angular ω_{robot} transcurrido un cierto Δt pequeño. Sabiendo que el radio de la rueda izquierda y derecha son r_l y r_r respectivamente y que giran a una velocidad angular ω_l y ω_r se puede hallar su velocidad lineal con la siguientes ecuaciones:

$$v_l = r_l * \omega_l \tag{2.1}$$

$$v_r = r_r * \omega_r \tag{2.2}$$

Por lo tanto, se está en condiciones de calcular V_{robot} y ω_{robot} en función de v_l, v_r y la distancia entre ruedas D.

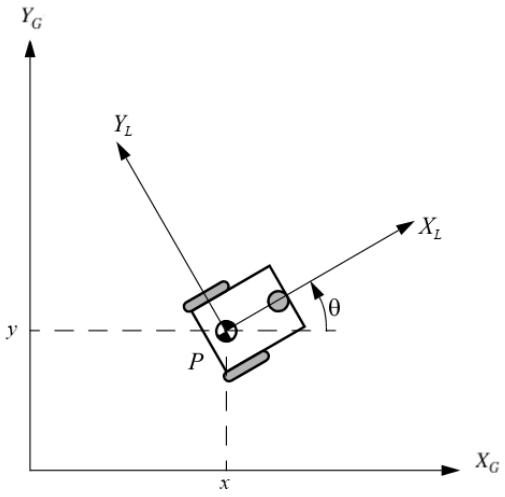


Figura 2.6: El marco de referencia global y marco local referente al robot.

$$V_{robot}(t) = \frac{v_l(t) + v_r(t)}{2} \tag{2.3}$$

$$\omega_{robot}(t) = \frac{v_r(t) - v_l(t)}{D} \tag{2.4}$$

De estas ecuaciones se desprende de que el robot se desplaza en línea recta cuando v_l y v_r son iguales y distintas de cero.

$$\begin{aligned} v_l &= v_r \\ V_{robot}(t) &= \frac{v_l(t) + v_r(t)}{2} = \frac{v_l(t) + v_l(t)}{2} = v_l(t) \\ \omega_{robot}(t) &= \frac{v_r(t) - v_l(t)}{D} = \frac{v_l(t) - v_l(t)}{D} = 0 \end{aligned} \tag{2.5}$$

Por otro lado, si ambas ruedas rotan a la misma velocidad pero en sentido opuesto ($v_r = -v_l$), el robot girara sobre su propio eje sin desplazarse.

$$\begin{aligned} v_r &= -v_l \\ V_{robot}(t) &= \frac{v_l(t) + v_r(t)}{2} = \frac{v_l(t) - v_l(t)}{2} = 0 \\ \omega_{robot}(t) &= \frac{v_r(t) - v_l(t)}{D} = \frac{-v_l(t) - v_l(t)}{D} = \frac{-2v_l(t)}{D} \end{aligned} \tag{2.6}$$

La dinámica de los robots se ocupa de la relación entre las fuerzas que actúan sobre un mecanismo de robot y las aceleraciones que producen. Pa-

ra esto, se estudia la relación entre las propiedades de masa e inercia, el movimiento, las fuerzas asociadas y el torque . De esta forma, es posible aproximar la postura del robot transcurrido un cierto Δt pequeño y conociendo la posición inicial (x_0, y_0, θ_0) utilizando las siguientes ecuaciones.

$$\begin{aligned} x(t) &= x_0 + V_{robot} * \cos\theta * \Delta t \\ y(t) &= y_0 + V_{robot} * \operatorname{sen}\theta * \Delta t \\ \theta(t) &= \theta_0 + \omega_{robot} * \Delta t \end{aligned} \tag{2.7}$$

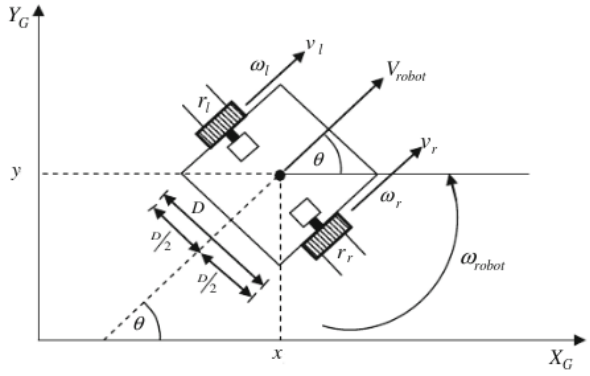


Figura 2.7: Cinemática de un robot diferencial con dos ruedas. El radio de la rueda izquierda y derecha es r_l y r_r respectivamente. La distancia entre ambas ruedas es D. Las ruedas rotan a una velocidad ω_l y ω_r y se desplazan a una velocidad v_l y v_r respectivamente. El punto P, centro del robot, está a una distancia $D/2$ de cada una y sus coordenadas son (x_l, y_l). La orientación del robot es θ. V_{robot} y ω_{robot} son la velocidad lineal y velocidad angular del robot. El valor de ambas para cierto Δt pequeño se puede aproximar a través de las ecuaciones 2.3 y 2.4.

2.2. Navegación

Para este trabajo es necesario estudiar la navegación de los robots diferenciales de dos ruedas. Por lo que en esta sección se definirá el espacio de configuración, la representación del entorno, la planificación del camino y la planificación de trayectorias. Cabe aclarar que esta descripción puede aplicarse o generalizarse a otros tipos de robots móviles o articulados.

2.2.1. Espacio de configuración

La configuración de un robot es la especificación de la posición de cada punto del mismo. Si el robot tiene k grados de libertad, la configuración se puede describir con k valores reales: q_1, q_2, ..., q_k. Queda entonces definido un punto q en un espacio de dimensión k. A este espacio se lo denomina espacio de configuración C. De esta forma, un objeto complejo de tres dimensiones puede ser descrito con un solo punto con k coordenadas.

También es de utilidad definir explícitamente el sub-espacio de configuraciones en las que el robot colisiona con algún obstáculo. Al mismo se lo conoce como espacio de configuración de obstáculos O. Una vez conocido O en el espacio, se puede calcular el espacio libre de colisiones F como el complemento de O en C. De esta forma, el problema la planificación del camino se traduce a encontrar un mapeo continuo $c : [0, 1] \rightarrow F$, con $c(0)$ la configuración inicial y $c(1)$ la configuración final.

Como se mencionó anteriormente, la posición de un robot móvil en el plano generalmente se representa con las variables (x, y, θ). En el caso de los robots diferenciales su condición de no-holonómicos hace que su velocidad se vea limitada en cada configuración (x, y, θ). Para estos casos existen formas para poder construir el espacio de configuración F, aunque lo más común para la planificación del camino es asumir que el robot es holonómico, simplificando enormemente el problema. Esta simplificación es posible ya que los robots diferenciales pueden rotar en el lugar, y por eso emular un camino holonómico. Más aún, es habitual asumir que el robot es simplemente un punto. Esto reduce el espacio de configuración para un robot móvil a una representación en dos dimensiones, siendo sus ejes x e y. Para la planificación del camino, no es necesario conocer la orientación θ. El resultado total de toda esta simplificación es espacio de configuración esencialmente idéntico al entorno físico en el que se encuentra el robot.

2.2.2. Representación del entorno

Es necesario descomponer el entorno para crear un modelo discreto que pueda ser procesable. Para esto existen varias estrategias de representar el entorno. Entre ellas se encuentran los mapas viales (mejor conocidos como roadmaps), la descomposición en celdas (exacta y aproximada) y las

funciones potenciales. En este trabajo se utilizó la representación usando descomposición en celdas.

Descomposición exacta en celdas

Este método consiste en descomponer el espacio libre de colisiones en un conjunto de regiones no superpuestas denominadas celdas. La unión de todas las celdas es exactamente el espacio libre. Los límites (o fronteras) de cada celda usualmente corresponden a un cambio crítico en el entorno físico, por ejemplo, un obstáculo. Luego, a partir de dicha descomposición se construye un grafo de conectividad para representar la adyacencia entre celdas [14]. Cada celda se representa como un nodo en el grafo y la relación de adyacencia se representa con enlaces que conectan a sus correspondientes nodos. Dos celdas son adyacentes si poseen un límite en común.

La descomposición trapezoidal es el método más popular entre los métodos descomposición exacta en celdas. Esta descomposición depende fuertemente en una representación poligonal del espacio de configuración. La figura 2.8 muestra un ejemplo de cómo queda descompuesto un espacio de configuración usando dicho método.

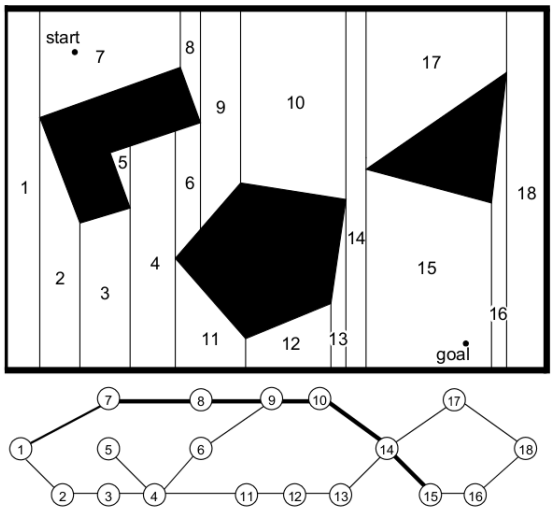


Figura 2.8: Descomposición trapezoidal de un espacio de configuración junto a su grafo de conectividad.

La principal desventaja de la descomposición exacta es que el número de celdas depende de la densidad y complejidad de los cuerpos en el entorno. Esto afecta directamente en la eficiencia computacional de la planificación en general, siendo solo factible para aquellos casos en el que el entorno es ralo.

Descomposición aproximada en celdas

Los métodos de descomposición aproximada en celdas se caracterizan por descomponer el espacio usando celdas de una forma predefinida (p. ej., rectángulos). A diferencia de la descomposición exacta, los límites de cada celda no reflejan ningún tipo de discontinuidad o cambio en el entorno físico. La unión de todas las celdas está estrictamente incluida en el espacio libre, pero no necesariamente son iguales, de ahí la aproximación.

Estos métodos de aproximación son los más populares dentro de la descomposición en celdas. Esto se debe en gran medida gracias a la popularidad de la representación en cuadrículas de ocupación (fig. 2.9). En una cuadricula de ocupación el entorno es representado por una cuadrícula discreta donde las celdas tienen un tamaño fijo. Cada una de estas celdas o está ocupada (parte de un obstáculo) o está vacía (parte del espacio libre), de ahí el nombre de cuadrícula de ocupación.

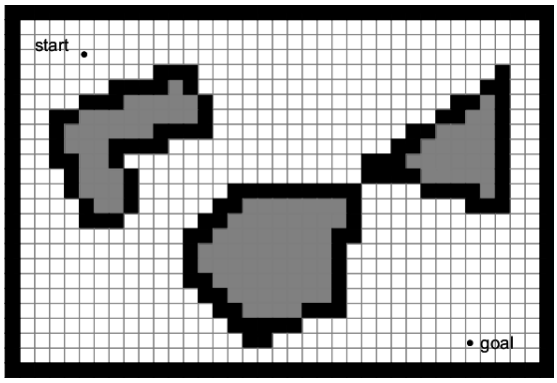


Figura 2.9: Cuadrícula de ocupación aplicada al mismo espacio.

Una vez construido el grafo de conectividad asociado a la descomposición, se está en condiciones de encontrar un camino que vaya desde el nodo que contiene la configuración inicial hasta el nodo que contenga la configuración final deseada. En la sección 2.2.3 se profundiza en los algoritmos usados en la planificación del camino en base a grafos de conectividad.

Descomposición aproximada utilizando un quadtree

Una forma de obtener una descomposición aproximada del espacio es haciendo uso de quadtrees. Un quadtree, o árbol cuaternario, como su nombre indica es un árbol donde cada nodo en el árbol tiene exactamente cuatro hijos o ninguno en el caso de las hojas. Esta estructura de datos y el algoritmo para la construcción del mismo fue creado por Raphael Finkel y J.L. Bentley en 1974 [10]. El quadtree se utiliza para describir clases de estruc-

turas de datos jerárquicas cuya propiedad común es que están basados en el principio de descomposición recursiva del espacio.

Para aplicar el método de descomposición utilizando quadtree primero se debe de delimitar el espacio de configuración con un rectángulo (o cuadrado). A este rectángulo se lo subdivide en cuatro rectángulos idénticos. Si el interior de alguno de estos rectángulos está complemente libre de obstáculos o totalmente ocupado por uno, no se continua con la subdivisión. De lo contrario, se subdivide recursivamente en cuatro nuevos rectángulos hasta llegar al tamaño mínimo de celda previamente definido. La celda de tamaño mínimo debe ser capaz de contener al robot en su totalidad. La figura 2.10 muestra el resultado de descomponer el entorno utilizando un quadtree.

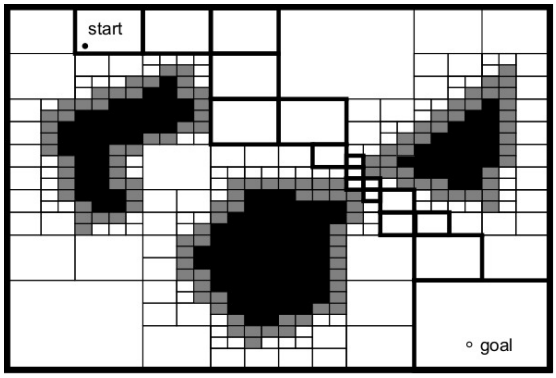


Figura 2.10: Descomposición del espacio utilizando quadtree. Las celdas blancas están completamente libres de obstáculos, las negras ocupadas por obstáculos y las grises contienen tanto áreas libres como ocupadas.

2.2.3. Planificación del camino

Reeds y Shepp [20] demostraron que la ruta óptima debe ser una de un conjunto de rutas discretas y computables. La longitud del camino no es la única métrica que se puede querer optimizar, también se pueden optimizar el consumo de energía, el tiempo, la seguridad, etc, así como combinaciones de ellas.

Existen ya varios algoritmos para resolver dicho problema. Entre los más conocidos se encuentran el algoritmo de Dijkstra ([8]), el algoritmo de Bellman - Ford ([2][11]) y el algoritmo A* ([12]). Es en este último que se hará un mayor énfasis al ser el utilizado por el programa implementado para planificar el camino.

Algoritmo A*

Para hallar el camino de menor costo entre dos vértices de un grafo ponderado, existen algoritmos que utilizan una función heurística que estima el costo de ir desde cada vértice al destino. Estos favorecen los vértices que tienen un costo menor de ir al destino, por ejemplo, el algoritmo Greedy Best-First-Search [19]. El problema con estos algoritmos es que pueden no devolver el camino de menor costo al destino, esto porque a partir de la elección de un óptimo local, puede no alcanzarse una solución óptima global. Por otro lado, existen algoritmos que favorecen a los vértices que tienen costo menor desde el origen, es el caso del algoritmo de Dijkstra. Estos algoritmos tienen el problema de que a veces es necesario escoger un vértice de costo mayor, para obtener una solución de costo menor.

Con la motivación de combinar lo mejor de estos dos tipos de algoritmos es que nace A*, el cual Peter Hard, Nils Nilsson y Bertram Raphael del Stanford Research Institute publicaron por primera vez en el año 1964 [12]. Este algoritmo de búsqueda del camino de menor costo en grafos ponderados es utilizado en muchos campos de la informática debido a su integridad y optimalidad.

A* es un algoritmo de búsqueda de mejor primero, lo que significa que está formulado en términos de grafos ponderados (fig. 2.11). A partir de un nodo inicial específico de un grafo, su objetivo es encontrar una ruta al nodo objetivo que tenga el menor costo (menor distancia recorrida, menor tiempo, etc.). Esto lo logra manteniendo un árbol de caminos que se originan en el nodo de inicio y extendiendo esos caminos un nodo vecino a la vez hasta que se satisface su criterio de parada.

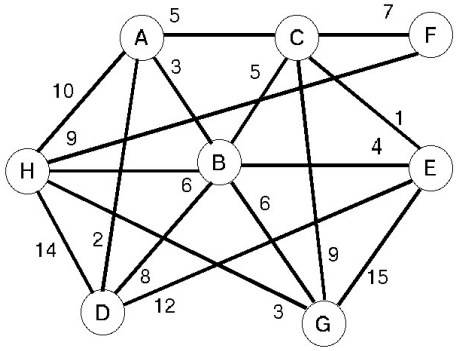


Figura 2.11: Ejemplo de Grafo Ponderado.

En cada iteración de su bucle principal, A* necesita determinar cuál de sus caminos extender. Lo hace basándose en el costo del camino y una estimación del costo requerido para llegar hasta la meta. Específicamente, A*

selecciona la ruta que minimiza la función de valuación $f(n) = g(n) + h(n)$, donde n es el siguiente nodo de camino, $h(n)$ es una función heurística que estima el costo de ir desde n hasta el destino y $g(n)$ el menor coste conocido hasta el momento desde el nodo inicial hasta el nodo n.

A la función de evaluación $f(n)$ que caracteriza a un nodo y que sirve para etiquetarlo, también se la conoce como peso de ese nodo. Cuanto menor sea el peso de un nodo, es decir, cuanto menor sea el valor de su función de evaluación, más probable será que el camino más corto atraviese ese nodo. En el cálculo del peso de los diferentes nodos se basará el funcionamiento del algoritmo A*.

Se mantendrá una lista ordenada por valor creciente del peso de los nodos que pueden ser explorados, y de ahí se seleccionará el de menor valor, que será el primero de la lista. El algoritmo empezará analizando el nodo que se toma como origen para el problema del camino más corto. Se calculará su peso, y a continuación pasará a explorar sus nodos sucesores, es decir, los nodos con los que esté unido por un enlace. Para éstos, se calculará su peso, y se seleccionará aquél de ellos que presente un menor valor, que será el que se someterá a análisis. A continuación, se calculará el peso de los nodos sucesores a este, agregando estos valores a la lista ordenada. Ahora el algoritmo continuará su ejecución seleccionando nuevamente el nodo de menor peso, este podrá no pertenecer a la rama actual de búsqueda. En caso de que, al calcular el peso de un sucesor de un nodo, sea uno ya visitado pero que tenga menor valor al calculado anteriormente, se agregando el nuevo valor a la lista de pesos. Este algoritmo continuara hasta que el sucesor del nodo analizado sea el destino. El pseudocódigo (1) detalla su funcionamiento.

2.2.4. Planificación del trayectorias óptimas

Dado un camino libre de colisiones existen diferentes maneras que el robot puede recorrerlo. Saber cual de ellas es la más rápida, puede ser fundamental para maximizar la productividad del robot y/o reducir costos. A este problema se lo conoce como la planificación de la trayectoria o planificación del control de movimiento óptimo.

Formalmente, dado un camino $c : [0, 1] \to F$ dos veces diferenciable en el espacio de configuración C y una función $s : [0, t_f] \to [0, 1]$ que asigna un valor s a cada tiempo $t \in [0, t_f]$, una trayectoria queda definida como la composición $c(s(t))$. Encontrar la trayectoria óptima en tiempo implica encontrar aquella función $s(t)$ que sujeta a las restricciones propias de los motores del robot, tenga el menor tiempo posible.

La trayectoria queda definida como el control de tiempo óptimo desde un estado inicial $(e_{inicial}, e'_{inicial})$, a un estado objetivo $(e_{destino}, e'_{destino})$. Siendo e la posición y velocidades en los diferentes ejes, e' las aceleraciones aplicadas en los diferentes ejes.

Algorithm 1 Pseudo código A*

Require: nodo $INICIAL$, nodo $DESTINO$, grafo $G = (V, E)$ con V conjunto de vértices y E aristas.

1: **function** buildPath(nodo,padres)
2: $path = nodo$ ▷ Inicializo conjunto path con $nodo$
3: $auxNodo = nodo$
4: **while** $padres[auxNodo] \neq \emptyset$ **do**
5: $auxNodo = padres[auxNodo]$
6: $path.addFront(auxNodo)$ ▷ Agrega el nodo al inicio del conjunto
7: **end while**
8: **end function**
9: $g = \emptyset$
10: $g[i] = \infty,\ \forall i \in [1..|V|],\ G = (V, E)$
11: $g[INICIAL] = 0$ ▷ Inicializar conjunto g con valor 0 para nodo inicial
12: $abiertos = \emptyset$
13: $abiertos.add(INICIO, f(INICIAL))$ ▷ Inicializar conjunto $abiertos$
14: $padres[INICIO] = \emptyset$ ▷ Inicializar conjunto $padres$
15: $haySolucion = false$
16: **while** $ABIERTOS \neq \emptyset$ & $!haySolucion$ **do**
17: $n = argminf(n), n \in abiertos$
18: $abiertos.remove(n)$
19: **if** $n \equiv DESTINO$ **then**
20: $haySolucion = true$
21: **else**
22: **for** $n' \in succ(n)$ **do** ▷ La función $succ(n)$ devuelve los nodos que son accesibles desde n
23: $g_{n'} = g[n]$+$costo(n, n')$ ▷ La función $costo(n, n')$ devuelve el costo de la arista que conecta n con n'
24: **if** $g_{n'} < g[n']$ **then**
25: $g[n'] = g_{n'}$
26: $f_{n'} = g_{n'} + h(n')$
27: $padres[n'] = n$
28: **if** $n' \notin abiertos$ **then**
29: $abiertos.add(n', f_{n'})$
30: **else**
31: $abiertos[n'] = f_{n'}$
32: **end if**
33: **end if**
34: **end for**
35: **end if**
36: **end while**

Algorithm 2 Pseudo código A*

```
37: if !haySolucion then
38:     buildPath(DESTINO, padres)
39: else
40:     ERROR                                   ▷ No hay solución
41: end if
```

En las siguientes secciones se presentan dos estrategias utilizadas para la planificación del control. Primero se presentará el algoritmo Giro y Avanzo, luego se presentara el algoritmo Grid Search.

Algoritmo Giro y Avanzo

El algoritmo más sencillo e intuitivo de planificación de trayectoria es el que se denominó Giro y Avanzo. La estrategia consiste en primero orientar al robot en dirección a la siguiente posición del camino, esto se consigue rotando sobre su posición. Una vez orientado, se avanza hasta alcanzar la siguiente posición del camino. Luego de cada movimiento, tanto rotacional como lineal, se detiene el robot. Esta planificación se repite para cada segmento del camino planificado hasta llegar al destino. En caso de querer minimizar el tiempo de recorrido, se puede optar por una estrategia bang-bang donde los motores del robot actúan a velocidad máxima.

A lo largo de este documento se hará referencia a este método denominándolo algoritmo de Giro y Avanzo o simplemente Giro y Avanzo. Este método se toma como referencia para futuros análisis comparativos.

Algoritmo Grid Search

La búsqueda en celda (Grid Search [5]) es un enfoque para la planificación de trayectorias óptimas en el tiempo.

Para el algoritmo de Grid Search se considera el siguiente sistema:

$$\begin{aligned} q' &= a \\ |a| &\leq a_{max} \end{aligned} \tag{2.8}$$

Donde a es el control de la aceleración y q es la posición unidimensional, por lo que el sistema puede verse como una masa puntual que se mueve sobre una línea con una fuerza de control.

La primera consideración de este algoritmo consiste en tomar un espacio discretizado del control establecido en $\{-a_{max}, 0, a_{max}\}$ y un intervalo de tiempo h constante.

Tomando el estado inicial $(q_{inicial}, q'_{inicial})$ como la raíz del árbol, se integran los tres controles hacia adelante en el tiempo por h para obtener tres nuevos estados. Estos tres estados nuevos serán los hijos de la raíz (figura

2.12). Luego de cada uno de estos tres estados, se los integra con los controles hacia adelante en el tiempo h y se obtiene un nuevo nivel del árbol. Si la trayectoria hacia un nuevo nodo en el árbol pasa a través de un obstáculo o excede un límite de velocidad, este nodo se poda.

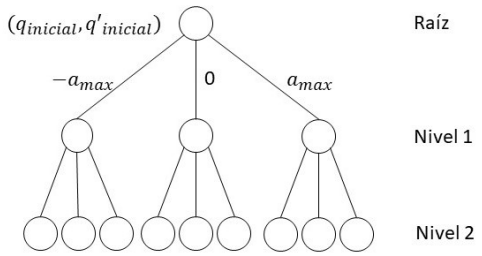


Figura 2.12: Árbol de búsqueda para tres controles

Se continua este proceso hasta que una trayectoria alcanza un estado en una región objetivo específica. La trayectoria se especifica mediante la concatenación de controles aplicados desde el nodo raíz hasta el nodo final. Dado que la búsqueda es primero en amplitud, explorando todos los estados alcanzables en el tiempo kh antes de pasar al tiempo $(k+1)h$, la trayectoria es óptima en el tiempo para la discretización elegida de tiempo y controles. El pseudocódigo 3 detalla su funcionamiento.

Se llama a esto Grid Search porque cada uno de los nodos alcanzados durante el crecimiento del árbol se encuentra en una cuadrícula regular en el espacio de estados. De cualquier estado, el nuevo estado obtenido al integrar las aceleraciones discretizadas para el tiempo h implicarán un cambio a un múltiplo integral de $a_{max}h$ y un cambio en q igual a un múltiplo integral de $\frac{1}{2}a_{max}h^2$.

Un ejemplo de la cuadricula de este tipo se muestra en la figura 2.13. El árbol de búsqueda que se muestra en esta cuadrícula tiene dos niveles de profundidad, comenzando en $(0, 0)$. En la figura las trayectorias reales entre vértices no son líneas rectas, sino cuadráticas.

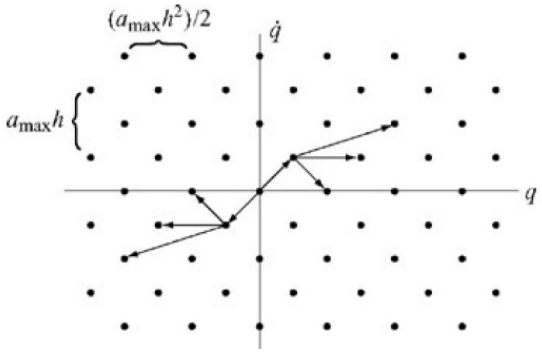


Figura 2.13: El árbol de búsqueda de la figura 2.12 se muestra en la cuadrícula del espacio de estado. Imagen extraída de [5]

Algorithm 3 Pseudo código de Grid Search

Require: estado inicial $(q_{inicial}, q'_{inicial})$, región destino G, conjunto de controles C y h que será el Δt a aplicar.

$T = \emptyset$
$T[0].add(q_{inicial}, q'_{inicial})$
$nivel = 0$
$resuelto = false$
$colaEstados = T[0]$
while $!resuelto$ & $!isEmpty(colaEstados)$ **do**
    for $e_{act} \in colaEstados)$ **do**
        for $c \in C$ **do**
            $q' = e_{act} + ch$
            $q = e_{act} + q'h$
            $e' = (q, q')$
            if $e' \notin T$ **then**
                $T(nivel + 1).add(e')$
            end if
            $resuelto = e' \in G$
        end for
    end for
    $nivel = nivel + 1$
    $colaEstados = T[nivel]$
end while
if $!resuelto$ **then**
    Devolver ERROR
end if

Capítulo 3

Solución implementada

Índice

En este capítulo se describen los aspectos más relevantes de la solución propuesta para el problema de la navegación (sección 2.2). Dicho problema se resuelve para un entorno estático conocido donde el único objeto móvil es el robot. La solución será específica para robots diferenciales con dos ruedas, los cuales serán modelados como un punto.

Se realizó una planificación fuera de línea cuyo objetivo es encontrar una o más secuencias de controles que lleven al robot desde su configuración inicial a la final en el menor tiempo posible.

Los obstáculos son representados por figuras geométricas cuya dimensión y ubicación dentro del entorno son conocidas. Esto implica que el robot no necesitaría de sensores para obtener la información relevante al entorno durante la planificación.

La solución se basa fuertemente en los aspectos teóricos ya mencionados en el capítulo 2.

3.1. Descripción general de la solución

Para la resolución del problema se implementó un programa fuera de línea que recibe la descripción del entorno, las propiedades del robot, y las coordenadas de origen y destino. Con esta información el constructor planifica una o varias trayectorias, todas ellas tienen el mismo tiempo de recorrido, que dejan al robot cerca del destino dado por una tolerancia Δd. Dentro de ese conjunto de trayectorias, se toma aquella que deje al robot más cerca del destino para simularla.

Como se puede apreciar en la figura 3.1, la solución está compuesta por una secuencia de cuatro módulos donde la salida de cada uno es consumida por el siguiente hasta obtener la salida final. Estos cuatro módulos los denominaremos (en orden de secuencia) Modelador del Entorno, Planificador de Camino, Planificador de Control y Emisor de Comandos. La ventaja de este diseño es que permite cambiar fácilmente la implementación de cualquiera de ellos siempre y cuando se respete la interfaz entre cada uno. A modo de ejemplo, si se quisiese usar otra estrategia para representar el entorno, bastaría con cambiar el primer módulo.

El último modulo, el emisor de comandos, se comunica con el simulador. Este recibe los comandos o velocidades que se deben aplicar al robot para llegar al destino.

3.2. Modelador del entorno

Como se explica en la Sección 2.2.2, lo primero que se debe de hacer es representar el entorno en una estructura admitida por el planificador del camino. Para llevar a cabo esta tarea, se implementó el modelador del

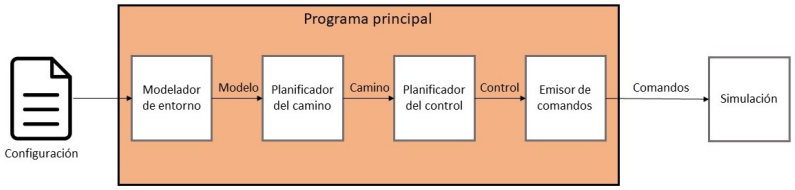


Figura 3.1: El programa principal está compuesto de cuatro módulos.

entorno. Tal como se muestra en la figura 3.2, el módulo toma como parámetros de entrada los puntos inicial y final, la ubicación de los obstáculos, la dimensión del entorno, el diámetro del robot y un margen porcentual de seguridad que junto al diámetro definen cuánto se deberán agrandar los obstáculos. Es necesario agrandar los obstáculos ya que se desea representar al robot como un punto (su centro) para poder aplicar las ecuaciones cinemáticas vistas en la sección 2.1.3. Al agrandar los obstáculos y considerar al robot como un punto se garantiza la conservación de las relaciones de distancia entre el robot y los obstáculos. Este resultado intermedio es el espacio de configuración (fig. 3.3). En la figura 3.3a se pueden ver el punto de partida (start), el punto objetivo (goal) y en negro los obstáculos. A partir de este es que se construye el modelo.

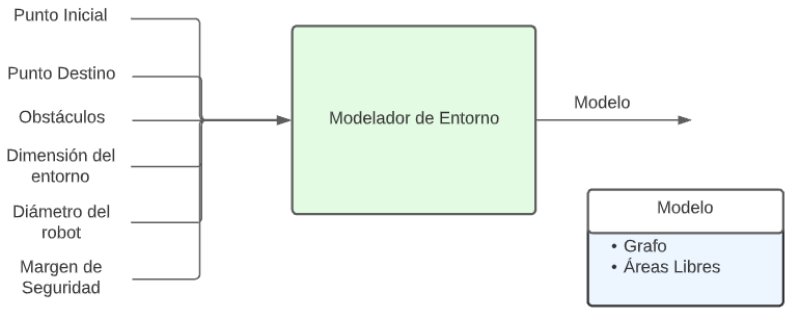
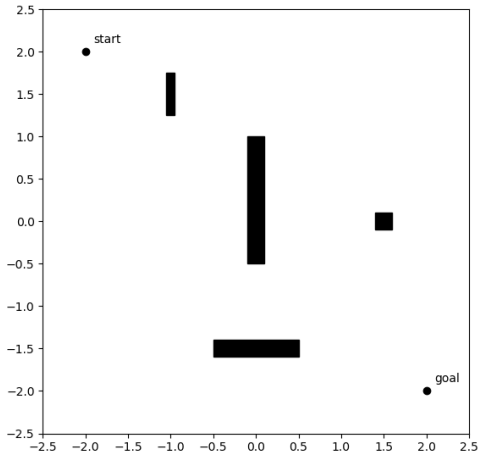
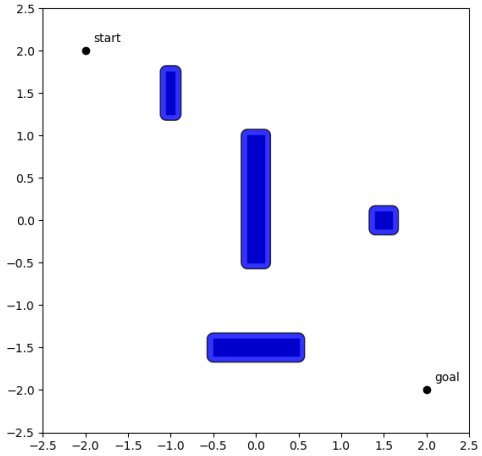


Figura 3.2: El Modelador del entorno, parámetros recibidos y salida.

El Modelador descompone el espacio libre utilizando un quadtree, tal como se explica en la sección 2.2.2. El tamaño mínimo de celda está definido por el diámetro del robot más el margen de seguridad. A partir de la descomposición se construye el grafo de salida cuyos nodos representan los centros de las celdas del quadtree. Cada uno de los nodos tendrá asociado las coordenadas de su centro. Si dos celdas son limítrofes, en el grafo serán adyacentes y el peso asociado al enlace será la distancia entre sus centros.



(a) Entorno de ejemplo.



(b) Los obstáculos agrandados son los de color azul.

Figura 3.3: El entorno de ejemplo junto a su espacio de configuración.

Una vez construido el grafo utilizando el algoritmo quadtree, se le añaden al grafo los nodos correspondientes a los puntos inicial y final. Estos nodos serán adyacentes al nodo correspondiente a celda que los contiene. La salida final del módulo está compuesta por el grafo y el conjunto de celdas libres.

En la figura 3.4 se puede ver el resultado de descomponer el espacio de la figura 3.3 utilizando el Modelador del Entorno. Las celdas con tono rojo están ocupadas mientras las blancas libres. Las zonas más oscuras, dentro de las celdas ocupadas corresponden a los obstáculos agrandados.

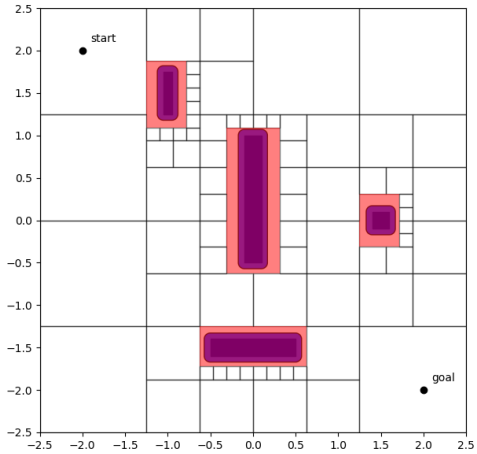


Figura 3.4: La descomposición resultante luego de aplicar el algoritmo quadtree al espacio de configuración. Las celdas con tono rojo están ocupadas mientras las blancas libres. La zona mas oscura dentro de las celdas ocupadas corresponden a los obstáculos agrandados.

3.3. Planificador del camino

Este módulo recibe como entrada el modelo, la posición inicial y la posición final. La salida del mismo consiste de dos caminos que van desde el punto inicial al final. Uno de los caminos es hallado utilizando el algoritmo A*, mientras que el otro camino es una mejora del anterior haciendo un uso más inteligente de las celdas libres. Ambos caminos buscan minimizar la distancia recorrida.

Como se verá más adelante, solo se hace uso del camino mejorado, pero si se deseara se puede utilizar el hallado por A*. Ambos son compatibles con

la entrada del siguiente módulo. La figura 3.5 es una representación visual de los parámetros de entrada y la salida del módulo.

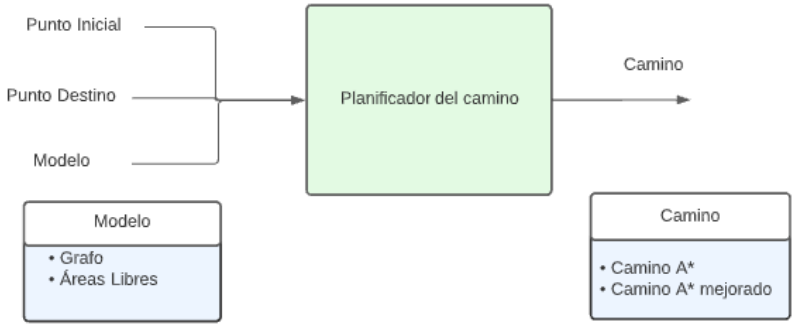


Figura 3.5: El planificador del camino recibe como parámetros los puntos inicial y final, y el modelo previamente construido. La salida está compuesta por dos caminos, uno hallado usando el algoritmo A* y el otro una mejora sobre el anterior.

Como se mencionó anteriormente, para hallar el camino más corto se procesa el grafo utilizando el algoritmo A*. En este caso, para la función heurística $f(n) = g(n) + h'(n)$ se toma $g(n)$ como la distancia recorrida hasta el nodo n y la función $h'(n)$ equivale a la distancia euclidiana entre el nodo n y el nodo destino.

Como ejemplo, en la figura 3.6 se puede observar que $f(p_2) = 18$, dado que $g(p_2) = 3 + 5$ y $h'(p_2) = 10$. Por lo que p_2 tiene menor $f(n)$ que los nodos p_3 y p_4, por lo que es el siguiente escogido.

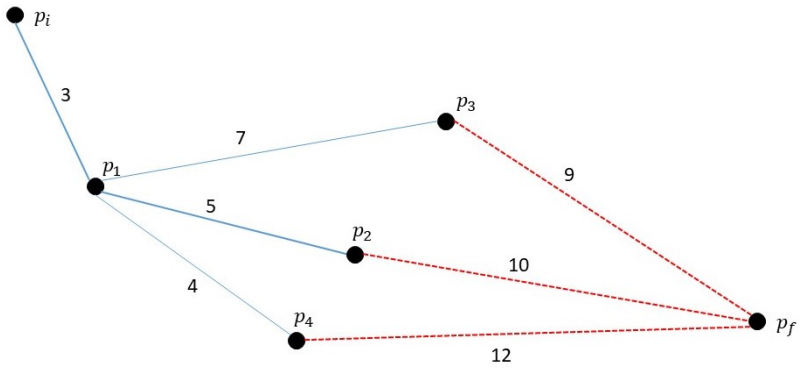


Figura 3.6: Puntos con sus distancias utilizadas para calcular $f(n)$. En este caso, $f(p_2) = 18$, dado que $g(p_2) = 3 + 5$ y $h'(p_2) = 10$

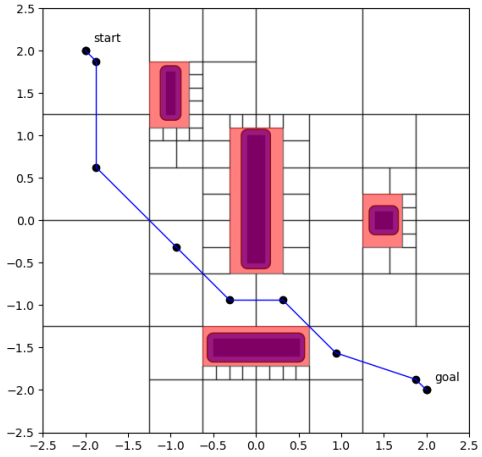


Figura 3.7: El camino resultante de aplicar A* sobre el grafo del quadtree. Se puede ver que el camino, además de pasar por los puntos de inicio y destino, pasa por centros de celdas del quadtree.

Como se puede ver en la figura 3.7, utilizar el camino hallado por A* obliga al robot a pasar por los centros de las celdas del quadtree. Teniendo en cuenta que el robot puede moverse sin colisionar dentro una celda libre, se puede mejorar el camino obtenido para que no necesariamente deba pasar por los centros. Esta mejora se realiza tomando el punto de ingreso a la celda y el punto de salida de la misma para unirlos (fig. 3.8). El resultado, es un nuevo camino igualmente seguro, ya que los puntos de entrada y salida de las celdas son los mismos que para A*.

Un tratamiento especial requiere las celdas que contienen a la posición inicial p_{ini} y a la posición final p_{fin}. Esto se debe a que no necesariamente existe punto de entrada o punto de salida de sus respectivas celdas. Por lo cual, para la celda que contiene a p_{ini} se toma como punto de entrada mismísimo p_{ini}. Análogamente, para la celda que contiene a p_{fin}, se toma como punto de salida a p_{fin}. Para un mejor entendimiento de la problemática se puede tomar como ejemplo la figura 3.9.

En definitiva, se encuentra un camino más corto que A* y se retornan ambos. En la figura 3.10 se puede observar la comparativa de los caminos obtenidos, azul el obtenido con A* y el rojo obtenido luego de la mejora.

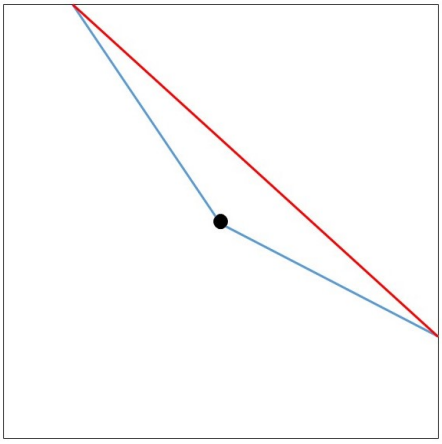


Figura 3.8: El camino azul es el obtenido por A* y el camino rojo es el optimizado.

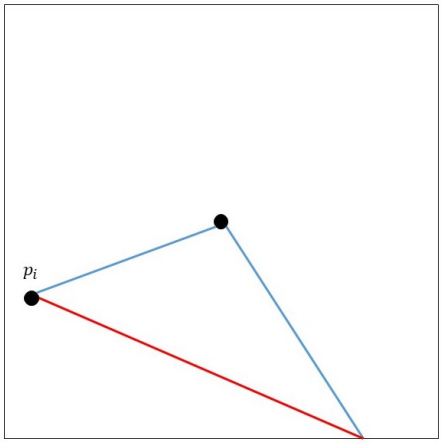
3.4. Planificador del control

El planificador de control es el módulo encargado de hallar secuencias de controles (o movimientos) necesarios para recorrer el camino planificado en tiempo mínimo. En este caso, los controles son la velocidades lineales de las ruedas del robot v_l y v_r que se ejecutarán durante un cierto Δt. Por lo tanto, la salida del módulo deberá al menos tener un conjunto que contiene aquellas secuencias de tuplas $(v_l, v_r, \Delta t)$ que recorren el camino planificado en el menor tiempo. Para este módulo se hizo una implementación basada en Grid Search (sección 2.2.4).

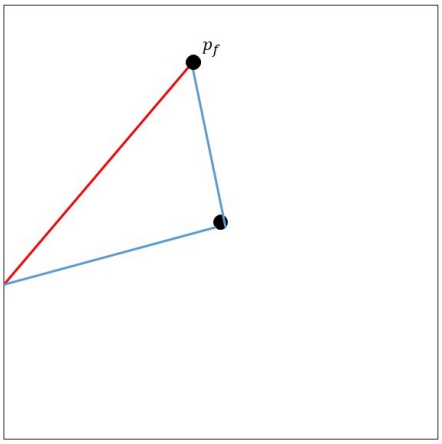
3.4.1. Planificación basada en Grid Search

Grid search por su naturaleza tiene un alto costo de procesamiento ya que el árbol de búsqueda crece de forma exponencial en función de los posibles controles. Es por esta razón que se adaptó el algoritmo para obtener resultados satisfactorios, pero de manera más eficiente. Esto se puede conseguir aplicando podas o filtros al árbol con el objetivo de acotar la cantidad de nodos a procesar. Los filtros evalúan el estado del nodo y determinan si pasando por ese nodo se llegaría al punto de destino en el menor tiempo posible. Si el nodo no cumple con las condiciones del filtro, se descarta el nodo. En la Sección 3.4.5 se describen los filtros implementados.

Por otro lado, en este caso ya se conoce un camino que lleve al destino. Teniendo un camino como guía no es necesario abarcar grandes áreas de búsqueda para encontrar los controles óptimos. Esto se puede traducir a nuevas condiciones de poda en la implementación del algoritmo. Por ejem-



(a) Optimización del camino para la posición inicial p_{ini}.



(b) Optimización del camino para la posición final p_{fin}.

Figura 3.9: Optimización realizada para la posición inicial p_{ini} y final p_{fin}. El camino azul es el obtenido por A* y el camino rojo es el optimizado.

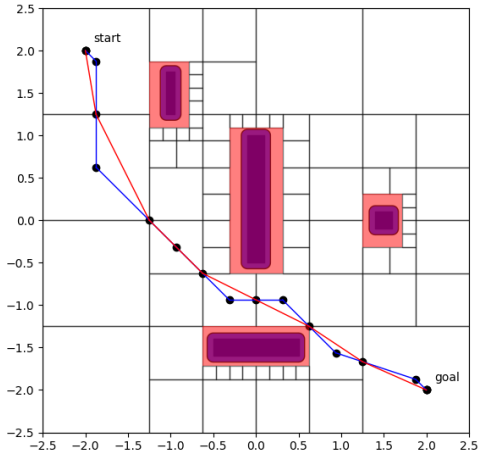


Figura 3.10: Se puede ver en azul el camino obtenido con el algoritmo A* y el rojo obtenido luego de la mejora. Para este ejemplo la distancia del camino obtenido con A* es de $6,31$ metros, mientras la distancia luego de la mejora es de $6,01$ metros.

plo, si los controles llevan al robot a alejarse demasiado del mismo, eliminar esa rama del árbol. En las siguientes secciones se explica en más detalle las diferentes formas en las que se aprovechó el camino y las correspondientes modificaciones al Grid Search.

Una de las notorias diferencias con el Grid Search descripto en la sección 2.2.4 es que en esta implementación se consideran los controles en función de la velocidad y no de la aceleración.

Al tomar en cuenta la velocidad de las ruedas como el control posible algunas consideraciones se tuvieron durante el diseño del algoritmo. En un inicio, al igual que lo hace Grid Search, se eligió una estrategia bang-bang, donde en todo momento el robot debe ir a la velocidad máxima posible. Sin embargo, en los primeros ensayos no hubo resultados muy prometedores. En consecuencia, se optó por controles que sean fracciones de la velocidad máxima.

Otra consideración que se tuvo en cuenta fue si era posible emplear un Δt diferente en cada nivel del árbol de forma tal que el robot recorra la mayor cantidad de distancia o sea capaz de girar poco según la necesidad. Siempre teniendo en cuenta el objetivo. Un ejemplo de esto, es contar con Δt mayor, para avanzar en línea recta hasta el siguiente punto, logrando que esto se haga en un solo paso y no en varios. De esta forma se reducirían la cantidad de niveles de procesamiento redundantes. Como se explicará

más adelante, la función que decida que Δt utilizar es clave ya que si se es muy ambicioso en la distancia que se quiere recorrer se pueden llegar a perder posibles soluciones y si el Δt resultante es muy pequeño impactará en la eficiencia del algoritmo.

En las siguientes secciones se aborda la implementación del módulo. Cuáles son sus entradas y su salida, los filtros, los controles aplicados, el Δt variable con las funciones implicadas y demás características del módulo. También se describirá como el algoritmo hace uso del camino recibido.

3.4.2. Parámetros de entrada y salida

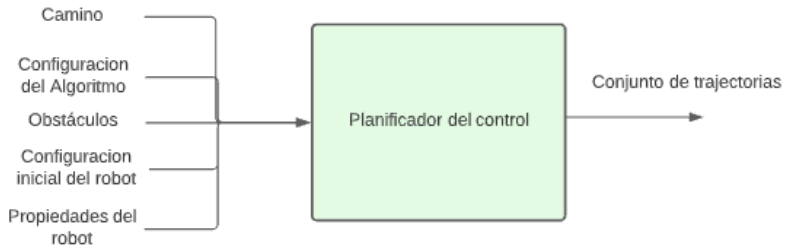


Figura 3.11: El planificador del control recibe un camino que va desde el puntos inicial al final, parámetros de configuración que permiten modificar el comportamiento del modulo, la ubicación de los obstáculos y la configuración inicial del robot $(x_{ini}, y_{ini}, \theta_{ini})$. También recibe el diámetro del robot y un margen de seguridad que en conjunto definen una banda de seguridad. La salida es un conjunto de trayectorias o secuencias de controles encontrados para recorrer el camino en el menor tiempo posible.

Parámetros de entrada.

Tal como se muestra en la figura 3.11 el módulo recibe parámetros de las siguientes entradas:

- **Configuración inicial del robot**: el estado inicial en el que se encuentra el robot. Formalmente: $(x_{ini}, y_{ini}, \theta_{ini})$.
- **Camino**: camino planificado desde el punto inicial $p_{ini} = (x_{ini}, y_{ini})$ al punto final $p_{fin} = (x_{fin}, y_{fin})$. Se representa como una lista de puntos, $p_{ini} = p_1, p_2, ..., p_{n-1}, p_n = p_{fin}$.
- **Obstáculos**: la ubicación de cada obstáculo en el entorno. Se utiliza para crear la banda de seguridad.

- **Configuración del algoritmo**: parámetros que modifican el comportamiento del algoritmo según lo deseado.
  - Δd: distancia en metros que determina el máximo error tolerable entorno al destino.
  - Δt_{min}: el intervalo de tiempo mínimo posible a utilizar en cada nivel del algoritmo. Está expresado en segundos.
  - $M = \{m_1, .., m_m\}$: conjunto de valores a multiplicar por la velocidad máxima del robot para formar el conjunto de controladores a aplicar.
  - α: valor entero que será utilizado por el filtro de proximidad al destino.
  - β: valor entero que será utilizado por el filtro de proximidad al camino.
  - γ: valor entero que será utilizado por la frecuencia de aplicación de filtros.
- **Propiedades del robot**: valores que definen al robot.
  - v_{max}: la velocidad lineal máxima que que pueden alcanzar las ruedas del robot. Está expresada en m/s.
  - d_r: el diámetro del robot en metros.
  - D: la distancia en metros entre ruedas del robot.
  - s_p: margen de seguridad. Es el porcentaje del diámetro del robot utilizado para determinar en qué áreas entorno al camino el robot se puede mover sin colisionar.

Parámetros de salida.

La salida del módulo es conjunto de trayectorias que llevan al robot desde su configuración inicial a una cercanía del punto final p_{fin} sin tener en cuenta la orientación final. Al robot le llevaría el mismo tiempo recorrer cada una de ellas. Este tiempo es el mínimo encontrado por el algoritmo. La diferencia entre cada trayectoria es la secuencia de elementos que la componen. Cada elemento está compuesto por las siguientes propiedades:

- (x_i, y_i, θ_i): configuración del robot en ese instante i.
- $(v_{izq}, v_{der})_i$: control que se debe aplicar para llevar al robot a la configuración del instante $i + 1$.
- Δt_i: el tiempo en segundos que se debe aplicar el control. Si la trayectoria tiene tamaño l y Δt_{tray} es el tiempo para recorrerla, se cumple que $\sum_{i=1}^{l} \Delta t_i = \Delta t_{tray}$.

3.4.3. Conjunto de controles

Como se adelantó anteriormente, los controles a aplicar en cada nivel del árbol de búsqueda serán función de la v_{max}. Cada control $(v_{izq}, v_{der})_i$ expresa la velocidad que se le debe aplicar a la rueda izquierda (v_{izq}) y a la rueda derecha (v_{der}) para llevar al robot a la siguiente configuración (o estado). Los valores que pueden tomar v_{izq} y v_{der} se calculan en función de v_{max} y el conjunto de entrada $M = \{m_1, ..., m_m\}$. Si V es el conjunto de posibles valores para v_{izq} y v_{der}, este está definido por la ecuación (3.1).

$$V = \{0, v_{max}, v_{max} * m_1, ..., v_{max} * m_m\} \tag{3.1}$$

Entonces, el conjunto de controles posibles para cada nivel, al que se denominó M_{gs}, esta definido como:

$$M_{gs} = V \times V - \{(0, 0)\} \tag{3.2}$$

Se le quita el control $(0, 0)$ ya que no se desea que el robot frene en los niveles intermedios. El control $(0, 0)$ solo se aplicará una vez que el robot llegue al destino p_{fin}.

3.4.4. Metodología

Como el camino planificado y el área búsqueda dentro del entorno pueden llegar a ser muy grandes, el procesamiento del algoritmo puede que sea realmente muy costoso. Es por eso que se decidió dividir el problema en sub problemas más pequeños donde la composición de sus soluciones es la solución al problema inicial. En lugar de buscar las posibles trayectorias (secuencia de controles) para todo el camino, es decir desde el punto p_{ini} al punto p_{fin}, se buscará las trayectorias desde p_{ini} a p_3 pasando cerca de p_2. Una vez encontradas esas trayectorias, se toman las configuraciones del robot más cercanas a p_2 junto a las secuencias de controles que llevaron al robot a ellas. Teniendo en cuenta esta información, se buscará ahora las trayectorias que lleven desde esas nuevas configuraciones al punto p_4 pasando cerca de p_3. Si se unen de forma adecuada las primeras trayectorias con las segundas, se tiene un nuevo conjunto de trayectorias que van desde p_{ini} a p_4 pasando cerca de p_2 y p_3. Este procedimiento se repite hasta llegar a p_{fin} y la solución final se construye concatenando las trayectorias encontradas con sus correspondientes de la última iteración.

Se decidió dividir el camino en secuencias de tres puntos ya que es la mínima cantidad de puntos que permiten planificar las curvas correctamente.

La forma de ejecutar el algoritmo es la siguiente. En la iteración i se toman la configuración q_i, el punto medio p_{i+1} y el punto final p_{i+2}. Se ejecuta el algoritmo de Grid Search hasta obtener el conjunto de trayectorias

que lleven al robot desde p_i hasta p_{i+2}. Para cada una de las soluciones se toma la posición del recorrido más próxima a p_{i+1}, esto se puede ver en la imagen 3.12. Luego, se vuelve a ejecutar el algoritmo nuevamente tomando como posiciones iniciales el conjunto anteriormente mencionado, con destino p_{i+3} y posición media p_{i+2}. Este proceso se repite hasta obtener la lista de posiciones que llevan al robot hasta la posición final ($p_{i+3} == p_{fin}$).

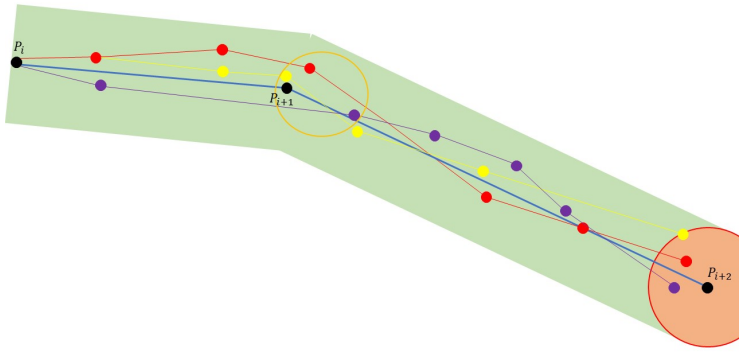


Figura 3.12: Los caminos rojo, amarillo y violeta son las soluciones devueltas por el algoritmo Grid Search y las posiciones dentro del circulo naranja serán las posiciones iniciales para la próxima iteración.

Una vez alcanzado la posición final p_{fin}, se devuelve el conjunto de soluciones obtenidas que llevan al robot de la posición inicial p_{ini} a la posición final p_{fin}.

El pseudo código del algoritmo 4 corresponde al algoritmo implementado. Se puede ver como se emplea Grid Search en los tramos del camino planificado.

3.4.5. Filtros

Dado el nivel n del árbol y c el tamaño del conjunto M_{gs}, el algoritmo de Grid Search procesa todas las hojas de dicho nivel, generando c^{n+1} hojas para el nivel $n + 1$. Este crecimiento exponencial es el inconveniente que tiene la aplicación del algoritmo Grid Search sobre entornos reales.

Teniendo en cuenta que el tiempo es la variable a minimizar y que la complejidad de procesamiento del Grid Search es exponencial, no es de mucha utilidad que los resultados obtenidos minimicen el tiempo de recorrido si se va a invertir potencialmente un tiempo muy elevado de procesamiento. Es por estas razones que surge la idea de aplicarle filtros que, de forma inteligente, permitan controlar el crecimiento del árbol de búsqueda sin perder posibles soluciones.

Algorithm 4 Pseudo código planificación de trayectorias óptimas

Require: Lista de puntos del camino planificado CP, modelo del entorno $modelo$

1: $p_{ini} = CP[1]$
2: $p_{int} = CP[2]$
3: $sol = \emptyset$
4: $i = 3$
5: **while** $i \leq size(CP)$ **do**
6: $p_{fin} = CP[i]$
7: $sol = gridSearch(p_{ini}, p_{fin}, modelo)$ ▷ Ejecutar el algoritmo Grid Search desde el conjunto de puntos p_{ini} a p_{fin}
8: **if** $i < size(CP)$ **then**
9: $p_{ini} = getPuntosMedios(sol, p_{int})$ ▷ Función getPuntosMedios devuelve el conjunto de puntos cercanos a p_{int} de cada solución
10: $p_{int} = p_{fin}$
11: **end if**
12: $i = i + 1$
13: **end while**
14: return sol

En este caso, se implementaron cuatro condiciones de filtro que se denominaron Visitados, Banda de seguridad, Proximidad al destino y Proximidad al camino planificado.

Visitados

Durante la búsqueda es posible llegar a una a configuración (x, y, θ) similar a una alcanzada anteriormente. Cuando esto sucede para no seguir añadiendo nodos a procesar o que el proceso caiga en bucles infinitos. Por lo tanto, se elimina la nueva configuración. A este filtro se lo denominó Visitados.

Ahora bien, ¿cómo se determina que dos configuraciones son equivalentes? En este caso, se consideran (x_i, y_i, θ_i) y (x_j, y_j, θ_j) equivalentes cuando la diferencia en la orientación entre ambas no es mayor a la tolerancia angular ($\Delta\theta_{gs}$) y la distancia entre los puntos (x_i, y_i) y (x_j, y_j) no es mayor tolerancia en distancia (Δd_{gs}). Formalmente, dos configuraciones (x_i, y_i, θ_i) y (x_j, y_j, θ_j) se consideran equivalentes si cumplen con la condición 3.3.

$$p_i \equiv p_j \leftrightarrow \left(|\theta_i - \theta_j| \leq \Delta\theta_{gs} \& \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \leq \Delta d_{gs} \right) \tag{3.3}$$

Tanto $\Delta\theta_{gs}$ y Δd_{gs} quedan definidas en función del conjunto de velocidades V y el parámetro de entrada Δt_{min}. Para el $\Delta\theta_{gs}$ también se to-

ma en cuenta la distancia entre ruedas (D). Primero se toma la velocidad mínima v_{min} del conjunto V sin tener en cuenta el 0, es decir $v_{min} = argmin(V - \{0\})$. Luego la tolerancia angular $\Delta\theta_{gs}$ se calcula utilizando la ecuación (3.4).

$$\Delta\theta_{gs} = \frac{v_{min}}{D}\Delta t_{min} \tag{3.4}$$

Para el cálculo de la tolerancia Δd_{gs} se utiliza la ecuación (3.5)

$$\Delta d_{gs} = v_{min}\Delta t_{min} \tag{3.5}$$

Los parámetros $\Delta\theta_{gs}$ y Δd_{gs} son los que van a determinar si una configuración (x_p, y_p, θ_p) es igual a otra (x_q, y_q, θ_q). Para considerar que estas dos posiciones son iguales deben cumplir (3.6)

$$(|\theta_p - \theta_q| \leq \Delta\theta_{gs}) \,\&\, \left(\sqrt{(x_p - x_q)^2 + (y_p - y_q)^2} \leq \Delta d_{gs}\right) \tag{3.6}$$

Como ejemplo, en la figura 3.13 se puede ver como la configuración p_2 se considera igual a p_1 dado que la diferencia de inclinación θ_2 es igual o menor a $\Delta\theta_{gs}$ y la distancia entre los puntos es igual o menor a Δd_{gs}. En cambio, la configuración p_3 se consideran distinta a p_1 dado que tiene una diferencia de inclinación θ_3 mayor a $\Delta\theta_{gs}$. Para el caso de la posición p_4, se considera distinta a la posición p_1 porque la distancia entre ellas es mayor a Δd_{gs}.

Banda de seguridad

La banda de seguridad (fig. 3.14) tiene doble propósito, acotar el conjunto de movimientos válidos en cada nivel del árbol de búsqueda, y asegurar que si el robot avanza hacia un punto dentro de la banda, no va a colisionar contra ningún obstáculo. Esta banda se construye alrededor del camino planificado, con un ancho (a_{bs}) determinado por el diámetro del robot (d_r) y el margen de seguridad (s_p) pasados por parámetro. Específicamente, a_{bs} se calcula con la ecuación 3.7. Este ancho está, al igual que todas las distancias manejadas, en metros.

$$a_{bs} = \frac{d_r}{2}\left(\frac{s_p}{100}\right) \tag{3.7}$$

Para asegurar que la banda esté libre de obstáculos, a la misma se le quitarán aquellas áreas que se solapen con áreas ocupadas por los obstáculos aumentados. Esto se puede observar con el ejemplo de la figura 3.15.

La banda de seguridad resultante permitirá filtrar aquellos movimientos que dejan al robot fuera de la banda. A su vez, filtrará aquellos movimientos que, si bien dejan al robot dentro de la banda, su recorrido se realiza saliendo de esta. Estas dos formas de filtrado se puede visualizar en la figura 3.16,

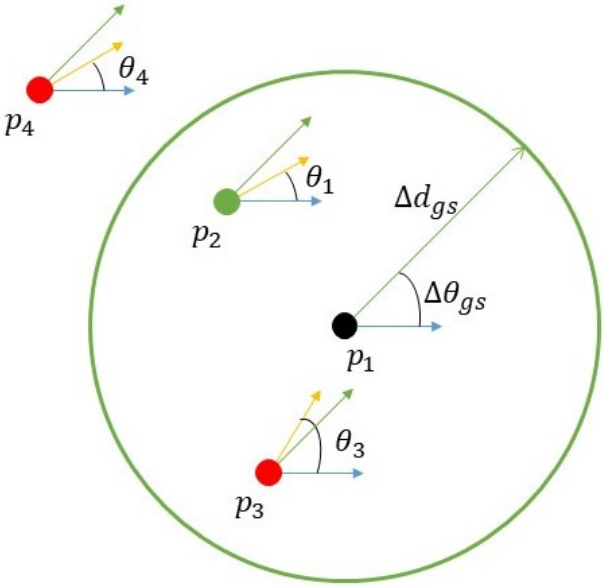


Figura 3.13: Las configuraciones p_1 y p_2 se consideran iguales. En cambio p_3 y p_4 son distintas a p_1. Las flechas amarillas son la inclinación de cada nodo y las verdes la tolerancia.

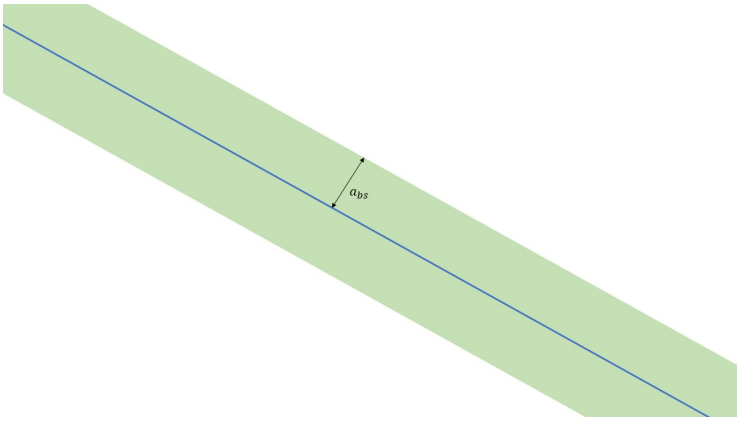
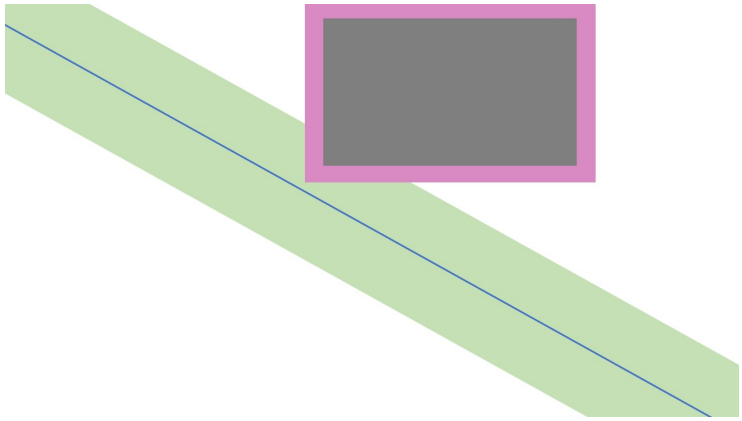
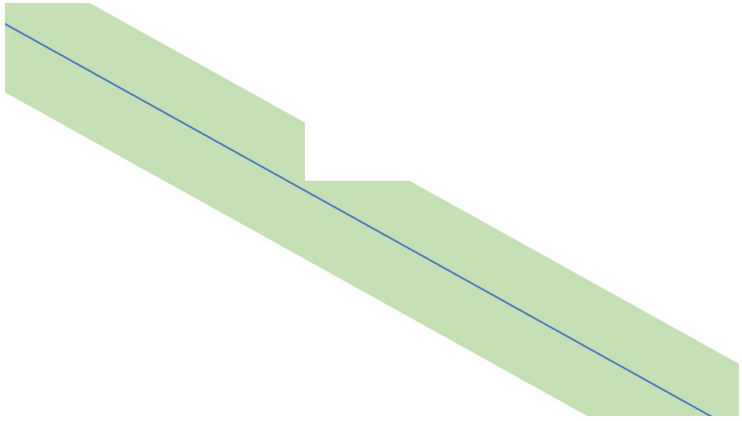


Figura 3.14: Banda de seguridad. La línea azul es el camino planificado y la banda de seguridad está representada por el área pintada de verde.



(a) En este tramo, la banda está superpuesta por un obstáculo aumentado (rectángulo gris con marco violeta) por lo que se debe quitar la intersección entre ambos.



(b) Área resultante al quitar la intersección.

Figura 3.15: La banda luego de eliminar de ella el área del obstáculo.

donde dada la posición p_i, se elimina el movimiento m_j, ya que al aplicarlo (camino amarillo), el destino p_{d1} queda fuera de la banda de seguridad. Respecto al movimiento m_k, también se elimina porque parte del recorrido desde la posición inicial p_i a la posición destino p_{d2} se realiza fuera de la banda de seguridad.

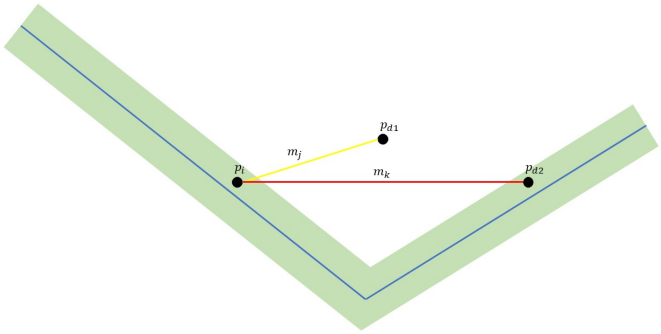


Figura 3.16: Posición inicial p_i, destinos p_{d1} y p_{d2}, que son el resultado de aplicar los movimientos m_j y m_k.

Proximidad al destino

Luego de procesar nivel n del árbol, un enfoque seria filtrar todos los nodos del nivel $n+1$, exceptuando aquel nodo más próximo al siguiente punto de destino p_d. Sin embargo, si bien este enfoque disminuye considerablemente la cantidad de nodos a procesar y deja al robot lo más cerca posible del destino, es probable que se pierdan mejores soluciones para llegar a p_{fin} o que al realizar la siguiente iteración no se encuentre solución alguna. Teniendo en cuenta esto y queriendo filtrar aquellos nodos del nivel $n+1$ en el cual el robot avanzó poco (que pueden resultar en ramas largas donde el robot solo gira) es que se implementó este filtro.

Sea p_p el nodo del nivel $n+1$ que deja al robot en la posición más próxima a p_d. El filtro por proximidad al destino va a descartar a todos los nodos del nivel $n+1$ que estén por fuera de un círculo centrado en p_d y radio r. El radio del círculo está definido en función de la distancia de p_p a p_d y los parámetros α y Δd. Más específicamente:

$$r = dist(p_p, p_d) + \alpha * \Delta d \tag{3.8}$$

En la figura 3.17 se puede ver que las líneas rojas marcan el ancho $\alpha\Delta d$ del área del filtro de proximidad. Tomando como base la posición más próxima al destino (p_p), las posiciones que estén dentro del área del filtro serán

tomados como válidos, lo que queden fuera serán eliminados.

Figura 3.17: Aplicación de filtro de proximidad al destino, tomando como base la posición más cercana p_p. Todos los nodos por fuera de la franja formada por las curvas rojas se descartan.

Si bien la aplicación de este filtro permite descartar una gran cantidad de nodos por iteración, es necesario ajustar el parámetro α para obtener un valor que permita que los tiempos de procesamiento del algoritmo sean razonables sin perder aquellos movimientos que por estar rezagados, no se tienen en cuenta, cuando en realidad son parte de trayectorias posibles.

Proximidad al camino planificado

Con el objetivo de usar al camino planificado como guía es que se implementó un filtro que descarte los nodos que se alejan una cierta distancia del mismo. Esta distancia se calcula luego de procesar cada nivel n del árbol en función de la distancia del nodo p_p del nivel $n+1$ que deje al robot más cerca al camino. Si d_{min} es la distancia de p_p al camino, se eliminan todos los nodos del nivel $n+1$ que estén a una distancia mayor a $d_{min} + \frac{a_{bs}}{\beta}$. Recordar que a_{bs} es el ancho de la banda y que β es un entero pasado como parámetro.

En la figura 3.18 se pueden observar todos los nodos del nivel $n+1$ representados como puntos rojos, salvo p_p. Todos los nodos que no estén entre las dos rectas rojas se eliminan.

Este filtro es parecido a la banda de seguridad antes mencionada. La diferencia radica en que, para el filtro de proximidad al camino, el ancho del área valida varía según el nivel del árbol, mientras que el ancho de la banda de seguridad (a_{bs}) permanece constante. La forma en que se aplica permite tener la flexibilidad de movimiento dado por la banda de seguridad, para luego quedarse con las posiciones más cercanas al camino planificado.

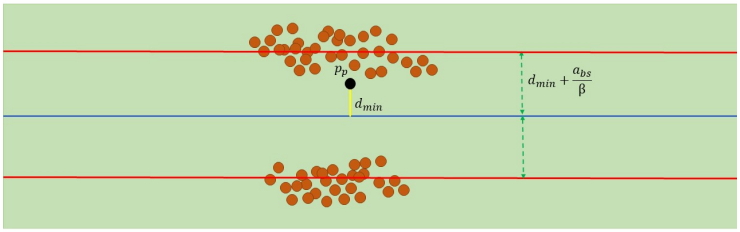


Figura 3.18: Aplicación de filtro de proximidad al camino, tomando como base la posición más cercana p_p.

Hay que tener en cuenta que cuanto más estricto se es con el parámetro β, el robot tenderá a moverse cerca de la línea de camino planificado, perdiendo la flexibilidad de movimiento que permite la banda de seguridad y corriendo riesgos de no superar curvas. Por este motivo, el valor que tome el parámetro β debe permitir filtrar la mayor cantidad de movimientos sin perder la flexibilidad de movimiento brindada por la banda de seguridad.

Frecuencia de aplicación

La aplicación de los filtros se divide en aquellos que se aplican en cada nivel del árbol y aquellos que se aplican cada cierta cantidad de niveles. Los que se aplican en cada nivel del árbol son el filtro de Visitados y el de Banda de seguridad. En cambio, los filtros de proximidad al destino y de proximidad al camino planificado son los que se aplican cada cierta cantidad de niveles.

La frecuencia de aplicación de los filtros está dada por el parámetro de entrada al algoritmo γ (ver sección 3.4.2), que define cada cuantos niveles del árbol se aplican los filtros antes mencionados. Con este parámetro se permite que el árbol crezca libremente cada γ niveles. Esto permite recuperar algo de flexibilidad perdida durante la aplicación de los filtros.

Hay que tener en cuenta que no aplicar filtros produce un crecimiento en el costo en procesamiento. Por este motivo se debe definir el parámetro γ que permita ganar libertad de movimientos y a la vez acotar el crecimiento desmedido del árbol de búsqueda.

Luego de aplicar los controles en cada nivel del árbol, los filtros se aplican en el siguiente orden:

1. Filtro de visitado.

2. Filtro de banda de seguridad.

3. Cada cierta cantidad de niveles, definido por el parámetro γ:

   a) Filtro de proximidad al destino.

   b) Filtro de proximidad al camino planificado.

3.4.6. Δt variable

El algoritmo original de Grid Search cuenta con un Δt constante en cada nivel del árbol de búsqueda. En esta variante la elección de este Δt deberá ser tal que permita realizar giros pequeños que logren superar curvas cerradas, pero a su vez lo suficientemente grande para que en rectas largas no se generen demasiados niveles del árbol.

Del análisis que se realizó, se concluyó que es mejor contar con un Δt variable por niveles del árbol. Es así que se decidió contar con un Δt pequeño para planificar mejor las curvas y un Δt mayor para planificar en menos niveles las rectas. Para esto se creó la función f_t, que retorna el Δt a aplicar al siguiente nivel del árbol.

A continuación, se detallará la definición de robot orientado al destino y la función f_t utilizada para obtener el Δt a aplicar.

Definición de robot orientado al destino

Se considera que el robot está orientado al destino p_d si el vector definido por el centro del robot y su orientación intersecta la bola de centro p_d y radio Δd. En caso contrario se toma el ángulo que se forma entre la orientación y el punto de destino.

En la figura 3.19, en la posición p_1, se considera al robot como orientado al destino p_d, ya el vector de orientación (negro) intersecta la bola de centro p_d y radio Δd. En la posición p_2 el robot no está orientado, ya que el vector de orientación no intersecta la bola de centro p_d y radio Δd. Por lo tanto, para la posición p_2, la inclinación será θ_2 al destino p_d.

Con esta definición se logra una mayor flexibilidad que solo considerar como orientado aquellas posiciones donde el vector de orientación está alineado con punto de destino p_d. La inclinación será importante para determinar el Δt a aplicar por nivel, como se verá a continuación.

Función Δt

Estando en el nivel i del árbol de búsqueda, se aplica la función f_t, que se muestra en la ecuación (3.9), para determinar el Δt a aplicar en el siguiente nivel $(i + 1)$ del árbol.

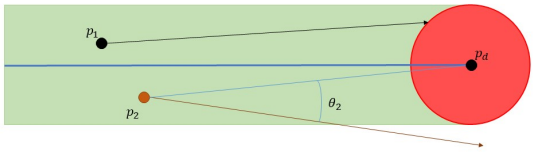


Figura 3.19: Posición p_1 orientado y posición p_2 con inclinación θ

Del conjunto de posiciones del nivel i se toma la distancia d_{min} que es la menor distancia al destino p_d y la media de las inclinaciones θ_m. Con estos valores se ejecuta la función f_t para determinar el Δt del nivel $(i+1)$.

$$f_t(d_{min}, \theta_m) = \begin{cases} \Delta t_{min} & si \quad |\theta_m| > \frac{\pi}{3} \\ \frac{f_1(d_{min})}{f_2(\theta_m)} & si \quad |\theta_m| \leq \frac{\pi}{3} \end{cases} \tag{3.9}$$

Siendo f_1 la función dependiente de la distancia al destino y f_2 la función dependiente de la inclinación del robot al destino. Ambas se describen a continuación.

Función f_1

El objetivo de la función f_1 será retornar el tiempo que le lleve al robot recorrer la distancia desde la posición actual al siguiente destino. Con ese objetivo la función f_1 recibe como entrada la distancia del robot al destino y la misma está dada por la ecuación (3.10).

$$f_1(d) = md \tag{3.10}$$

Siendo d la distancia en metros y m pendiente de la ecuación. La distancia d_{min} que se utiliza es la mínima del conjunto de posiciones del nivel i del árbol de búsqueda. Para el valor de m se toma la fracción dependiente de la velocidad máxima lineal del robot, $\frac{1}{v_{max}}$. Por lo tanto la ecuación f_1 está dada por 3.11.

$$f_1(d_{min}) = \frac{1}{v_{max}} d_{min} \tag{3.11}$$

La gráfica que describe la función f_1 se puede ver en la figura 3.20.

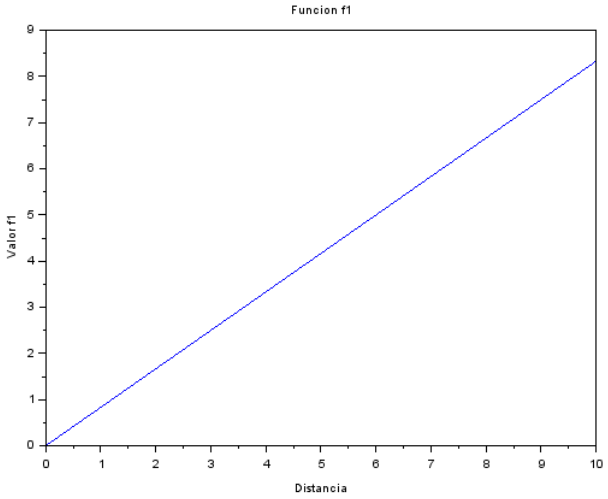


Figura 3.20: Gráfica f_1. Cuanto mayor es la distancia al destino, mayor será el valor de f_1.

Función f_2

El objetivo de la función f_2 es que sea el factor de disminución de la función f_1 dependiente de la inclinación media del nivel i del árbol. Si la media es igual a cero, la función f_2 es igual a uno, en caso contrario cuanto mayor la inclinación, más grande será el valor de f_2. Con ese objetivo, la función f_2 recibe como entrada la media de la inclinación del nivel del árbol y la misma tiene la ecuación (3.12).

$$f_2(\theta) = \begin{cases} 1 + |p \tanh(\theta \frac{180}{\pi})| & si \quad |\theta| \leq \frac{\pi}{2} \\ 1 & si \quad |\theta| > \frac{\pi}{2} \end{cases} \tag{3.12}$$

Siendo θ la inclinación en radianes y p un valor de ponderación de la $tanh$. La inclinación que se va a utilizar es θ_{med}, que será la mediana del conjunto de todas las inclinaciones del nivel del árbol en la que se encuentra el algoritmo.

Para el cálculo del valor p se toma la posición inicial del robot. Con esta posición se calcula la distancia del robot al destino (d_{fin}) y el tiempo que demora en recorrer dicha distancia (Δt_{max}) a velocidad máxima lineal (v_{max}) (3.13).

$$\Delta t_{max} = \frac{d_{fin}}{v_{max}} \tag{3.13}$$

Con el Δt_{max} calculado y el Δt_{min} se obtiene el p con la ecuación (3.14).

$$p = \frac{|\frac{\Delta t_{max}}{\Delta t_{min}} - 1|}{\tanh(\frac{\pi}{6})} \tag{3.14}$$

La gráfica que describe la función f_2 se puede apreciar en la figura 3.21. Por otro lado, la función de $f = \frac{f_1}{f_2}$ se encuentra graficada en la figura 3.22.

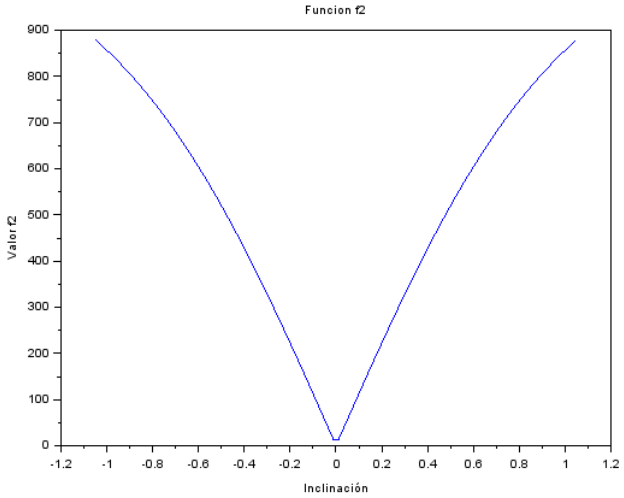


Figura 3.21: Gráfica f_2

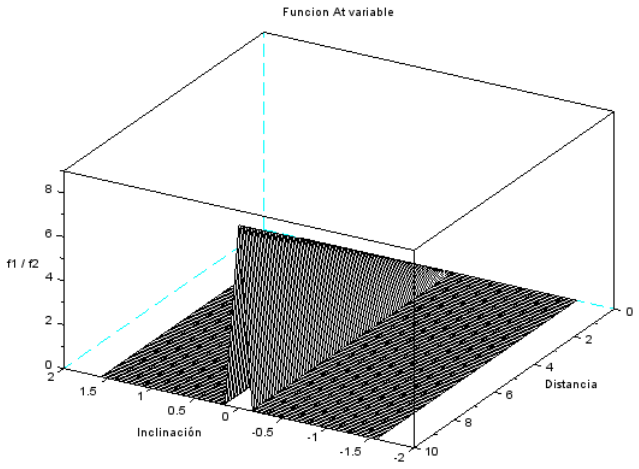


Figura 3.22: Gráfica Δt variable

3.5. Emisor de comandos

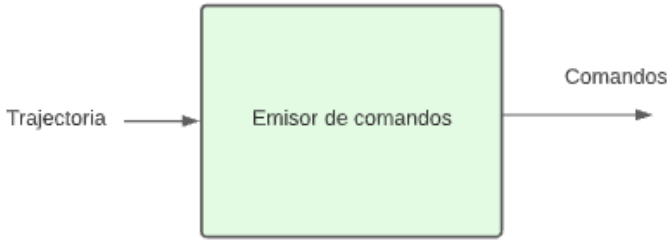


Figura 3.23: El emisor de comandos recibe una trayectoria (o secuencia de controles con sus respectivos Δt los transforma para que el formato sea entendido por el simulador y se la envía a este en forma de comandos.

Es necesario validar que el control planificado es factible en una simulación. Con este objetivo se creó el módulo de emisor de comandos, el cual será el puente entre el programa y el simulador. Esta toma la trayectoria planificada, la transforma en comandos que el robot debe aplicar y envía los mismos al simulador.

Para este módulo se optó por una implementación que envía la lista de

comandos completa a través de un socket TCP. Esto implica que el simulador debe de estar ejecutándose al momento de enviar los comandos para que se pueda realizar la conexión TCP.

3.6. Simulación

El simulador seleccionado para esta instancia es el Webots. Este software nos provee las herramientas necesarias para simular un robot móvil y su entorno. No es el propósito de esta sección entrar en detalle de cómo funciona Webots pero si se describen los componentes más relevantes para un mejor entendimiento del lector. Si se desea conocer más sobre Webots se puede acceder a su sitio web [6]. Una simulación Webots está compuesta por un mundo, uno o más controladores y un plugin opcional para modificar el comportamiento físico de Webots.

Un mundo Webots es una la descripción 3D de las propiedades del robot (o varios) y su entorno. En él se especifican para cada objeto su posición, orientación, geometría, apariencia (como el color o el brillo), sus propiedades físicas, etc. El mundo no contiene el código de los controladores, pero si se especifica el nombre del controlador que requiere cada robot.

Un controlador es un programa que describe el comportamiento del robot. Los controladores son capaces de controlar los motores del robot, ya sea especificando su aceleración o velocidad angular. A su vez, son capaces de leer los valores de los sensores del robot para un mejor control del mismo. Cuando se inicia la simulación, se comienza con la ejecución de los controladores, cada uno ejecutándose como un proceso independiente. Webots permite implementar un tipo especial de controlador llamado Supervisor [7]. Estos controladores pueden ejecutar operaciones que no pueden ser llevadas a cabo por un robot móvil real. Normalmente estas acciones son llevadas a cabo por una persona. Dentro de estas operaciones se encuentran mover el robot a una posición aleatoria del mapa, obtener la posición exacta del robot dentro del marco global, etc.

Con estas simulaciones nos permitirán validar que el Khepera III es capaz de recorrer el camino utilizando la trayectoria planificada. En esta sección se toma el entorno visto en la figura 3.3a como ejemplo para detallar las características de la implementación.

3.6.1. Implementación

Para cada entorno de prueba se implementó su análogo en Webots. En la figura 3.24 se ve el modelado de uno de ellos (3.3a).

Siguiendo la misma línea, se implantaron dos controladores. Uno correspondiente al algoritmo de Giro y Avanzo y otro para el Grid Search. Por simplicidad, los controladores son de la clase Supervisor [7]. De esta forma,

Figura 3.24: Representación en Webots del mapa mostrado en la figura 3.3a. En la parte superior izquierda se encuentra el Khepera III en su posición inicial.

se puede obtener fácilmente la ubicación del robot en cada ciclo de ejecución y así poder hacer la comparación entre la trayectoria computada y la realizada. En ambos casos, lo primero que se hace es recibir los comandos vía TCP. Luego, se aplican en orden cada comando. La ejecución de la simulación termina cuando ya no hay más comandos por consumir. Al final de la misma, se guarda en un archivo la trayectoria realizada por el robot.

Ambos controladores tienen un ciclo de ejecución de 16 ms (time step). Esto quiere decir que cada 16 ms el controlador considerará que acción realizar sobre el robot. En particular, estos controladores verifican si deben aplicar un nuevo comando o seguir aplicando el mismo hasta alcanzar el tiempo de aplicación deseado.

El controlador del Grid Search no posee ningún tipo de control sobre el movimiento del robot. Esto quiere decir que no se verifica que el robot haya realmente alcanzado la orientación o posición deseada luego de la aplicación de un comando.

Por otro lado, como se consideró la aceleración infinita, se tuvo que mitigar el error introducido por la inercia para el controlador del el Giro y Avanzo. Es por ello que el controlador posee una pequeña corrección en el movimiento. Cada comando se ejecutará más tiempo en función de cuánto se tardó en alcanzar la velocidad deseada.

Capítulo 4

Análisis experimental

Índice

En este capítulo se detallarán las diferentes pruebas que se realizaron para validar que la adaptación del Grid Search realmente mejora al algoritmo. En particular se analizará el impacto que tienen el Δt variable, los filtros, y la cantidad de controladores sobre el algoritmo, teniendo como métrica el tiempo de procesamiento y la calidad de la solución.

Por otro lado, se hace una comparativa entre planificar el control usando la adaptación al Grid Search y la estrategia básica de Giro y Avanzo. Para este análisis se realizan pruebas en tres escenarios diferentes. El objetivo de estas pruebas es saber cuanto tiempo lleva encontrar la solución utilizando una u otra estrategia y cual de las dos retorna mejores soluciones teniendo en cuenta el tiempo que le llevaría al robot ejecutar el plan.

Por último, en diferentes escenarios se simularán las trayectorias planificadas utilizando la solución propuesta y el Giro y Avanzo. Con estas pruebas se busca validar que la propuesta es posible de llevarse a cabo en un entorno simulado.

4.1. Comparativa Δt variable vs Δt fijo

En el capítulo anterior (3.4.6) se presentó la implementación del Δt variable para conseguir que el algoritmo procese menos niveles del árbol de grid search. En esta sección se muestran las pruebas realizadas y los resultados obtenidos que permitieron probar que emplear un Δt diferente por nivel es mejor que utilizar el mismo Δt en cada nivel.

Para estas pruebas se ejecutó el mismo entorno con el Δt variable por nivel y luego con el Δt fijo ($0{,}0154$ segundos) por nivel. Estas pruebas se realizaron dejando fijo la cantidad de controladores y los los parámetros de configuración de los filtros aplicados (cuadro 4.1).

Movimientos	Prox Dest	Prox camino	Filtrar	V_{max}
$\emptyset$	2	2	2	$1m/s$

Cuadro 4.1: Parámetros de entrada comparativa Δt fijo vs Δt variable.

En la figura 4.1 se puede observar la diferencia de las soluciones obtenidas con Δt fijo y Δt variable.

Como se puede observar en la figura 4.2, la solución obtenida con el Δt fijo es mejor, dado que permite planificar las curvas de antemano, haciendo que a la salida de la misma el robot se direccione mejor para llegar la siguiente posición. Además, se puede observar que al contar con un Δt fijo, las correcciones de dirección son menos bruscas. Esta mejor solución se debe a que entre niveles el Δt aplicado es el Δt_{min}, por lo que permite hacer correcciones de dirección con mayor precisión.

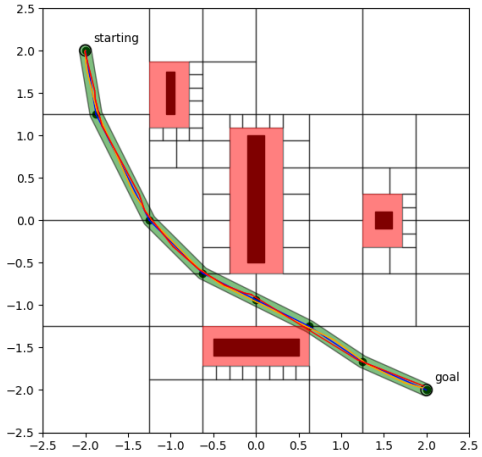


Figura 4.1: Comparativa de caminos obtenidos con Δt fijo (camino amarillo) y Δt variable (camino rojo). La línea azul es el camino planificado.

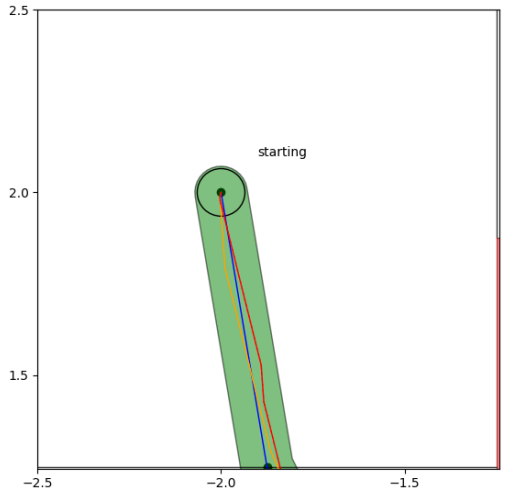
Como se puede ver en el cuadro 4.2, si bien se obtiene un tiempo de recorrido menos con un Δt fijo, el costo de procesamiento es bastante mayor que cuando se usa Δt variable. Por otro, con un Δt variable no solo la profundidad del árbol de búsqueda es menor sino que en cada nivel del árbol hay menos nodos para procesar haciendo que el tiempo de búsqueda sea considerablemente menor.

-	Niveles procesados	Nodos procesados	Tiempo de procesamiento	Tiempo de solución
Δt fijo	675	139047	$15{,}6022s$	$6{,}03679s$
Δt variable	303	24801	$1{,}68936s$	$6{,}12682s$

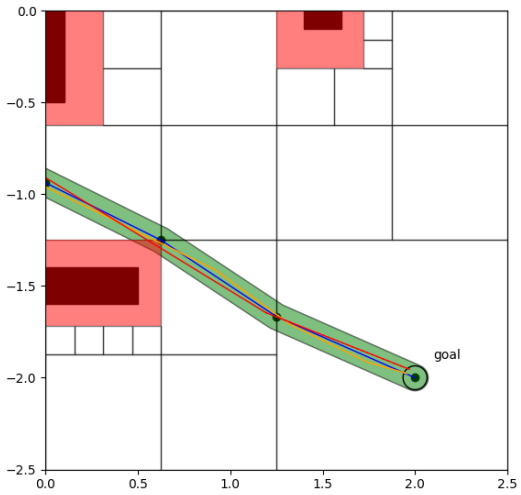
Cuadro 4.2: Comparativa de resultados Δt fijo vs Δt variable.

Al aplicar Δt mayores, las distancias y giros son mayores. Esto hace que al aplicar los filtros de poda, una mayor cantidad de nodos sean descartados del árbol de búsqueda. Como se puede observar en la figura 4.3, para los primeros niveles del árbol de búsqueda, en el caso del Δt variable en el nivel treinta la cantidad de posiciones a procesar es notoriamente menor que para el Δt fijo.

Para procesar los primeros tres puntos del camino planificado fueron necesarios ciento cuarenta niveles para un Δt fijo (fig. 4.4), contra los ochenta

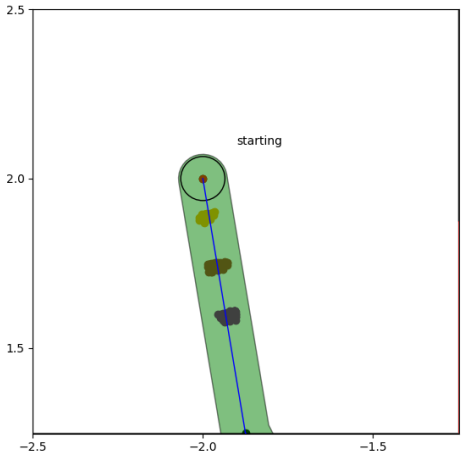


(a) Acercamiento a comparativa

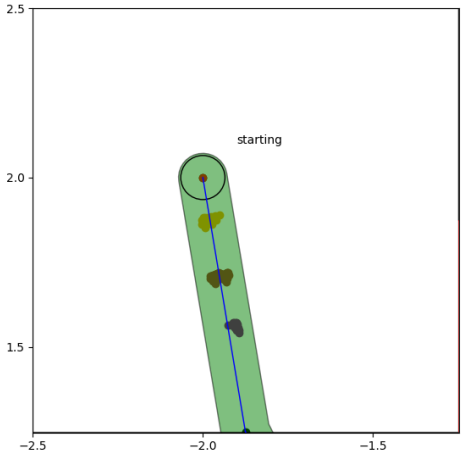


(b) Acercamiento a comparativa

Figura 4.2: Acercamiento a comparativa de caminos obtenidos con Δt fijo (camino amarillo) y Δt variable (camino rojo). La línea azul es el camino planificado.



(a) Δt fijo.



(b) Δt variable.

Figura 4.3: Niveles 0 (rojo), 10 (amarillo), 20 (marrón) y 30 (violeta) del árbol de búsqueda para Δt fijo y Δt variable.

y seis niveles que se procesaron para el Δt variable (fig. 4.5).

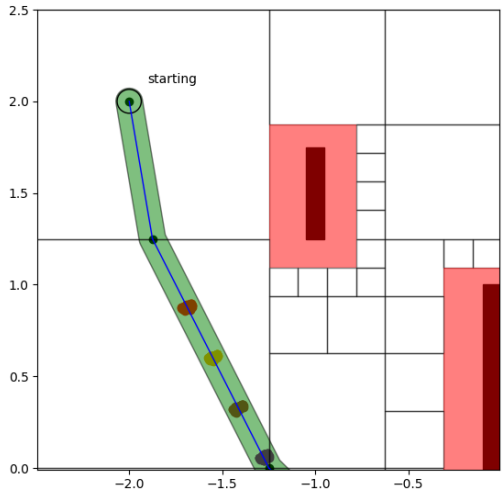


Figura 4.4: Niveles 80 (rojo), 100 (amarillo), 120 (marrón) y 140 (violeta) del árbol de búsqueda para Δt fijo.

En la figura 4.6 se puede observar como la solución obtenida con el Δt variable permite planificar tramos más largos con menos niveles. Esto se debe a la posibilidad de emplear en ese nivel un Δt mayor al fijo. Esta es la ventaja de usar Δt variable sobre el Δt fijo.

Si bien contar con un Δt variable es beneficioso para el tiempo de procesamiento, se pierde la precisión en los movimientos que se obtiene al contar con un Δt fijo. La mejora en tiempo de recorrido obtenida al contar con un Δt fijo no compensa el aumento en el tiempo de procesamiento requerido.

Por este motivo es necesario ajustar los valores de configuración de filtros y cantidad de controladores para recuperar dicha flexibilidad. A continuación, presentaremos las pruebas realizadas con los filtros y la cantidad de controladores para recuperar dicha flexibilidad perdida con un Δt variable.

4.2. Aplicación de filtros

En el capítulo anterior (3.4.5) se presentaron los filtros que se implementaron para conseguir que el algoritmo procese menos movimientos por nivel del árbol de grid search. A continuación, se muestran las pruebas rea-

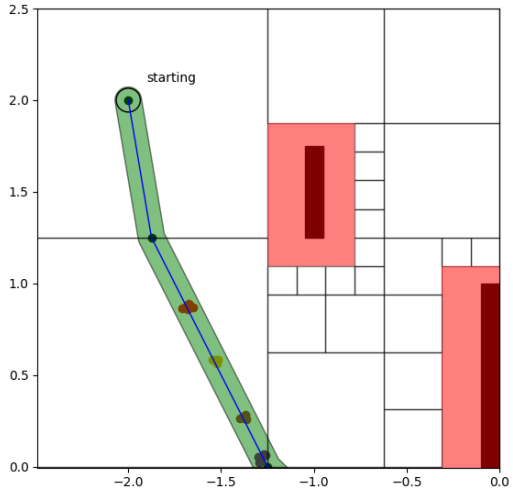


Figura 4.5: Niveles 60 (rojo), 70 (amarillo), 80 (marrón) y 86 (violeta) del árbol de búsqueda para Δt variable.

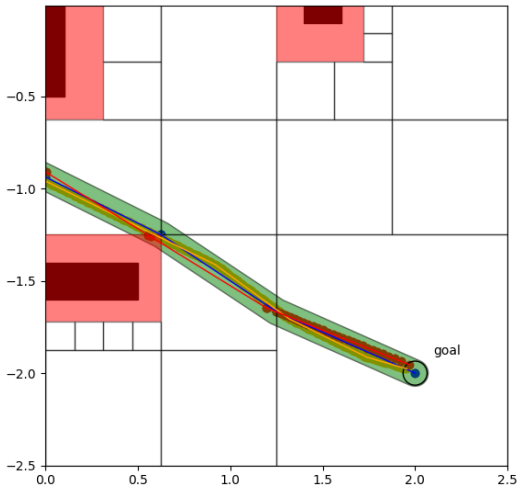


Figura 4.6: Aproximación a solución obtenida con Δt fijo (camino amarillo) y con Δt variable (camino rojo).

lizadas y los resultados obtenidos. Para estas pruebas se dejó fijo los demás parámetros (cuadro 4.3), solo variando los valores de filtros aplicados.

-	Δt_{min}	Movimientos	Prox Dest	Prox camino	Frec	V_{max}
Filtros laxos	$0{,}0154s$	$\emptyset$	3	1	3	$1m/s$
Filtros estrictos	$0{,}0154s$	$\emptyset$	1	4	3	$1m/s$

Cuadro 4.3: Parámetros de entrada comparativa frecuencia de aplicación de filtros.

Es intuitivo concluir que el filtrado tiene la ventaja de conseguir que el procesamiento sea más ágil, pero en contrapartida, puede generar que si son muy restrictivos no se encuentre solución. En la figura 4.7 se puede apreciar la solución obtenida para filtros más estrictos (camino rojo) contra la filtros más laxos (camino amarillo).

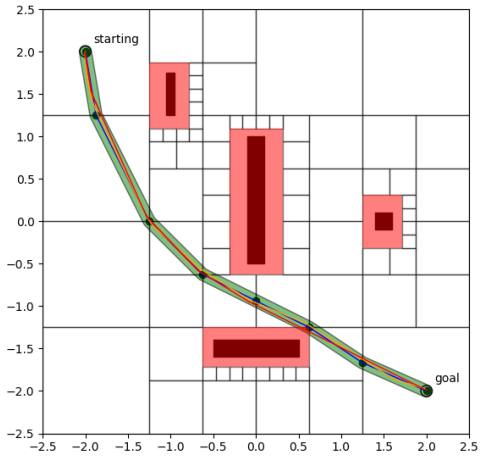
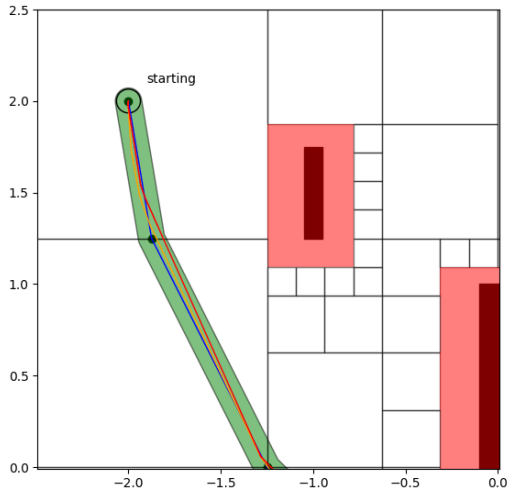
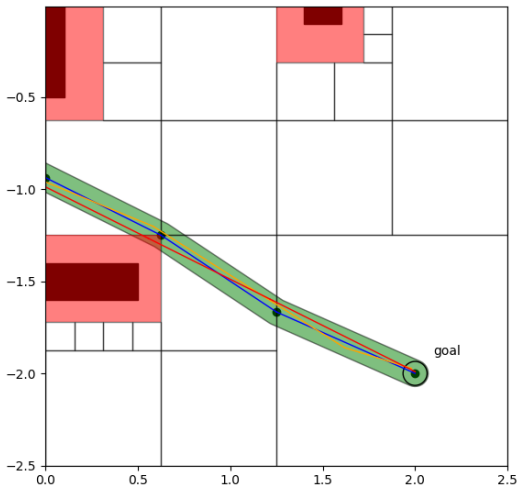


Figura 4.7: Comparativa de caminos obtenidos con filtros laxos (amarillo) y filtros restrictivos (rojo). La línea azul es el camino planificado.

Como se puede apreciar en la figura 4.7, la solución obtenida para filtros más laxos es mejor en tiempo de recorrido (cuadro 4.4). Esto porque permite al planificador, tener un mayor margen para dejar crecer el árbol de búsqueda, antes de podarlo. Permitiendo que se planifiquen las tomas de curva antes (figura 4.8a) y de tal forma que a la salida de la misma quede orientado a la siguiente posición. Además, cuando se tiene filtros más laxos,



(a) Acercamiento a comparativa



(b) Acercamiento a comparativa

Figura 4.8: Acercamiento a comparativa de caminos obtenidos con filtros laxos (amarillo) y filtros restrictivos (rojo).La línea azul es el camino planificado.

se permite que el robot se mueva en toda la banda de seguridad (figura 4.8b). Esto permite posicionar mejor al robot para tomar las curvas.

-	Niveles	Nodos	Procesamiento	Tiempo de solución
Filtros laxos	373	336096	$29{,}9993s$	$6{,}10928s$
Filtros estrictos	142	2964	$0{,}260511s$	$6{,}13358s$

Cuadro 4.4: Comparativa de resultados filtros laxos vs filtros estrictos.

Por otro lado, también hay que tener en cuenta el costo de procesar una mayor cantidad de movimientos al tener filtros más laxos (cuadro 4.4). Para el caso de la figura 4.7 el tiempo de procesamiento con filtros laxos (camino rojo) fue mayor que el tiempo de procesamiento para filtros estrictos (camino amarillo). La mejora en el tiempo de recorrido conseguida por tener filtros más laxos no justifica el aumento en el tiempo de procesamiento. Como ejemplo, en las soluciones mostradas en la figura 4.7, la mejora conseguida fue del orden de las centésimas de segundo, por lo que el tiempo extra de procesamiento no compensaba la mejora en la solución obtenida.

En conclusión, es necesario balancear la flexibilidad de los filtros a aplicar, contra el tiempo de procesamiento requerido. Es necesario ajustar los mismos, para obtener una mejor solución en tiempo de recorrido, pero sin aumentar el tiempo de procesamiento, de tal manera que la suma de los dos siga siendo la mejor posible.

Hasta acá se observó el efecto de aplicar los tres parámetros de ajuste (proximidad al destino, proximidad al camino y frecuencia de aplicación) de los filtros sobre la solución obtenida. A continuación, veremos por separado estos filtros y su efecto sobre la solución obtenida.

4.2.1. Filtro de proximidad al destino

En esta sección se verá cómo afecta la aplicación más estricta o más laxa del filtro de proximidad al destino. Para esto, se dejaron fijo los otros parámetros de configuración de los filtros de proximidad al camino y frecuencia de aplicación, solo modificando el parámetro que afecta al filtro de proximidad al destino (cuadro 4.5). Además, se tomó el conjunto de controladores $\{(v_{max}, v_{max}), (0, v_{max}), (v_{max}, 0)\}$.

Δt_{min}	Movimientos	Prox camino	Frec	V_{max}
$0{,}0154s$	$\{\emptyset\}$	1	1	$1m/s$

Cuadro 4.5: Parámetros de entrada comparativa filtro proximidad al destino.

La figura 4.9 muestra la diferencia en la solución obtenida con un filtro de al destino laxo (camino amarillo), contra un filtro de proximidad al destino restrictivo (camino rojo).

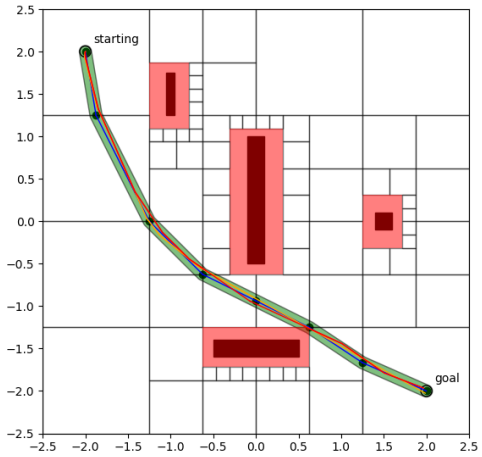


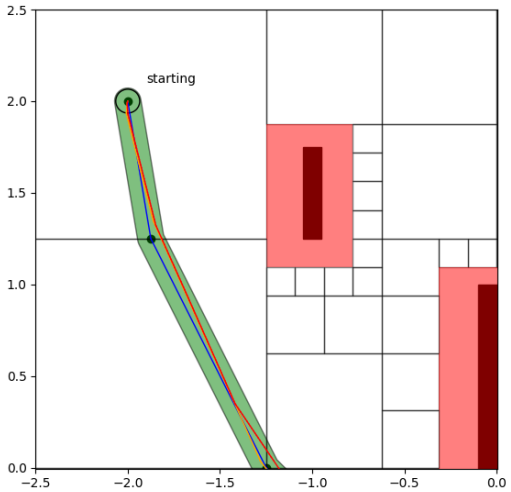
Figura 4.9: Comparativa de caminos obtenidos con filtro de proximidad al destino laxo (amarillo) y restrictivo (rojo). La línea azul es el camino planificado.

Como se puede observar en la figura 4.10, la solución obtenida con el filtro de proximidad al destino laxo, permite planificar las curvas antes, dado que no se prioriza el llegar a la siguiente posición, como si se hace con un filtro más estricto.

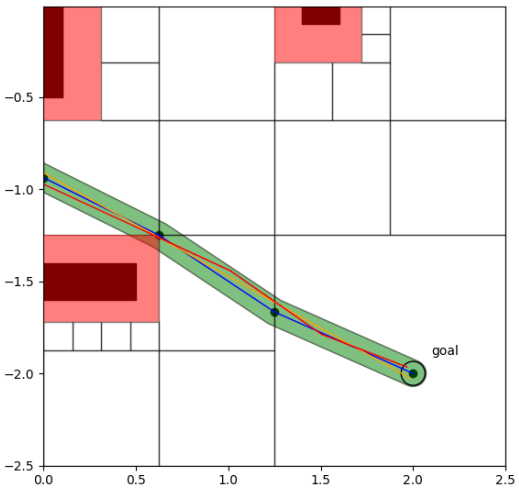
Prox dest	Niveles	Nodos	Procesamiento	Tiempo de solución
2	373	336096	$29{,}9993s$	$6{,}10928s$
6	124	3179673	$0{,}260511s$	$6{,}13358s$

Cuadro 4.6: Comparativa de resultados filtro proximidad al destino laxo vs estricto.

Hay que tener en cuenta que la solución con filtros más laxos fue más costosa en tiempo de procesamiento que la de filtro más estrictos (4.6). Esto se debe a que al ser más laxo existen más movimientos para procesar por nivel del árbol. Como se puede ver en la figura 4.11, los primeros niveles del árbol de búsqueda cuentan con menos movimientos cuando se aplican filtros más estrictos.

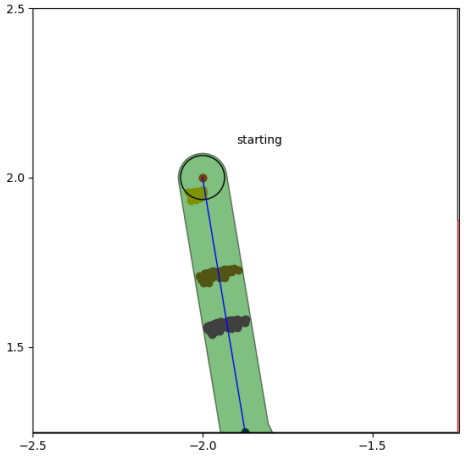


(a) Acercamiento a comparativa

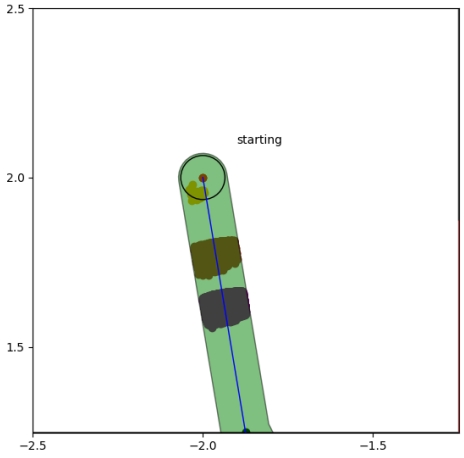


(b) Acercamiento a comparativa

Figura 4.10: Acercamiento a comparativa de caminos obtenidos con filtro de proximidad al destino laxo (amarillo) y restrictivo (rojo). La línea azul es el camino planificado.



(a) Filtro estricto de proximidad al destino.



(b) Filtro laxo de proximidad al destino.

Figura 4.11: Diferencia entre aplicar el filtro de proximidad al destino de forma más o menos laxa. En ambas figuras se se muestran los niveles 0 (rojo), 5 (amarillo), 20 (marrón) y 30 (violeta) del árbol de búsqueda. Si se es más laxo la cantidad de puntos por nivel será cada vez mayor.

Como se puede ver en el cuadro 4.6, el menor tiempo de procesamiento se logra ya que se deben procesar menos movimientos por nivel del árbol de búsqueda y son menos los niveles del árbol a procesar para encontrar la solución. En la figura 4.12, sue puede observar, para el caso de filtros estrictos fue necesario procesar setenta y ocho niveles para encontrar la solución en las primeras tres posiciones del camino planificado, contra los ciento veinticinco niveles necesarios procesar con un filtro laxo (fig. 4.13).

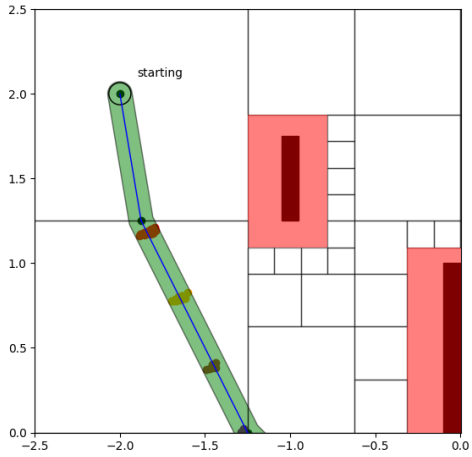


Figura 4.12: Niveles 50 (rojo), 60 (amarillo), 70 (marrón) y 78 (violeta) del árbol de búsqueda para un filtro estricto de proximidad al destino.

Para entender por qué es necesario procesar menos niveles del árbol cuando se tiene un filtro más estricto, es necesario recordar cómo funciona el Δt variable por nivel. Uno de los factores que afecta fuertemente el cálculo del Δt es la inclinación media de todos los puntos del nivel. Al contar con un filtro más estricto se priorizan los movimientos que acercan al robot a la siguiente posición del camino, haciendo que el conjunto de posiciones por nivel tenga una inclinación orientada al destino y por lo tanto una inclinación media menor. Esto produce que para filtros más estrictos se pueden aplicar Δt mayores en las rectas.

En contrapartida, como se puede ver en la figura 4.14, para el mismo nivel del árbol, la heterogeneidad de las orientaciones para un filtro laxo hace que la media sea mayor. Esta mayor inclinación media produce que el Δt a aplicar por nivel sea menor y por lo tanto sean necesarios más niveles para procesar la misma recta.

Una menor inclinación media hace que al planificar se pueda optar por un Δt mayor, por lo tanto, entre un nivel y otro las distancias recorridas

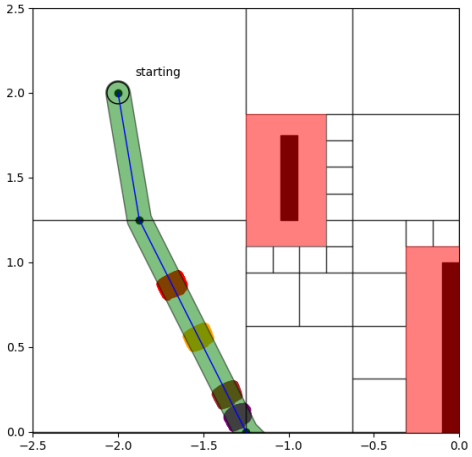


Figura 4.13: Niveles 80 (rojo), 100 (amarillo), 120 (marrón) y 125 (violeta) del árbol de búsqueda para un filtro estricto de proximidad al destino.

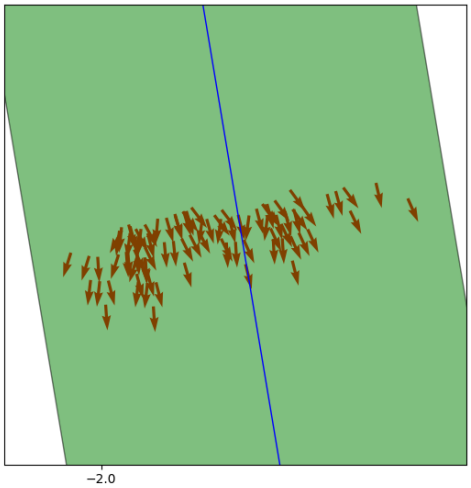
son mayores. Este es el motivo por el cual para encontrar una solución es necesario procesar menos niveles del árbol de búsqueda cuando se utiliza un filtro estricto.

De lo expuesto anteriormente, la ventaja de contar con un filtro estricto de proximidad al destino es poder planificar rectas largas en menor cantidad de niveles del árbol de búsqueda. En contrapartida, para la planificación de curvas es mejor un filtro laxo de proximidad al destino. Esta mejora en la planificación curvas, se debe a que se cuenta con mayor cantidad de posiciones dentro de la banda del filtro que permite planificar la toma de la curva antes.

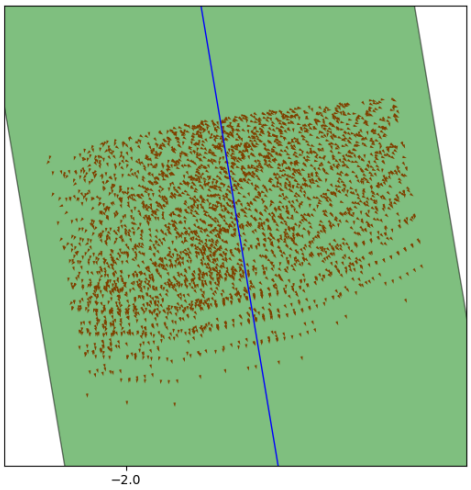
Por lo tanto, es necesario ajustar el filtro de proximidad al destino para procesar la menor cantidad de movimientos por nivel, beneficiando la planificación de las rectas pero permitiendo la flexibilidad que permita planificar correctamente las curvas.

4.2.2. Filtro de proximidad al camino

En esta sección se verá cómo afecta la aplicación más estricta o más laxa del filtro de proximidad al camino. Para esto, se dejaron fijo los otros parámetros de configuración de los filtros de proximidad al destino y frecuencia de aplicación, solo se modificó el parámetro que afecta al filtro de proximidad al camino (cuadro 4.7). Además, se tomó el conjunto de controladores $\{(v_{max}, v_{max}), (0, v_{max}), (v_{max}, 0)\}$.



(a) Filtro estricto.



(b) Filtro laxo.

Figura 4.14: Vectores de dirección para el nivel 20 del árbol de búsqueda para filtros estricto y laxo.

Δt_{min}	Movimientos	Prox dest	Frec	V_{max}
$0{,}0154s$	$\emptyset$	4	1	$1m/s$

Cuadro 4.7: Parámetros de entrada comparativa filtro proximidad al destino.

La figura 4.15 muestra la diferencia entre la solución obtenida con un filtro estricto de proximidad al camino (camino roja), y la solución obtenida con un filtro laxo de proximidad al camino (camino amarilla).

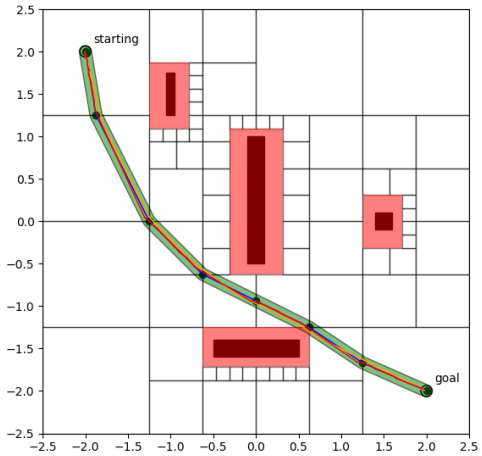
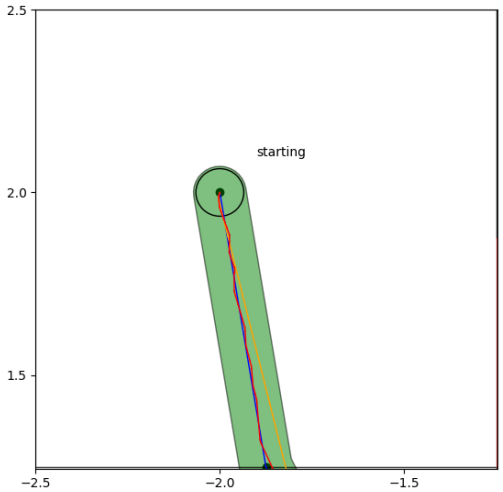


Figura 4.15: Comparativa de caminos obtenidos con filtro de proximidad al camino laxo (amarillo) y restrictivo (rojo). La línea azul es el camino planificado.

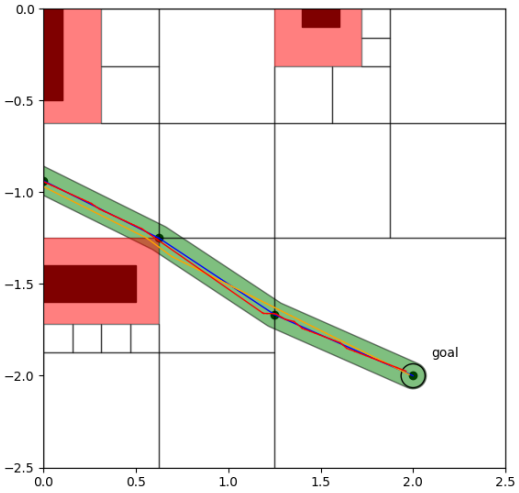
Como se puede observar en la figura 4.16, la solución obtenida con el filtro laxo permite planificar las trayectorias usando todo el ancho de la banda, mientras que al usar un filtro más estricto el planificador tiende a mantenerse sobre el camino planificado. Esta mayor libertad permite planificar trayectorias de forma que se tomen las curvas de manera óptima.

Como era de esperarse, el tiempo de procesamiento con el filtro más laxo fue mayor que la del filtro más estricto (cuadro 4.8). El menor tiempo se logra gracias a que existen menos movimientos a procesar por nivel del árbol. A modo de ejemplo, se puede ver en la figura 4.17a, en la cual los primeros niveles del árbol de búsqueda para el caso de filtros más estrictos cuentan con menos movimientos a procesar que los mismos niveles (fig. 4.17b) pero con filtro más laxo.

El menor tiempo de procesamiento no solo se logra al procesar menos

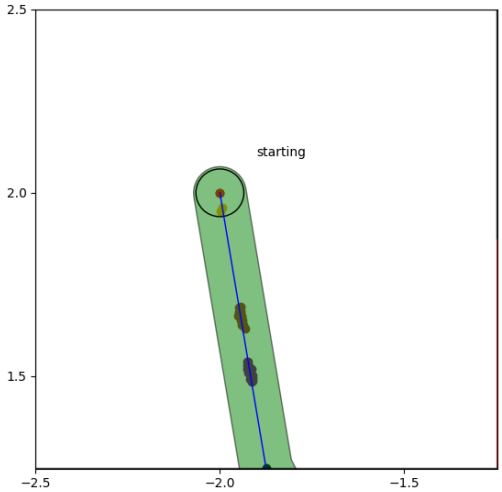


(a) Acercamiento a comparativa

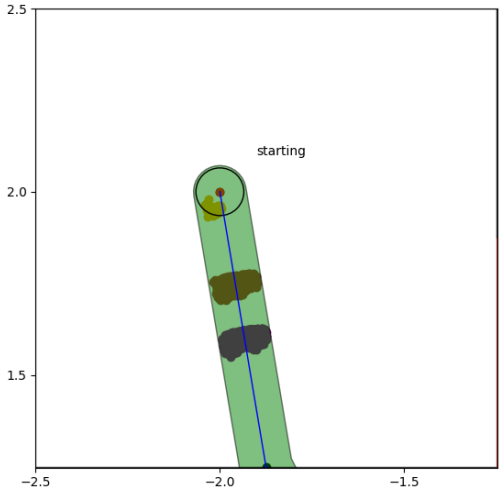


(b) Acercamiento a comparativa

Figura 4.16: Acercamiento a comparativa de caminos obtenidos con filtro de proximidad al camino laxo (amarillo) y restrictivo (rojo). La línea azul es el camino planificado.



(a) Filtro estricto de proximidad al camino.



(b) Filtro laxo de proximidad al camino.

Figura 4.17: Comparativa entre un filtro de proximidad laxo frente otro más estricto. En ambas figuras se ven los niveles 0 (rojo), 5 (amarillo), 20 (marrón) y 30 (violeta) del árbol de búsqueda.

Prox camino	Niveles	Nodos	Procesamiento	Tiempo de solución
1	377	745479	$82{,}2709s$	$6{,}10125s$
9	214	3060	$0{,}230248s$	$6{,}47533s$

Cuadro 4.8: Comparativa de resultados filtro proximidad al camino laxo vs estricto.

movimientos por nivel del árbol de búsqueda, sino como sucede con el filtro de proximidad al destino, porque también son necesarios menos niveles del árbol para encontrar la solución (cuadro 4.8). Como se puede ver en la figura 4.18, para el caso de filtros estrictos fue necesario procesar treinta y nueve niveles para encontrar la solución en las primeras tres posiciones del camino planificado, contra los ciento tres niveles necesarios para un filtro laxo (figura 4.19.

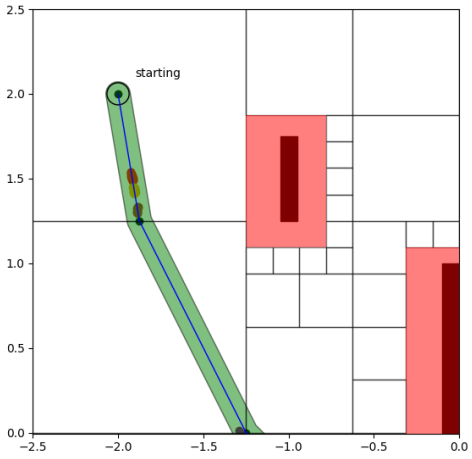


Figura 4.18: Niveles 32 (rojo), 35 (amarillo), 38 (marrón) y 39 (violeta) del árbol de búsqueda para un filtro estricto de proximidad al camino.

Como sucede con el filtro de proximidad al destino, un efecto directo de contar con un filtro más estricto es que se priorizan los movimientos que se mantienen cerca del camino. Esto genera que el conjunto de posiciones por nivel tengan una inclinación orientada al destino (fig. 4.20a) y como consecuencia la media de la orientación es menor. En caso contrario, para un filtro más laxo la heterogeneidad de orientaciones es mayor. Esto se puede ver en la figura 4.20.

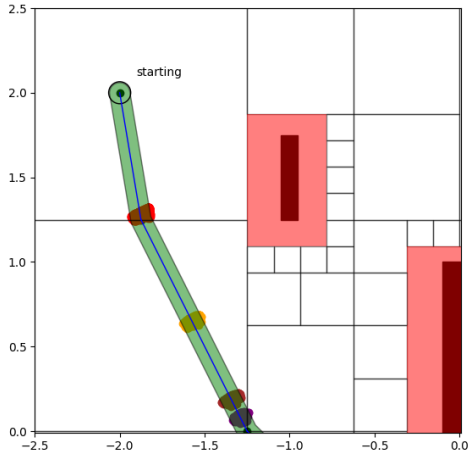


Figura 4.19: Niveles 50 (rojo), 80 (amarillo), 100 (marrón) y 103 (violeta) del árbol de búsqueda para un filtro laxo de proximidad al camino.

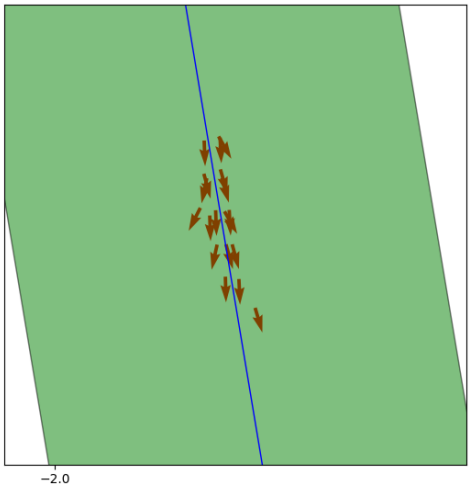
La consecuencia de esta menor inclinación media hace que el planificar pueda optar por Δt mayores y por lo tanto entre un nivel y otro las distancias recorridas son mayores. Análogamente a lo que sucede con el filtro de proximidad al destino, este es el motivo por el cual es necesario menos niveles del árbol de búsqueda para un filtro estricto para encontrar la solución.

La ventaja de contar con un filtro estricto de proximidad al camino es restringir los movimientos que alejan al robot del camino planificado. Además, como sucede con el filtro de proximidad al destino, un filtro estricto de proximidad al camino permite planificar rectas largas en menor cantidad de niveles del árbol de búsqueda. En contrapartida la planificación de curvas es mejor para un filtro laxo de proximidad al camino. Esta mejora en la planificación de curvas se debe a que se cuenta con mayor flexibilidad de movimiento dentro de la banda, la cual permite planificar la toma de la curva de forma óptima.

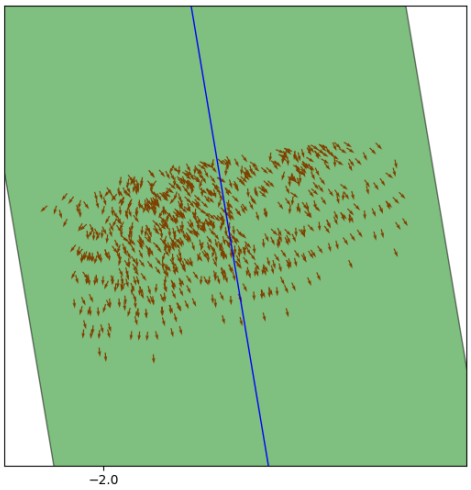
Por lo tanto, es necesario ajustar el filtro de proximidad al camino para procesar la menor cantidad de movimientos por nivel, beneficiando la planificación de las rectas, pero permitiendo cierta flexibilidad para planificar correctamente las curvas.

4.2.3. Frecuencia de aplicación de filtros

En esta sección se analizará cómo afecta la frecuencia de aplicación de los filtros. Para esto, se dejaron fijo los parámetros de configuración de los



(a) Filtro estricto



(b) Filtro laxo

Figura 4.20: Vectores de dirección para el niveles 20 del árbol de búsqueda para filtros estricto y laxo.

filtros de proximidad al destino y proximidad al camino (cuadro 4.9), solo modificando el parámetro que afecta a la frecuencia de aplicación. Además se tomó el conjunto de controladores $\{(v_{max}, v_{max}), (0, v_{max}), (v_{max}, 0)\}$.

Δt_{min}	Movimientos	Prox Dest	Prox camino	V_{max}
$0{,}0154s$	$\emptyset$	2	3	$1m/s$

Cuadro 4.9: Parámetros de entrada comparativa frecuencia de aplicación de filtros.

En la figura 4.21 se puede ver la diferencia en la solución obtenida con una frecuencia menor de aplicación de filtros (camino amarillo), contra una frecuencia mayor de aplicación (camino rojo).

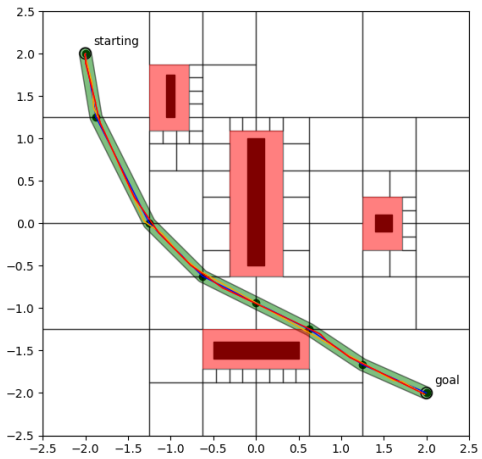
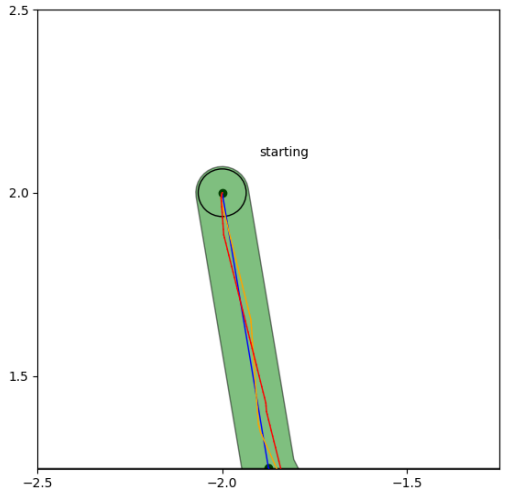


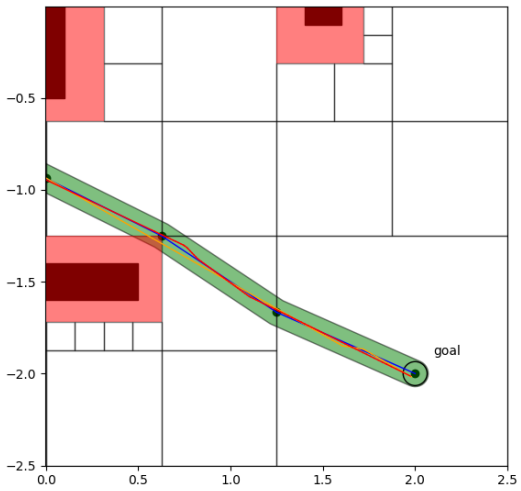
Figura 4.21: Comparativa de caminos obtenidos con una frecuencia menor de aplicación de filtros (camino amarillo) y frecuencia mayor (rojo). La línea azul es el camino planificado.

Como se puede observar en la figura 4.22, la solución obtenida con una frecuencia menor de aplicación de los filtros permite planificar las curvas antes, dado que no se prioriza el llegar a la siguiente posición del camino. Además, una menor frecuencia permite que se use todo el ancho de la banda. En definitiva, se logran hallar trayectorias que tomen las curvas de manera óptima.

Las ventajas de una frecuencia menor de aplicación de filtros son las mismas que se obtienen al tener filtros laxos de proximidad al destino y de proximidad al camino. Esto es porque se deja crecer el árbol de búsqueda



(a) Acercamiento a comparativa



(b) Acercamiento a comparativa

Figura 4.22: Acercamiento a comparativa de caminos obtenidos con una frecuencia menor de aplicación de filtros (camino amarillo) y frecuencia mayor (rojo). La línea azul es el camino planificado.

libremente al no aplicar ningún filtro.

Frec	Niveles	Nodos	Procesamiento	Tiempo de solución
2	186	8856	$0{,}660975s$	$6{,}16658s$
14	526	3493335	$339{,}755s$	$6{,}10961s$

Cuadro 4.10: Comparativa de resultados con frecuencia de aplicación de filtros mayor vs menor.

Como es de esperarse, al aplicar los filtros con menor frecuencia se procesa una mayor cantidad de movimientos por nivel y en consecuencia el tiempo de procesamiento es mayor (cuadro 4.10). Se puede ver figura 4.23a la cantidad de nodos en los primeros niveles del árbol de búsqueda cuando los filtros son aplicados cada dos niveles. La cantidad de movimientos a procesar, es menor que los mismos niveles (figura 4.23b) con una frecuencia menor de aplicación de filtros.

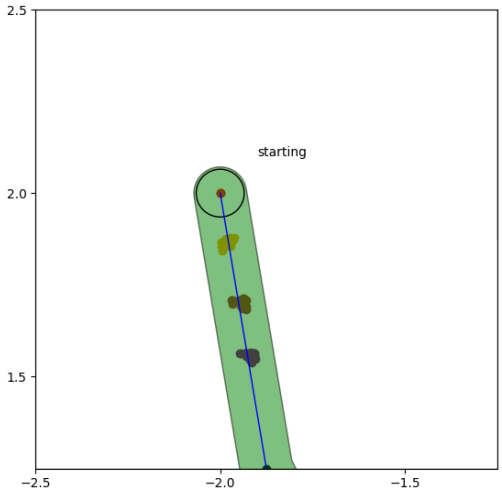
Cuanto mayor sea la frecuencia de aplicación de los filtros, más movimientos se descartan y menos niveles se procesan (cuadro 4.10). A modo de ejemplo, la figura 4.24 muestra para el caso de una frecuencia mayor de filtros, fue necesario procesar 77 niveles para encontrar la solución en las primeras tres posiciones del camino planificado, contra los 121 niveles necesarios para un una frecuencia menor de aplicación de filtros (fig. 4.25).

La consecuencia de tener una frecuencia mayor de aplicación de filtros, es el efecto que se describió anteriormente, el conjunto de puntos por nivel tienen una inclinación orientada al destino (fig. 4.26a) y como consecuencia la media de la orientación es menor. En contrapartida, para una frecuencia menor de aplicación de filtros, como la heterogeneidad de las orientaciones es mayor, genera que la media sea mayor, como se puede ver en la figura 4.26b.

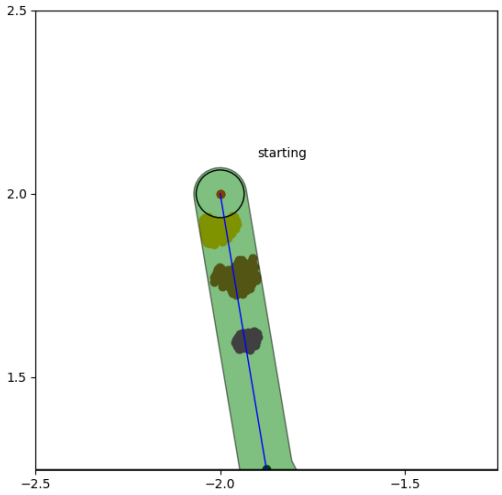
Como sucede en las secciones anteriores, una menor inclinación media hace que al planificar se pueda optar por Δt mayores. Por lo tanto, entre un nivel y otro las distancias recorridas son mayores. Este es el motivo por el cual es necesario menos niveles del árbol de búsqueda, para una frecuencia mayor de aplicación de filtros.

En la figura 4.27 se puede ver el nivel previo a aplicar los filtros y como queda el nivel siguiente luego de aplicarlos. El resultado de aplicar los filtros depende de cuan estrictos sean los mismos, pero como puede apreciarse pasa de ser una nube de posiciones a un conjunto compacto de posiciones. Además, como se puede ver en la figura 4.28, la aplicación de los filtros hace que se pase de una heterogeneidad en las direcciones a una conjunto orientado al destino.

La ventaja de contar con una frecuencia mayor de aplicación, que es la misma que tienen filtros en general, es la posibilidad de planificar rectas



(a) Filtrado con mayor frecuencia.



(b) Filtrado con menor frecuencia.

Figura 4.23: Las figuras muestran la cantidad de nodos en los niveles 0 (rojo), 5 (amarillo), 20 (marrón) y 30 (violeta) del árbol de búsqueda para dos frecuencia de aplicación de filtros diferente.

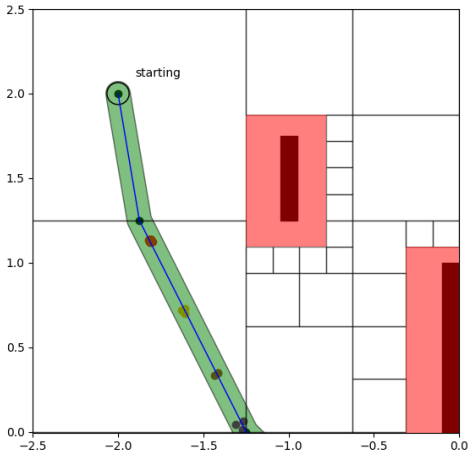


Figura 4.24: Niveles 50 (rojo), 60 (amarillo), 70 (marrón) y 77 (violeta) del árbol de búsqueda para una frecuencia mayor de aplicación de filtros.

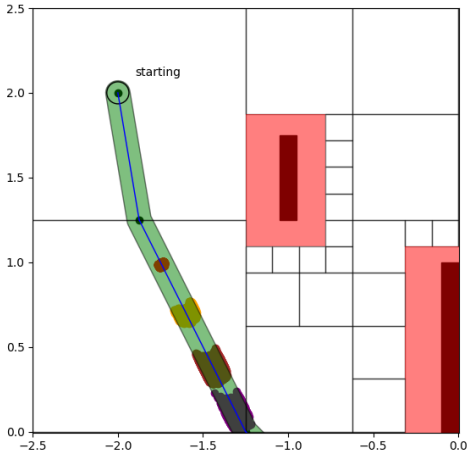
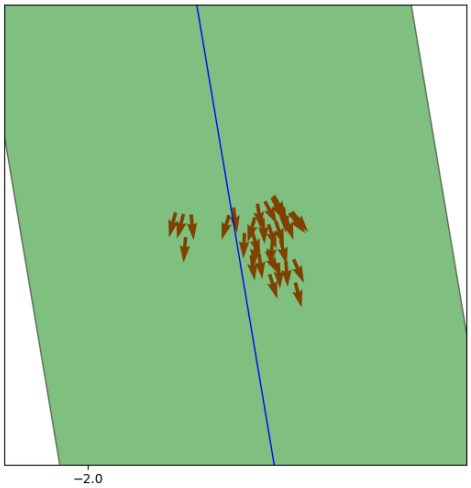
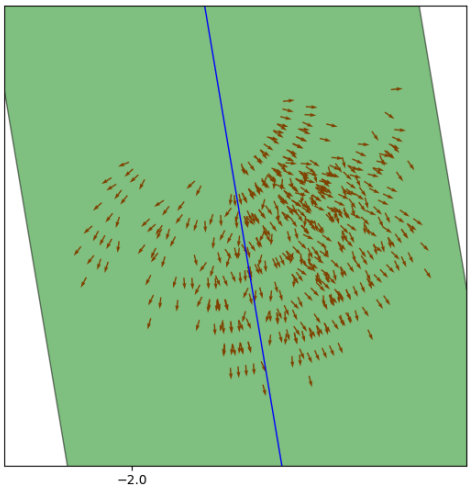


Figura 4.25: Niveles 70 (rojo), 90 (amarillo), 110 (marrón) y 121 (violeta) del árbol de búsqueda para una frecuencia menor de aplicación de filtros.



(a) Frecuencia mayor



(b) Frecuencia menor

Figura 4.26: Vectores de dirección para el niveles 20 del árbol de búsqueda para frecuencia mayor y menor de aplicación de filtros.

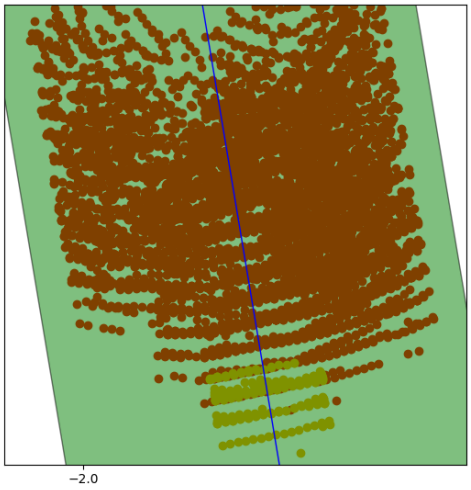


Figura 4.27: Niveles previo a aplicar los filtros (rojo) y nivel donde se aplicaron los filtros (amarillo).

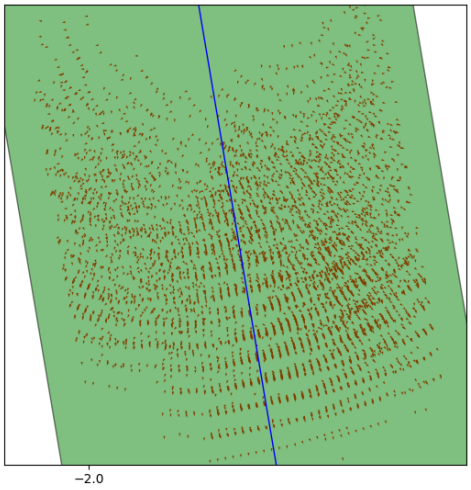
largas en menor cantidad de niveles del árbol de búsqueda. En contrapartida, una frecuencia menor permite explorar caminos que de otra manera serian podados por los filtros. Esta menor frecuencia también hace que se planifiquen la toma de curvas mejor.

De la experimentación se concluye que una menor frecuencia de aplicación de filtros debe estar asociada a filtros más restrictivos, que permitan podar más movimientos cuando se aplican, compensando los niveles de libertad. Por lo tanto, es necesario ajustar la frecuencia de aplicación de filtros para que el aumento exponencial que conlleva no afecte de forma sensible el tiempo de procesamiento. Este ajuste está íntimamente relacionado con los valores de filtros utilizados.

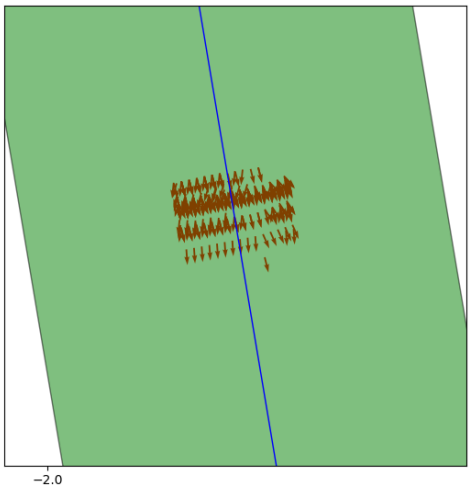
4.3. Cantidad de controladores

Como se presentó en el capítulo anterior (3.4.2), los controladores son en base a la velocidad de cada rueda, siendo el menor conjunto para cada rueda $\{0, v_{max}\}$. Con este se forma el conjunto mínimo de controladores del robot $\{(0, v_{max}), (v_{max}, 0), (v_{max}, v_{max})\}$, siendo el primer valor la velocidad de la rueda izquierda y el segundo el de la derecha. Hay que tener en cuenta que se quita el elemento $(0, 0)$ del conjunto.

Además, en dicho capitulo se detalló que es posible agregar fracciones



(a) Nivel previo a la aplicación de filtros



(b) Nivel de aplicación de filtros

Figura 4.28: Vectores de aplicación de filtros previo y posterior a aplicar los filtros.

de v_{max} al conjunto de controladores $\{0, v_{max}\}$ de cada rueda. Esto se realiza modificando el conjunto de entrada $M = \{m_1, .., m_m\}$ y quedando el conjunto de velocidades de la siguiente forma: $\{0, v_{max}, v_{max} * m_1, .., v_{max} * m_m\}$.

Δt_{min}	Prox Dest	Prox camino	Frec	V_{max}
$0{,}0154s$	2	4	1	$1m/s$

Cuadro 4.11: Parámetros de entrada comparativa cantidad de controladores.

Se realizaron pruebas sobre la cantidad de controladores para analizar como los mismos afectaban a la solución final y al costo de procesamiento del algoritmo. Para esto, se dejaron fijos los demás parámetros (cuadro 4.11) y solo se modificó el valor de M. Se usaron los siguientes conjuntos de controladores:

- 3 controladores $\{(0, v_{max}), (v_{max}, 0), (v_{max}, v_{max})\}$
- 8 controladores $\{(0, v_{max}), (0, \frac{v_{max}}{2}), (v_{max}, 0), (\frac{v_{max}}{2}, 0), (v_{max}, v_{max}), (\frac{v_{max}}{2}, v_{max}), (v_{max}, \frac{v_{max}}{2}), (\frac{v_{max}}{2}, \frac{v_{max}}{2})\}$
- 15 controladores $\{(0, v_{max}), (0, \frac{v_{max}}{2}), (0, \frac{v_{max}}{4}), (v_{max}, 0), (\frac{v_{max}}{2}, 0), (\frac{v_{max}}{4}, 0), (v_{max}, v_{max}), (\frac{v_{max}}{2}, v_{max}), (\frac{v_{max}}{4}, v_{max}), (v_{max}, \frac{v_{max}}{2}), (v_{max}, \frac{v_{max}}{4}), (\frac{v_{max}}{2}, \frac{v_{max}}{2}), (\frac{v_{max}}{4}, \frac{v_{max}}{2}), (\frac{v_{max}}{2}, \frac{v_{max}}{4}), (\frac{v_{max}}{4}, \frac{v_{max}}{4})\}$

En la figura 4.29 se puede observar las soluciones obtenidas con tres controladores (línea amarilla), ocho controladores (línea roja) y quince controladores (línea violeta).

Como se puede ver en la figura 4.30, a medida que se aumenta la cantidad de controladores la solución obtenida mejora. Esto se debe a que se cuenta con un abanico más amplio de giros que permiten orientar al robot de manera más precisa. Además, al contar con mayor cantidad de controladores es posible que el robot tome las curvas de forma que a la salida de la misma se encuentre orientado a la siguiente posición del camino.

Al contar con mayor número de controladores, el planificador va a optar por seleccionar una velocidad menor a la máxima en determinadas ocasiones para orientar el robot al destino de forma más precisa y rápida. Esto permite que luego pueda tomar rectas más largas a velocidad máxima, como se puede observar en la figura 4.30. En contrapartida, con una cantidad menor de controladores, se pierde más tiempo en orientar al robot al destino, por lo que las rectas que se pueden tomar a máxima velocidad son menores. Por lo tanto, el contar con mayor cantidad de controladores, hace que el robot esté más tiempo recorriendo la trayectoria a velocidad máxima y que la solución obtenida sea mejor en el tiempo de recorrida.

También al contar con un mayor número de controladores, se gana en flexibilidad, permitiendo trazar curvas más cerradas que si contáramos con

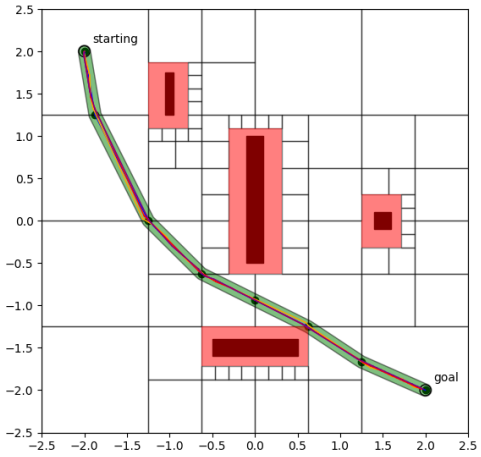


Figura 4.29: Comparativa de caminos obtenidos con 3 (amarillo), 8 (rojo) y 15 (violeta) controladores

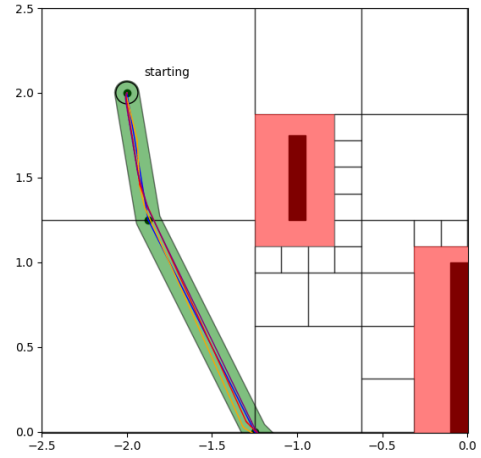


Figura 4.30: Acercamiento a comparativa de caminos obtenidos con 3 (amarillo), 8 (rojo) y 15 (violeta) controladores

un menor número de controladores. Esto hace que para algunos entornos con un menor número de controladores no se encuentre una solución.

Contr	Niveles	Nodos	Procesamiento	Tiempo de solución
3	311	9336	$0{,}70355s$	$6{,}26643s$
8	126	151320	$9{,}15863s$	$6{,}0979s$
15	135	6331725	$400{,}013s$	$6{,}08638s$

Cuadro 4.12: Comparativa de resultados por cantidad de controladores.

El problema de aumentar la cantidad de controladores, es que aumenta la cantidad de movimientos a procesar para obtener la solución. Como se vio anteriormente, el crecimiento del árbol de búsqueda es exponencial, esto hace que el tiempo de procesamiento al contar con un mayor número de controladores crezca de igual manera. Este aumento en el tiempo de procesamiento (cuadro 4.12), muchas veces, no se ve reflejado en una disminución del tiempo del recorrido que justifique dicha demora.

Para lograr procesar una mayor cantidad de controladores en menor tiempo, es necesario ser más restrictivos con los filtros a aplicar. Esto nos genera que la solución obtenida no sea mejor a la obtenida con una menor cantidad de controladores. Además, que puede suceder que por las restricciones de los filtros no se llegue a obtener una solución.

Por lo tanto, es necesario ajustar la cantidad de controladores para ganar en flexibilidad de movimientos, pero ajustando los filtros para que no aumente el tiempo de procesamiento de forma desproporcionada.

4.4. Comparativa algoritmos

En esta sección se compara el resultado obtenido con la estrategia básica de Giro y Avanzo (2.2.4), contra la solución implementada. Para esto, se compararon tres escenarios, con estas dos estrategias, para así poder identificar las ventajas y desventajas de cada uno. En cada uno de los escenarios, se optó por la combinación de parámetros de filtros y cantidad de controladores que minimiza el tiempo de procesamiento más el tiempo de recorrido (cuadro 4.13) .

Escenario	Δt_{min}	Mov	Prox Dest	Prox camino	Frec	V_{max}
1	$0{,}0154s$	$\emptyset$	1	4	3	$1m/s$
2	$0{,}0154s$	0,5	2	7	2	$1m/s$
3	$0{,}0154s$	$\emptyset$	1	1	2	$1m/s$

Cuadro 4.13: Parámetros de entrada comparativa de algoritmos.

El resultado del primer escenario se puede ver en la figura 4.31. En ella se encuentra la trayectoria obtenida por la solución implementada (camino amarillo), contra la estrategia de Giro y Avanzo (camino azul). Para este caso, el tiempo de procesamiento más el tiempo de recorrido supera al tiempo que lleva recorrer el camino con la estrategia de Giro y Avanzo. Esta ventaja en el tiempo de recorrido se debe a que la estrategia de Giro y Avanzo es ventajosa cuando el camino planificado tiene pocas curvas, como es este caso, que llevan a que el robot recorra grandes distancias en línea recta. El costo de planificar dichas rectas con la solución implementada es superior al costo que lleva planificar con la estrategia de Giro y Avanzo. Por lo que, si bien el tiempo de recorrido es menor que con la estrategia de Giro y Avanzo, el tiempo mayor de procesamiento no compensa la mejora en la solución.

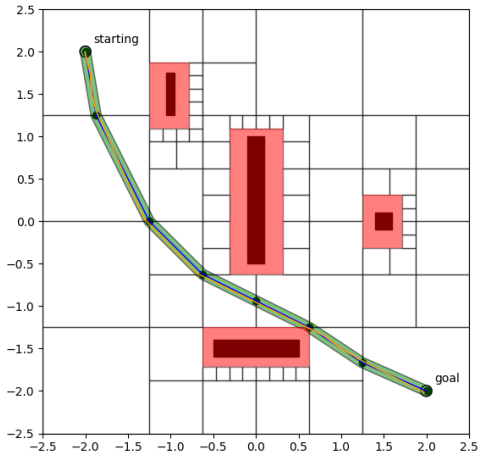


Figura 4.31: Recorrido amarillo es el planificado con la solución implementada contra el azul que es el obtenido con Giro y Avanzo.

En el cuadro 4.14 se puede observar los tiempos de procesamiento y de recorrido obtenidos con la solución implementada, contra el algoritmo de Giro y Avanzo. Es de notar que la mejora obtenida con el algoritmo implementado no justifica el tiempo de procesamiento invertido.

En el segundo escenario (fig 4.32), el tiempo de procesamiento más el tiempo de recorrido obtenido con la solución implementada es menor al obtenido con la estrategia de Giro y Avanzo (cuadro 4.15). En este caso, al ser un recorrido planificado con una mayor cantidad de curvas que la anterior, la estrategia de Giro y Avanzo no es efectiva, siendo la solución implementada mejor en la planificación de dicho recorrido.

En la tabla 4.15, se puede observar los tiempos de procesamiento y de

-	Tiempo recorrido	Tiempo procesamiento	Total
GridSearch	$6{,}1335s$	$0{,}2962s$	$6{,}4297s$
GiroAvanzo	$6{,}2852s$	$0{,}0009s$	$6{,}2861s$

Cuadro 4.14: Escenario 1: resultados obtenidos Grid Search contra Giro y Avanzo.

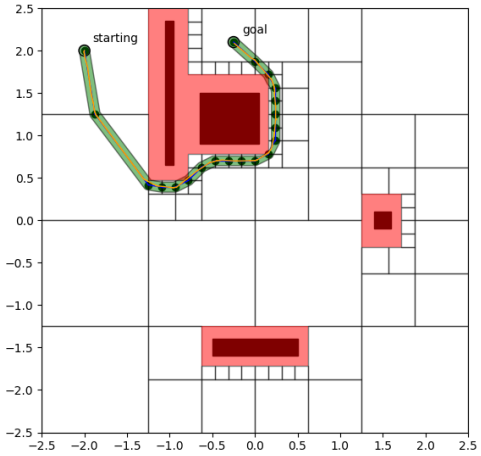


Figura 4.32: Recorrido amarillo es el planificado con la solución implementada contra el azul que es el obtenido con Giro y Avanzo.

recorrido obtenidos con la solución implementada, contra el algoritmo de Giro y Avanzo. En este caso se puede observar que la mejora obtenida con el algoritmo implementado, justifica el tiempo mayor de procesamiento.

-	Tiempo recorrido	Tiempo procesamiento	Total
GridSearch	$5{,}1319s$	$0{,}3667s$	$5{,}4986s$
GiroAvanzo	$5{,}6396s$	$0{,}0009s$	$5{,}6405s$

Cuadro 4.15: Escenario 2: resultados obtenidos Grid Search contra Giro y Avanzo.

Los resultados de del ultimo escenario se pueden ver en la figura 4.33. En este caso, como en el anterior, el tiempo de recorrido obtenido con grid search más el tiempo de procesamiento es menor al tiempo obtenido con Giro y Avanzo.

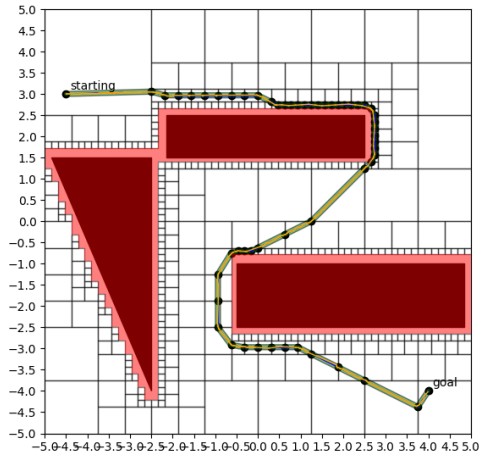


Figura 4.33: Recorrido amarillo es el planificado con la solución implementada contra el azul que es el obtenido con Giro y Avanzo.

En el cuadro 4.16 se puede observar los tiempos de procesamiento y de recorrido obtenidos con la solución implementada, contra el algoritmo de Giro y Avanzo. Se puede observar que la mejora obtenida con el algoritmo implementado justifica el tiempo mayor de procesamiento.

Como se vio anteriormente, la estrategia de Giro y Avanzo es mejor en casos donde los caminos planificados contengan mayor cantidad de rectas que de curvas. Esto se debe a que planificar dichas rectas lleva un tiempo menor de procesamiento. En contrapartida para caminos planificados con

-	Tiempo recorrido	Tiempo procesamiento	Total
GridSearch	$20{,}5161s$	$0{,}9570s$	$21,4731s$
GiroAvanzo	$21{,}7228s$	$0{,}0018s$	$21{,}7246s$

Cuadro 4.16: Escenario 3: resultados obtenidos Grid Search contra Giro y Avanzo.

mayor cantidad de curvas, la solución implementada es mejor.

Hay otro factor que no se tiene en cuenta en la estrategia de Giro y Avanzo, es que al considerar que la aceleración es infinita, las detenciones en cada punto del camino se hacen en tiempo cero por lo que no se tiene en cuenta la inercia que tenga el robot en ese momento. La diferencia con la estrategia implementada es que el efecto de la inercia es menor ya que no se detiene el robot en ningún momento del recorrido. Por lo tanto, es de suponer que esta diferencia es a favor de la estrategia implementada, dando un mejor tiempo de recorrido, debido a que la detención del robot no se puede hacer en tiempo cero.

4.5. Simulación de la trayectoria planificada.

En esta sección se muestran los resultados obtenidos luego de simular la trayectoria planificada en diferentes escenarios. En cada escenario se utilizarán las estrategias basadas en Grid Search y Giro y Avanzo. Se comparan los resultados entre la simulación de uno y el otro. El robot utilizado en todos los escenarios es el Khepera III. El mismo fue modelado por el planificador utilizando los parámetros de configuración que se muestran en el Cuadro 4.17. La velocidad máxima del robot en cada rueda es de $19{,}1rad/s$. Teniendo en cuenta que el radio diámetro de cada rueda es de $0{,}0041m$ [13], la velocidad lineal equivalente es de $0{,}39155m/s$.

En todos los escenarios, para el Giro y Avanzo los controles son (v_{max}, v_{max}), $(-v_{max}, v_{max})$ y $(0, 0)$.

Distancia entre ruedas	Diámetro	Margen de seguridad
0.08841 m	0.13 m	10 %

Cuadro 4.17: Configuración del Khepera III utilizada para todos los escenarios.

4.5.1. Entorno 1 - Trayectoria simple

El objetivo de este escenario es probar que el robot es capaz de recorrer una trayectoria relativamente simple, sin curvas pronunciadas. El escenario

se realizó en el entorno que se muestra en la Figura 3.3a. Tal como se puede ver en la Figura, el robot parte de la posición $(-2{,}0, 2{,}0)$ y con θ igual a $-116°$. El objetivo está situado en el punto $(2{,}0, -2{,}0)$.

Los parámetros de entrada utilizados en Grid Search se muestran en el Cuadro 4.18.

Δt_{min}	Movimientos	Prox Dest	Prox camino	Filtrar	V_{max}
0.154 s	$\emptyset$	3	1	2	0.1 m/s

Cuadro 4.18: Parámetros de entrada del Grid Search en el escenario 1.

Resultado tras la ejecución.

-	Tiempo	Punto final	Dist. Objetivo
Plan GridSearch	$60{,}6511s$	$(1{,}9835, -1{,}9925)$	$0{,}018246m$
Sim GridSearch	$61{,}5840s$	$(2{,}1036, -2{,}1296)$	$0{,}165919m$
Plan GiroAvanzo	$61{,}1241s$	$(2{,}00, -2{,}00)$	$0{,}000000m$
Sim GiroAvanzo	$62{,}5120s$	$(2{,}0792, -2{,}2262)$	$0{,}239665m$

Cuadro 4.19: Escenario 1: resultados obtenidos.

En el cuadro 4.19 se pueden ver los resultados obtenidos durante la planificación para Grid Search (Plan GridSearch) y para el Giro y Avanzo (Plan GiroAvanzo) junto con sus respectivas simulaciones (Sim GridSearch y Sim GiroAvazo). En este primer escenario, la diferencia significativa entre la simulación y la planificación se encuentra en la distancia al objetivo. En el Grid Search el robot quedó aproximadamente 8 veces más lejos que lo planificado. Si se estuviera en un entorno donde el error tolerado sea menor a $0,1$ probablemente el robot hubiese colisionado. Aun así, teniendo en cuenta que el controlador no aplica correcciones ni tracking, se puede considerar que los resultados para este primer caso son bastante satisfactorios.

En la figura 4.34 se puede observar las diferencias entre el camino planificado (azul), la trayectoria planificada (amarillo) y el recorrido del Khepera III en la simulación (verde). Por otro lado, la comparativa entre la simulación de Grid Search y de Giro y Avanzo se puede ver en la Figura 4.35. Girando y avanzando el robot quedó un poco más lejos que en la simulación del Grid Search. Incluso, el robot paso más cerca del obstáculo, pudiendo haber colisionado contra este de tener un poco más de error en algún movimiento.

4.5.2. Entorno 1 - Trayectoria simple a mayor velocidad

Este segundo escenario, al igual que anterior, se realizó en el entorno que se muestra en la figura 3.3a con el robot partiendo del mismo punto

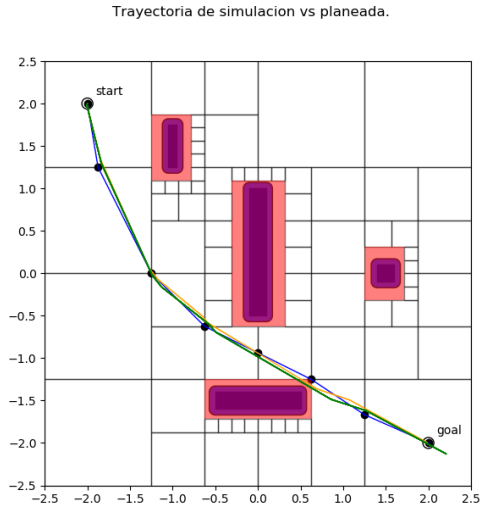


Figura 4.34: Escenario 1: Comparativas entre el camino planificado (azul), trayectoria planificada (amarillo) y la trayectoria realizada por el Khepera III durante la simulación (verde).

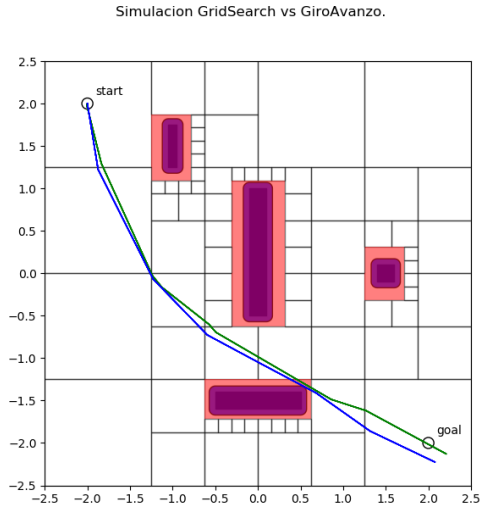


Figura 4.35: Escenario 1: Comparativas entre la simulación del Grid Search (verde) y del Giro y Avanzo (azul).

($p_{ini} = (-2{,}0, 2{,}0)$) y con la misma orientación (θ=$-116°$) que en el escenario anterior. El punto objetivo también es el el mismo ($p_{fin} = (2{,}0, -2{,}0)$). Los parámetros de entrada se muestran en el Cuadro 4.20. La diferencia con el primer escenario es que la velocidad máxima de cada rueda será de $0,2m/s$. El objetivo de este escenario es observar si el desempeño de Grid Search se ve afectado negativamente al incrementar la velocidad.

Δt_{min}	Movimientos	Prox Dest	Prox camino	Filtrar	V_{max}
0.154 s	$\emptyset$	3	1	2	0.2 m/s

Cuadro 4.20: Parámetros de entrada escenario 2.

Resultado tras la ejecución.

-	Tiempo	Punto final	Dist. Objetivo
Plan GridSearch	$30{,}8413s$	$(1{,}9548, -1{,}9660)$	$0{,}0565600m$
Sim GridSearch	$31{,}7600s$	$(2{,}1932, -2{,}1478)$	$0{,}2432511m$
Plan GiroAvanzo	$30{,}6581s$	$(2{,}00, -2{,}00)$	$0{,}00m$
Sim GiroAvanzo	$32{,}9440s$	$(2{,}3965, -1{,}54503)$	$0{,}6034980m$

Cuadro 4.21: Escenario 2: resultados obtenidos.

En el cuadro 4.21 se tienen los resultados obtenidos durante la planificación y la simulación para ambas estrategias. En este caso, la diferencia de distancias al destino entre lo planificado por el Grid Search y su simulación es de $0{,}186m$, aproximadamente 3 veces más de lo planificado. Por lo tanto, se puede decir que, para este entorno, incrementar la velocidad no deteriora el resultado de la simulación, sino que lo contrario. Esto se puede deber a la menor cantidad de comandos que se deben aplicar en la simulación de este escenario con respecto a la anterior. Cada aplicación de comando introduce error respecto a lo planificado. No es lo mismo tener que aplicar dos comandos con a una velocidad de $0{,}1m/s$ en cada rueda que uno a $0{,}2$ para llegar al mismo destino.

En la figura 4.36 se puede observar las diferencias entre el camino planificado (azul), la trayectoria planificada (amarillo) y el recorrido del Khepera III en la simulación (verde). Salvo en algunas excepciones en los cuales algún comando se aplicó durante un tiempo significativamente mayor o menor al deseado, la simulación se mantiene bastante fiel a la trayectoria planificada. Se puede observar que el robot quedó más lejos del destino que en el caso anterior, sin embargo, como se mencionó anteriormente, la distancia entre lo planificado y simulado es menor.

La comparativa entre la simulación de Grid Search y de Giro y Avanzo se puede ver en la Figura 4.37. En este caso, el Grid Search no empeoro

tanto su rendimiento en la simulación, mientras que el Giro y Avanzo obtuvo peores resultados que en el escenario 1. Posiblemente debido a que el controlador de Giro y Avanzo no corrige el error producido por la inercia tan bien como se deseaba. Especialmente en los giros.

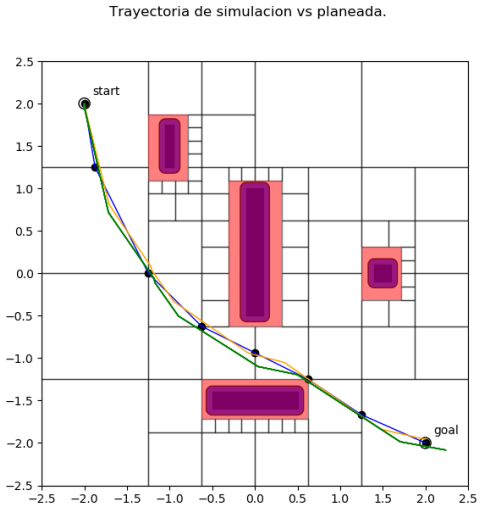


Figura 4.36: Escenario 2: Comparativas entre el camino planificado (azul), trayectoria planificada (amarillo) y la trayectoria realizada por el Khepera III durante la simulación (verde).

4.5.3. Entorno 2 - Curva cerrada

El objetivo de este escenario es ver como como se comporta el robot en la simulación cuando la trayectoria a seguir es una curva cerrada. Para este tercer escenario el espacio de configuración es el que se muestra en la Figura 4.32. El robot parte de la posición $(-2{,}0, 2{,}0)$ y con θ igual a $-116°$. El punto objetivo es el $(-0{,}25, 2{,}1)$. Los parámetros de entrada se muestran en el Cuadro 4.22.

Δt_{min}	Movimientos	Prox Dest	Prox camino	Filtrar	V_{max}
0.154 s	$\emptyset$	3	1	2	0.2 m/s

Cuadro 4.22: Parámetros de entrada para Grid Search en el escenario 3.

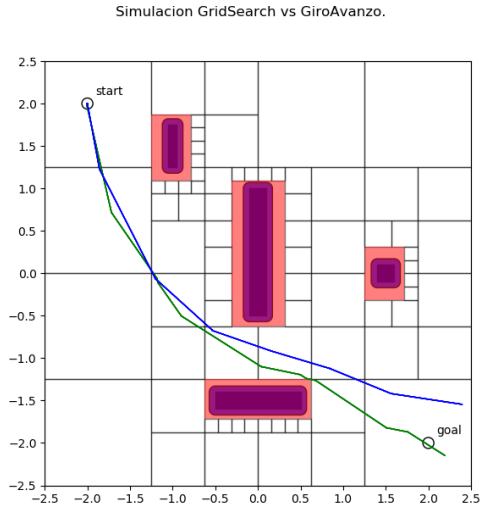


Figura 4.37: Escenario 2: Comparativas entre la simulación del Grid Search (verde) y del Giro y Avanzo (azul).

-	Tiempo	Punto final	Dist. Objetivo
Plan GridSearch	$25{,}8140s$	$(-0{,}19860, 2{,}06270)$	$0{,}0635078m$
Sim GridSearch	$26{,}4960s$	$(-0{,}30720, 2{,}04544)$	$0{,}0790483m$
Plan GiroAvanzo	$26{,}1499s$	$(-0{,}25000, 2{,}10000)$	$0{,}0000000m$
Sim GiroAvanzo	$31{,}7440s$	$(-0{,}26337, 2{,}16105)$	$0{,}0624969m$

Cuadro 4.23: Escenario 3: resultados obtenidos.

Resultado tras la ejecución.

Los resultados obtenidos tras la ejecución se encuentran en el cuadro 4.23. En este caso, lo simulado para el Grid Search es bastante fiel a lo planificado. Los $0{,}0155405m$ de diferencia en la distancia al objetivo son despreciables teniendo en cuenta que el robot tiene un radio de $0{,}13m$.

En la figura 4.38 se puede observar las diferencias entre el camino planificado (azul), la trayectoria planificada (amarillo) y el recorrido del Khepera III en la simulación. Salvo en algunas excepciones en los cuales algún comando se aplicó durante un tiempo significativamente mayor o menor al deseado, la simulación se mantiene bastante fiel a la trayectoria planificada.

Al igual que en los escenarios anteriores, se cuenta con la figura 4.39 con la comparativa entre la simulación de Grid Search y Giro y Avanzo. En este caso no hay diferencia significativa entre ambos salvo en los tiempos de recorrido. Como se ve en el cuadro 4.25, girando y avanzando al robot le

llevaría $7{,}104s$ más ($40\,\%$ más) que con utilizando Grid Search.

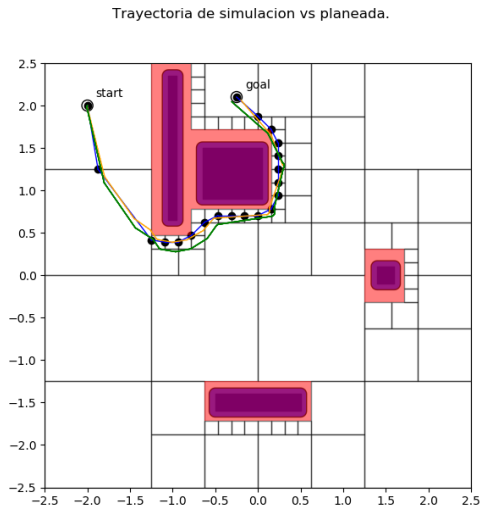


Figura 4.38: Escenario 3: Comparativas entre el camino planificado (azul), trayectoria planificada (amarillo) y la trayectoria realizada por el Khepera III durante la simulación (verde).

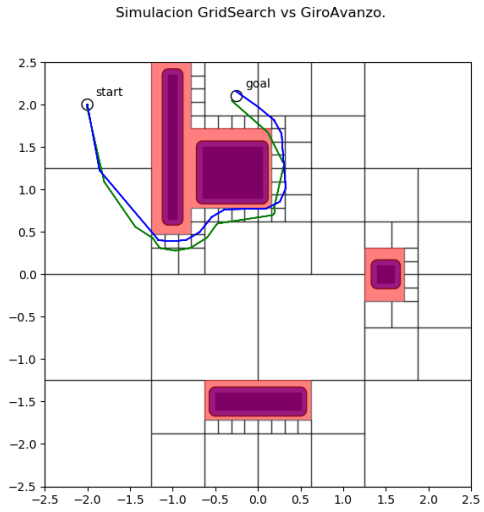


Figura 4.39: Escenario 3: Comparativas entre la simulación del Grid Search (verde) y del Giro y Avanzo (azul).

4.5.4. Entorno 2 - Curva cerrada a mayor velocidad

Para el cuarto escenario el espacio de configuración es el que se muestra en la figura 4.33. El robot parte de la posición $(-2{,}0, 2{,}0)$ y con θ igual a $-116°$. El punto objetivo es el $(-0{,}25, 2{,}1)$. Los parámetros de entrada se muestran en el Cuadro 4.24. La diferencia con el escenario anterior es que la velocidad que tomaran las ruedas es de $0{,}3m/s$. El objetivo de este escenario es ver si hay variación significativa en los resultados respecto al anterior al incrementar la velocidad.

Δt_{min}	Movimientos	Prox Dest	Prox camino	Filtrar	V_{max}
0.154 s	$\emptyset$	3	1	2	0.3 m/s

Cuadro 4.24: Parámetros de entrada escenario 4.

Resultado tras la ejecución.

-	Tiempo	Punto final	Dist. Objetivo
Plan GridSearch	$17{,}2676s$	$(-0{,}20160, 2{,}07840)$	$0{,}0530011m$
Sim GridSearch	$17{,}6800s$	$(0{,}111057, 1{,}01758)$	$1{,}1410501m$
Plan GiroAvanzo	$17{,}6039s$	$(-0{,}25000, 2{,}10000)$	$0{,}0000000m$
Sim GiroAvanzo	$24{,}7840s$	$(-0{,}28826, 2{,}05763)$	$0{,}0570880m$

Cuadro 4.25: Escenario 4: resultados obtenidos.

El cuadro 4.25 muestra los resultados tras la ejecución. En este caso, el robot quedó en el punto $(0{,}111057, 1{,}01758)$, muy lejos del objetivo. Si se ve la figura 4.40 que muestra la trayectoria planificada (amarillo) y la simulada (verde), se puede observar como el Khepera III colisiona contra un obstáculo durante la simulación. Claramente, en este caso, realizar la curva a mayor velocidad lleva a un mayor error.

Para este escenario, el desempeño en la simulación para el Giro y Avanzo obtuvo mejores resultados que Grid Search. El robot llega al objetivo sin chocar. Esto se debe a que el controlador de Giro y Avanzo si posee corrección, mientras que el de Grid Search no. En la figura 4.41 se pueden ver los resultados para ambas simulaciones.

4.5.5. Entorno 2 - Curva cerrada con mas controles

En el quinto escenario es en el mismo entorno que el cuarto escenario, pero en este caso las velocidades de cada rueda podrán tomar los valores $0, 28m/s$ y $0, 14m/s$. El objetivo de este escenario fue probar la hipótesis que

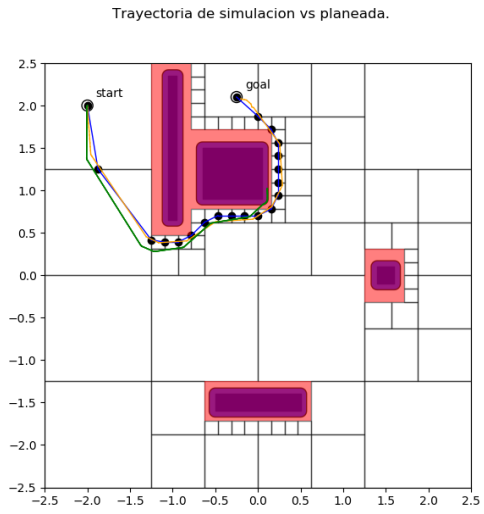


Figura 4.40: Escenario 4: Comparativas entre el camino planificado (azul), trayectoria planificada (amarillo) y la trayectoria realizada por el Khepera III durante la simulación (verde).

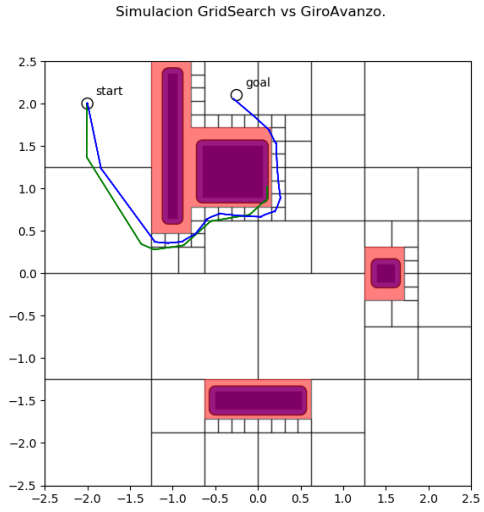


Figura 4.41: Escenario 4: Comparativas entre la simulación del Grid Search (verde) y del Giro y Avanzo (azul).

con velocidades más pequeñas y con más controles, el Grid Search tiene un mejor desempeño.

Los parámetros de entrada del Grid Search se muestra en el cuadro 4.26. Así como en el cuarto escenario, el robot parte de la posición $(-2{,}0, 2{,}0)$ y con θ igual a $-116°$ y el punto objetivo es el $(-0{,}25, 2{,}1)$.

Δt_{min}	Movimientos	Prox Dest	Prox camino	Filtrar	V_{max}
$0{,}154s$	$\{0{,}5\}$	3	1	2	$0{,}28m/s$

Cuadro 4.26: Parámetros de entrada escenario 5.

Resultado tras la ejecución.

-	Tiempo	Punto final	Dist. Objetivo
Plan GridSearch	$18{,}3816s$	$(-0{,}20100, 2{,}09310)$	$0{,}0494834m$
Sim GridSearch	$18{,}9280s$	$(-0{,}28845, 2{,}18304)$	$0{,}0915098m$
Plan GiroAvanzo	$18{,}8248s$	$(-0{,}25000, 2{,}10000)$	$0{,}0000000m$
Sim GiroAvanzo	$25{,}8080s$	$(-0{,}35128, 2{,}07169)$	$0{,}1051623m$

Cuadro 4.27: Escenario 5: resultados obtenidos.

Los resultados de las ejecuciones se pueden ver en el cuadro 4.27. Los resultados obtenidos por Grid Search notoriamente mejores que en el escenario cuatro. La diferencia en las distancias al destino es de $0{,}0420m$, y el robot quedó a una distancia menor que el error tolerado. El costo de estos mejores resultados fue un mayor tiempo de cómputo para la planificación del Grid Search.

En la figura 4.42 se puede observar las diferencias entre el camino planificado (azul), la trayectoria planificada (amarillo) y el recorrido del Khepera III en la simulación. A diferencia del escenario cuatro, el robot llega satisfactoriamente al objetivo sin colisionar con algún obstáculo. Por otro lado, en la Figura 4.43 se tiene la comparativa entre la simulación de Grid Search y el Giro y Avanzo.

4.5.6. Análisis y conclusión general de los resultados

En todos los escenarios se puede observar que el tiempo de recorrido de las simulaciones es siempre mayor a lo planificado. Esto se debe en parte a que los controladores en Webots operan cada un ciclo de $16ms$ (time step), obligando a que el tiempo de aplicación de comandos siempre sea múltiplo del time step. Como es muy poco probable que la planificación implique que los comandos se deban aplicar durante un Δt divisible entre 16, los comandos siempre se aplicarán durante un tiempo diferente al planificado. En

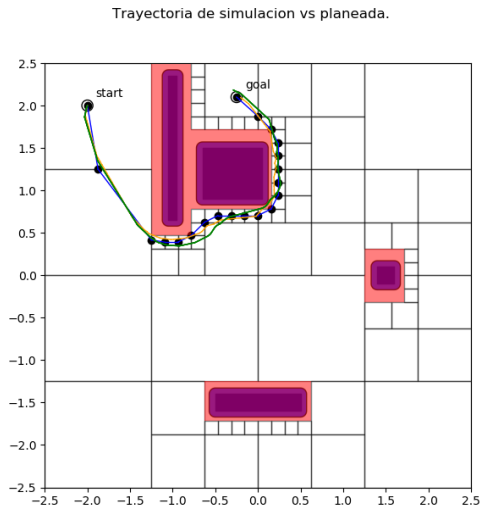


Figura 4.42: Escenario 5: Comparativas entre el camino planificado (azul), trayectoria planificada (amarillo) y la trayectoria realizada por el Khepera III durante la simulación (verde).

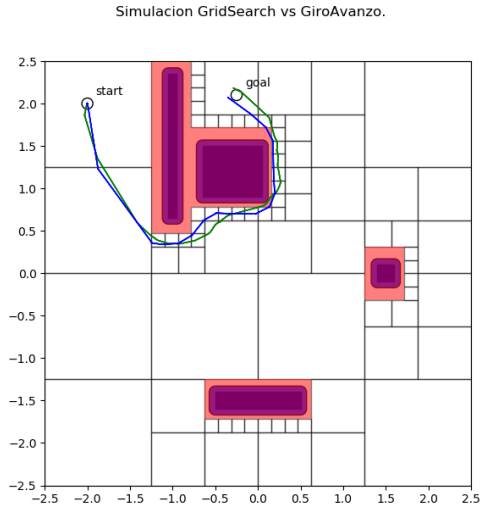


Figura 4.43: Escenario 5: Comparativas entre la simulación del Grid Search (verde) y del Giro y Avanzo (azul).

consecuencia, se introduce un error o diferencia en la simulación respecto al plan de ejecución. Si a esto se le suma que durante la planificación se consideró la aceleración como infinita y que los controladores de Webots carecen de un seguimiento o control apropiado del movimiento del robot, es que, por ejemplo, se pueden encontrar resultados como en el escenario 4 (sección 4.5.4) donde el robot colisiona.

Si se comparan los resultados de la simulación del Grid Search respecto los del Giro y Avanzo, se puede apreciar (como era de esperarse) que el tiempo de recorrido girando y avanzando es mayor que en Grid Search. Por otro lado, el controlador del Giro y Avanzo a pesar de tener una corrección, deja en casi todos los casos al robot más lejos del destino.

En conclusión, los resultados obtenidos en cada escenario, salvo en el 4 (sección 4.5.4), están dentro de cierto margen de error esperado. Esto implica que la trayectoria planificada por el algoritmo basado en Grid Search se puede simular satisfactoriamente y salvo excepciones no se obtienen peores resultados que si se siguiera una estrategia de Giro y Avanzo. Si se quisiera obtener mejores resultados en la simulación, sería necesario un controlador de bucle cerrado para un mejor seguimiento de las trayectorias.

Capítulo 5

Conclusiones y trabajo futuro

En el siguiente capítulo se exponen conclusiones referentes al proyecto en general, los resultados alcanzados, dificultades encontradas y se plantean posibles mejoras.

5.1. Conclusiones generales

Uno de los mayores desafíos enfrentados durante el desarrollo de este trabajo fue reducir el tiempo de procesamiento del algoritmo basado Grid Search. Las primeras pruebas arrojaban resultados muy negativos, en donde el programa demoraba decenas de minutos en dar una solución, si es que se hallaba. Resolver este problema fue a lo que se le dedicó más tiempo. Lo primero fue reducir el tamaño del problema sub dividiéndolo en problemas más pequeños, es por eso que decidió aplicar Gird Search a solo tres puntos del camino e ir iterando hasta encontrar la solución. Aun así, los resultados no eran buenos, por lo que los filtros jugarían un papel fundamental para reducir el costo de procesamiento. Se probaron diferentes condiciones de poda e incrementar las restricciones a aplicar en los filtros implementados (sección 3.4.5) para acotar el crecimiento del árbol búsqueda, corriendo el riesgo de perder soluciones válidas. Es así que se llegó a los filtros ya mencionados y los parámetros para adaptar su comportamiento según sea necesario. Es por esto que es importante destacar los beneficios de los filtros creados y la metodología de aplicación de Grid Search, que permitieron encontrar soluciones de planificación de control, en los entornos planteados, en tiempos aceptables de procesamiento. Logrando así un balance entre flexibilidad de movimiento y tiempo de procesamiento.

De la experimentación se destaca la mejora en la planificación de con-

trol obtenida con el algoritmo implementado, contra el algoritmo básico de Giro y Avanzo. Como se pudo ver, las soluciones obtenidas fueron de menor tiempo de recorrido. También, las soluciones obtenidas, tienen un mejor recorrido desde la posición inicial a la final, donde el robot no se detiene en ningún momento. Es por esto, que las soluciones obtenidas con el algoritmo implementado, se benefician de un menor efecto negativo por la inercia del robot, que el algoritmo de Giro y Avanzo.

Al pasar a la simulación en Webots, el primer problema a resolver era la inercia del robot, ya que se había asumido que la aceleración iba a ser infinita durante la planificación. Los controladores implementados por diseño no iban a tener un seguimiento del movimiento del robot como para corregir el error. Esto y la inercia hicieron que las simulaciones tuvieran una diferencia importante frente al plan trazado cuando se utilizaban velocidades cercanas a la máxima admitida por el Khepera III. Es por ello que se optó tomar velocidades mas pequeñas durante las pruebas en el simulador.

En esas condiciones, los resultados obtenidos en cada escenario fueron satisfactorios, logrando probar que el plan computado por el módulo de planificación de control es factible. No solo eso, sino que el tiempo que llevó al robot llegar al destino esta dentro de lo planeado y por debajo de lo que llevaría si el robot siguiese una estrategia de Giro y Avanzo.

Como conclusión final, se puede decir que con ciertas limitantes el planificador basado en Grid Search logró tener mejores resultados que Giro y Avanzo.

5.2. Trabajo a futuro

Actualmente los parámetros de los filtros son configurables, una de las líneas de trabajo futuras seria optimizar estos valores, utilizando la información del entorno, el robot y la cantidad de controladores. Vemos necesario que se tenga que realizar un primer análisis, para definir los valores óptimos de dichos filtros, logrando que el tiempo de procesamiento sea el menor posible y la solución obtenida sea la de menor tiempo de recorrido posible.

También se propone, continuar con la creación de un módulo de corrección de ejecución de la planificación de control, para mitigar los errores introducidos en la simulación. Esto permitiría obtener simulaciones más cercanas al control planificado. Este módulo, se encargaría de corregir la trayectoria del robot para aproximarla a la trayectoria planificada.

Por último. se plantea otra línea de trabajo a futuro, que es llevar esta solución a un entorno real. Esto nos permitiría experimentar las soluciones teóricas obtenidas en este trabajo, en un entorno real.

Bibliografía

[1] Avanta. Top 20 largest warehouses in the world. Sitio web. https://www.avantauk.com/top-14-largest-warehouses-in-the-world/.

[2] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[3] Mike Blackwell. The URANUS mobile robot. Sitio web. https://www.ri.cmu.edu/pub_files/pub3/blackwell_mike_1990_1/blackwell_mike_1990_1.pdf.

[4] Animesh Chhotray, Madhuri Pradhan, Krishna Pandey, and Dayal Parhi. *Kinematic Analysis of a Two-Wheeled Self-Balancing Mobile Robot*, volume 396, pages 87–93. "Springer India", 01 2016.

[5] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, May 2005.

[6] Cyberbotics. Webots: robot simulator. Sitio web. https://cyberbotics.com/ Accedida 01-08-2021.

[7] Cyberbotics. Webots: Supervisor. Sitio web. https://cyberbotics.com/doc/reference/supervisor Accedida 01-08-2021.

[8] E. W. Dijkstra. A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271, 1959.

[9] e puck.org. E-puck. Sitio web. http://www.e-puck.org/.

[10] Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974. http://dblp.uni-trier.de/db/journals/acta/acta4.html#FinkelB74.

[11] L. R. Ford. *Network Flow Theory.* RAND Corporation, Santa Monica, CA, 1956.

[12] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.

[13] K-Team. Khepera III. user manual. Sitio web. http://ftp.k-team.com/KheperaIII/UserManual/Kh3.Robot.UserManual.pdf.

[14] J.C. Latombe. *Robot Motion Planning: Edition en anglais*. The Springer International Series in Engineering and Computer Science. Springer, 1991. https://books.google.com.uy/books?id=Mbo_p4-46-cC.

[15] Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[16] U. Nehmzow. *Mobile Robotics: A Practical Introduction*. Applied computing. Springer London, 2003.

[17] Boulette's Institute of Technology. Subsumption architecture with robolab 2.5. Web site. https://www.convict.lu/Jeunes/Subsumption.htm and https://www.convict.lu/Jeunes/Subsumption_II.htm. Accedida 25-04-2022.

[18] R.P. Paul. *Robot Manipulators: Mathematics, Programming, and Control : the Computer Control of Robot Manipulators*. Artificial Intelligence Series. MIT Press, 1981.

[19] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., USA, 1984.

[20] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *PACIFIC JOURNAL OF MATHEMATICS*, 1990.

[21] Generation Robots. Pioneer 3-DX. User manual. Sitio Web. https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf.

[22] R. Siegwart, I.R. Nourbakhsh, and D. Scaramuzza. *Introduction to Autonomous Mobile Robots*. Intelligent Robotics and Autonomous Agents series. MIT Press, 2011. https://books.google.com.uy/books?id=4of6AQAAQBAJ.