PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

ESCUELA DE POSGRADO

DISEÑO DE UN SISTEMA DE CONTROL Y PLANEAMIETO DE TRAYECTORIA COORDINADO EN EL TIEMPO PARA MÚLTIPLES ROBOTS MÓVILES NO HOLONÓMICOS EN PRESENCIA DE OBSTÁCULOS

TESIS PARA OPTAR EL GRADO ACADÉMICO DE MAGÍSTER EN INGENIERÍA DE CONTROL Y AUTOMATIZACIÓN

AUTOR

Luis Enrique Dulanto Ramos

ASESOR

Ph.D. Antonio Manuel Morán Cárdenas

Noviembre, 2021

RESUMEN

La presente tesis tiene como objetivo diseñar un sistema de control y planeamiento de trayectoria coordinado para múltiples robots móviles no holonómicos en mapas con presencia de obstáculos variados. En esta se simula el control y planeamiento en modelos matemáticos de tipo bicicleta.

El sistema implementado consiste de tres partes, las cuales son el planeamiento de caminos, el generador de trayectorias y el control de seguimiento de trayectorias. El planeamiento de caminos se dividió en tres partes. En la primera parte se desarrolló el planeador local para un robot no holonómico, modificando el algoritmo Hybrid A*, de manera que utilice las ecuaciones movimiento circular del móvil en vez de las cinemáticas. Este algoritmo permite al robot encontrar los caminos que lo llevan de una configuración de posición y orientación inicial a una final en mapas con obstáculos variados. En la segunda parte se agregó al planeador local el planeamiento en el tiempo, combinando a este con el algoritmo de planeamiento de caminos en intervalos seguros (SIPP), el cual permite al robot evadir obstáculos en el tiempo. Finalmente, en la tercera parte se desarrolló el planeador global usando el algoritmo de búsqueda basada en conflictos (CBS), el cual resuelve los conflictos que se presentan entre los caminos de los móviles, imponiendo restricciones en el tiempo en el movimiento de cada uno de ellos. Por otro lado, el generador de trayectorias es desarrollado en una única parte, en la cual, se plantea la función de costo a optimizar, se calcula todos los gradientes y se plantea utilizar el algoritmo de descenso de gradiente de forma desacoplada para la optimización de trayectoria de cada móvil. Mientras que el desarrollo del sistema de control de seguimiento de trayectoria se dividió en dos partes. En la primera se linealiza el modelo matemático por extensión dinámica para sistemas flatness diferencial y en la segunda parte se desarrolla el controlador LQR de cada móvil que permite seguir las trayectorias de referencia deseadas.

Al término de la tesis se logra el planeamiento, generación de trayectoria y el control de seguimiento de trayectoria de hasta 10 móviles no holonómicos en mapas con obstáculos variados, evitando la colisión con los obstáculos del entorno y la colisión con otros móviles durante el planeamiento y la optimización de trayectoria. Así mismo, se verifica que el planeador es capaz de resolver conflictos en entornos propensos al atasco como mapas tipo T o H.

ÍNDICE GENERAL

INTRODUCCIÓN 1

Capítulo 1: Marco Teórico 2

1.1 Marco Problemático 2

1.2 Fundamento Teórico 3

1.2.1 Grafos 3

1.2.2 Algoritmos de Búsqueda 3

1.3 Estado del Arte 13

1.3.1 Planeamiento de Caminos Multi-Agente 13

1.3.2 Coordinación en el Tiempo 19

1.3.3 Generación de Trayectoria Multi-Robot 28

1.4 Propuesta de Solución 35

1.5 Objetivos de la Tesis 41

Capítulo 2: Modelamiento 42

2.1 Modelo Cinemático 42

2.2 Modelo Dinámico 44

2.2.1 Modelamiento Lateral 44

2.2.2 Modelamiento Longitudinal 46

2.3 Selección del modelo 47

Capítulo 3: Planeamiento 48

3.1 Representación del Mapa 48

3.2 Planeamiento Local 48

3.2.1 Algoritmo Hybrid A* 49

3.2.2 Diseño del Planeador Local 51

3.3 Planeamiento en el Tiempo 59

3.3.1 Planeamiento SIPP 60

3.3.2 Diseño del Planeador Local en el Tiempo 62

3.4 Planeamiento Global 66

3.4.1 Algoritmo CBS 66

3.4.2 Diseño del Planeador Global 67
Capítulo 4: Generación de Trayectoria y Control.....72
4.1 Generación de Trayectoria 72
4.1.1 Propuesta de Optimización.....72
4.1.2 Cálculo de Función de Costo.....74
4.1.3 Cálculo de Gradientes y Hessianas.....76
4.1.4 Algoritmo de Optimización.....79
4.2 Control.....82
4.2.1 Sistema Flatness Diferencial 82
4.2.2 Modelo Cinemático Flatness Diferencial.....83
4.2.3 Linealización por Extensión Dinámica 84
4.2.4 Seguimiento de Trayectoria.....85
4.3 Integración del Sistema 86
Capítulo 5: Simulación 87
5.1 Simulación del Planeamiento 87
5.1.1 Visualización del Planeamiento 87
5.1.2 Pruebas del Planeador Local.....88
5.1.3 Pruebas del Planeador Global 92
5.2 Simulación del Generador de Trayectoria 97
5.2.1 Visualización de Optimización de Trayectoria.....97
5.2.2 Pruebas del Generador de Trayectoria.....98
5.3 Simulación del Controlador.....103
5.3.1 Visualización de Seguimiento de Trayectoria 103
5.3.2 Pruebas del Controlador.....103
5.4 Simulación del Sistema Integrado 113
CONCLUSIONES.....121
RECOMENDACIONES 122
BIBLIOGRAFÍA 123

ÍNDICE DE FIGURAS

Figura 1.1: a) Robots en entorno tipo H, b) Robots en entorno tipo T [Elaboración propia]. 2
Figura 1.2: Representación de un grafo dirigido ponderado [Elaboración propia]. 3
Figura 1.3: Pasos del algoritmo de Dijkstra para llegar del nodo A al F [Elaboración propia]. 4
Figura 1.4: Reconstrucción de camino que lleva del nodo A al F [Elaboración propia]. 5
Figura 1.5: Pseudocódigo de algoritmo de Dijkstra [Elaboración propia]. 5
Figura 1.6: Pseudocódigo de algoritmo de reconstrucción de camino [Elaboración propia]. 5
Figura 1.7: Pseudocódigo de algoritmo A estrella [Elaboración propia]. 6
Figura 1.8:(a) Problema a resolver. (b) Solución de problema por A* [Elaboración propia]. 6
Figura 1.9: Paso 3 en algoritmo A* con nodo retirado coloreado en gris [Elaboración propia]. 7
Figura 1.10: Distancia de Manhattan a objetivo y costos del nodo [Elaboración propia]. 7
Figura 1.11: Primeras 6 iteraciones del algoritmo A*. La siguiente estimación del costo se muestra en celeste [Elaboración propia]. 8
Figura 1.12: Pseudocódigo de algoritmo CBS de alto nivel [10]. 9
Figura 1.13: Problema para ejemplificación de algoritmo CBS [10]. 9
Figura 1.14: Solución individual de cada agente [10]. 9
Figura 1.15: Nodo inicial del árbol de restricciones [Elaboración propia]. 10
Figura 1.16: Ejemplo de elección de mejor nodo [Elaboración propia]. 10
Figura 1.17: Búsqueda de conflictos entre caminos solución de los agentes [Elaboración propia]. 10
Figura 1.18: Generación de nuevos nodos con sus restricciones, caminos y costos [Elaboración propia]. 11
Figura 1.19: Solución del problema con algoritmo CBS [Elaboración propia]. 11
Figura 1.20: Árbol de restricciones para 3 agentes. [Elaboración propia]. 12
Figura 1.21: Árbol de restricciones alternativo para 3 agentes [Elaboración propia]. 12
Figura 1.22: Problema de planeamiento de caminos multi agente [Elaboración propia]. 14

Figura 1.23: Primera iteración según solución estándar de algoritmo A* [Elaboración propia]. 14
Figura 1.24: Pseudocódigo del algoritmo de detección de Independencia [13]. 15
Figura 1.25: Problema para ejemplificación de algoritmo de descomposición de operados [Elaboración propia].....16
Figura 1.26: Solución del problema según A* con descomposición de operador [Elaboración propia]. 16
Figura 1.27: Ejemplo de árbol de costos para 3 agentes [17].....17
Figura 1.28:(i) Problema a resolver. (ii) MDDs para agente 1. (iii) MDDs para agente 2 [17]. 17
Figura 1.29: (i) Fusión de MDDs. [17].....18
Figura 1.30: Pseudocódigo de algoritmo ICTS [17]. 18
Figura 1.31: (a) Camino prefijado de un móvil y un obstáculo móvil. (b) Espacio de configuración en tiempo del camino prefijado del móvil [Elaboración propia]. 19
Figura 1.32: Planeamiento en espacio de configuración en tiempo [Elaboración propia]. 20
Figura 1.33: Espacio de configuración en tiempo para plano bidimensional [18]. ...20
Figura 1.34: Ejemplo de movimiento de dos robots con caminos prefijados [19]. ...21
Figura 1.35: Perfiles de velocidad cuando: (a) Se alcanza Vmax. (b) No se alcanza vmax [19].....22
Figura 1.36: Perfiles de velocidad cuando: (a) Se alcanza v=0. (b) No se alcanza v=0 [19]. 23
Figura 1.37: Problema de ejemplo para planeamiento temporal [22]. 24
Figura 1.38: (a) Mapa del ejemplo representado en grafo. (b) Solución del problema, obtenida por algún método de planeamiento multi-agente, representado en una tabla [22]. 24
Figura 1.39: Grafo de planeamiento temporal del ejemplo [22]. 24
Figura 1.40: Grafo de planeamiento temporal aumentado del ejemplo [22]. 24
Figura 1.41: Red simple temporal del ejemplo [22]. 25
Figura 1.42: Grafo de distancia resultante de la red simple temporal del ejemplo [22]. 25
Figura 1.43: (a) Problema a resolver. (b) Estructura de grafo con forma de H [24]. 26
Figura 1.44: Grafo de Flujo de vehículos en el tiempo [24]. 26
Figura 1.45: Capacidad y costo de las aristas del grafo de flujo [Elaboración propia]. 27
Figura 1.46: (a) Solución por horizonte deslizante. (b) Solución por tiempo de llegada fijo [30]. 30

Figura 1.47:(a) Móvil 1 restringido a no pasar por B. (b) Restricción de pasar por D antes de A. (c) Restricción de pasar por D antes de A con un obstáculo [31]. 32
Figura 1.48: Algoritmo SCP para un solo móvil [34]. 34
Figura 1.49: Algoritmo solución desacoplado basado en SCP [34].. 34
Figura 1.50: Algoritmo iSCP [34]. 34
Figura 1.51: Planeamiento no holonómico con: a) Árbol de configuración, b) Lattice, c) Optimización [Elaboración propia]. 39
Figura 2.1: Representación de modelo cinemático del móvil [Elaboración propia]. .42
Figura 2.2: Representación de fuerzas laterales que actúan en el móvil [Elaboración propia]. 44
Figura 2.3: Ángulos de deslizamiento en rueda frontal y posterior [Elaboración propia]. 45
Figura 2.4: Representación de fuerzas longitudinales que actúan en un móvil [Elaboración propia]. 46
Figura 3.1: a) Mapa con obstáculos, b) Representación de mapa en grafos, c) Representación de mapa en grilla, d) representación de mapa con funciones potenciales [Elaboración propia]. 48
Figura 3.2: Representación de la exploración de un móvil usando Hybrid A Star [Elaboración propia]. 49
Figura 3.3: a) Distancia holonómica entre un móvil y su destino, b) Distancia no holonómica entre un móvil y su destino [Elaboración propia]. 49
Figura 3.4: a) Discretización de giros de robot no holonómico, b) Representación en nodos de configuraciones resultantes de giros de robot no holonómico [Elaboración propia]. 50
Figura 3.5: Representación de discretización de la posición y orientación de un móvil en el espacio cartesiano [Elaboración propia].] 50
Figura 3.6: Pseudocódigo de creación de matriz de distancia holonómica [Elaboración propia]. 51
Figura 3.7: Pseudocódigo de creación de nodos que describen celdas adjuntas [Elaboración propia]. 51
Figura 3.8: Visualización de matriz holonómica [Elaboración propia]. 51
Figura 3.9: Visualización de circunferencias y tangentes de un móvil en su punto de inicio y destino [Elaboración propia]. 52
Figura 3.10: Representación geométrica de rectas tangentes exteriores [Elaboración propia]. 52
Figura 3.11: Representación geométrica de rectas tangentes interiores [Elaboración propia]. 53

Figura 3.12: Representación geométrica de circunferencias tangentes agregadas [Elaboración propia]. 54
Figura 3.13: Pseudocódigo de distancia holonómica entre dos configuraciones [Elaboración propia]. 55
Figura 3.14: Representación geométrica del giro de un móvil [Elaboración propia].56
Figura 3.15: Pseudocódigo de cálculo de la siguiente configuración del móvil [Elaboración propia]. 57
Figura 3.16: a) Representación de colisión de móvil con el entorno, b) Nodos de configuración resultantes de los posibles movimientos del móvil [Elaboración propia]. 57
Figura 3.17: Pseudocódigo de obtención de siguientes nodos de configuración [Elaboración propia]. 57
Figura 3.18: Pseudocódigo de algoritmo Hybrid A star [Elaboración propia]. 58
Figura 3.19: Pseudocódigo de algoritmo de reconstrucción de solución [Elaboración propia]. 59
Figura 3.20: Pseudocódigo de planeador local [Elaboración propia]. 59
Figura 3.21: Adición de nodo de espera [Elaboración propia]. 59
Figura 3.22: a) Problema de robot holonómico con dos obstáculos móviles, b) Representación de creación de nodos SIPP [Elaboración propia].....60
Figura 3.23: Pseudocódigo de obtención de siguientes nodos SIPP [43].....61
Figura 3.24: Pseudocódigo de algoritmo SIPP [43]. 62
Figura 3.25: Representación de matriz de tiempo de ocupación por punto y de obtención de rango de ocupación de configuración [Elaboración propia]. 62
Figura 3.26: Pseudocódigo de obtención de rango de ocupación de configuración [Elaboración propia]. 62
Figura 3.27: Pseudocódigo de obtención de intervalo seguro de configuración [Elaboración propia]. 63
Figura 3.28: Pseudocódigo de creación de nodo SIPP inicial [Elaboración propia]. 63
Figura 3.29: Pseudocódigo de algoritmo de obtención de siguientes nodos SIPP [Elaboración propia]. 64
Figura 3.30: Pseudocódigo de algoritmo Hybrid A star con planeamiento SIPP [Elaboración propia]. 65
Figura 3.31: Pseudocódigo de algoritmo de reconstrucción de solución [Elaboración propia]. 65
Figura 3.32: Pseudocódigo de algoritmo del planeador local en el tiempo [Elaboración propia]. 65
Figura 3.33: Árbol de restricciones formado con nodos CBS [Elaboración propia]. .66

Figura 3.34: Colisión de móviles en instante t=5 [Elaboración propia].....67
Figura 3.35: Árbol de restricciones modificado [Elaboración propia]. 67
Figura 3.36: Pseudocódigo de inserción de restricciones [Elaboración propia]. 68
Figura 3.37: Pseudocódigo de búsqueda del primer conflicto [Elaboración propia].68
Figura 3.38: Pseudocódigo de solucionador de nodo CBS [Elaboración propia]. 69
Figura 3.39: Pseudocódigo de obtención de siguientes nodos CBS [Elaboración propia]. 69
Figura 3.40: Pseudocódigo de obtención de siguientes nodos CBS [Elaboración propia]. 70
Figura 3.41: Pseudocódigo de cálculo del costo de nodo CBS [Elaboración propia]. 70
Figura 3.42: Pseudocódigo del algoritmo de alto nivel [Elaboración propia]. 71
Figura 3.43: Pseudocódigo del planeador global [Elaboración propia]. 71
Figura 4.1: Representación de fuerzas durante la optimización [Elaboración propia]. 73
Figura 4.2: Representación de las fuerzas aplicadas por un móvil 2 a un móvil 1 [Elaboración propia]. 73
Figura 4.3: Adición de circunferencias en punto de espera [Elaboración propia]. ...73
Figura 4.4: a) Trayectorias con tiempos distintos, b) Trayectorias interpoladas en el tiempo con corrección en puntos de espera [Elaboración propia]. 80
Figura 4.5: Optimización de trayectorias resultantes del planeamiento [Elaboración propia]. 81
Figura 4.6: Pseudocódigo de algoritmo de optimización de trayectorias [Elaboración propia]. 81
Figura 4.7: Diagrama de bloques de controlador [Elaboración propia]. 86
Figura 4.8: Diagrama de integración del planeamiento, generación de trayectoria y control de seguimiento de trayectoria [Elaboración propia]. 86
Figura 5.1: Elementos de Simulación de planeamiento [Elaboración propia]. 87
Figura 5.2: Planeamiento de un móvil para mapa a) de obstáculos no convexos, b) de obstáculos redondos, c) de obstáculos rectangulares, d) de obstáculos de barras, e) de obstáculos de puntos dispersos, f) de estructuras curvas, g), de estructuras rectas y h) sin obstáculos [Elaboración propia]. 88
Figura 5.3: Planeamiento de móvil en mapa de barras trasversales de 2000x2000 [Elaboración propia]. 90
Figura 5.4: Tiempo de solución Vs Dimensión de mapa [Elaboración propia]. 91
Figura 5.5: Movimiento de móviles en a) dirección directa y b) dirección opuesta [Elaboración propia]. 92

Figura 5.6: Tiempo de Solución Vs Número de móviles [Elaboración propia]. 93
Figura 5.7: Número de Restricciones Vs Número de móviles [Elaboración propia]. 93
Figura 5.8: Número de nodos CBS Vs Número de móviles [Elaboración propia]. ...93
Figura 5.9: a) Inversión de orden en mapa tipo T, b) Intercambio de posición en mapa tipo T, c) Cruce de caminos en mapa tipo I y d) Pasillo ocupado en mapa tipo H [Elaboración propia]. 94
Figura 5.10: Planeamiento en mapa a) de obstáculos no convexos, b) de obstáculos rectangulares, c) de obstáculos redondos, d) de obstáculos de barras, e) de obstáculos de puntos dispersos, f) de estructuras curvas y g) de estructuras rectas [Elaboración propia]. 95
Figura 5.11: Visualización de optimización de trayectoria de un móvil [Elaboración propia]. 97
Figura 5.12: Visualización de optimización de trayectoria de múltiples móviles [Elaboración propia]. 98
Figura 5.13: Efecto de optimización en corrección de posición final y restricción de giro en puntos de espera [Elaboración propia]. 98
Figura 5.14: Resultados del planeamiento y tiempos transcurridos [Elaboración propia]. 99
Figura 5.15: Efecto de optimización en evasión de obstáculos del entorno [Elaboración propia]. 99
Figura 5.16: Efecto de optimización en minimización de longitud de camino [Elaboración propia]. 99
Figura 5.17: Efecto de optimización en la evasión de colisiones con otros móviles [Elaboración propia]. 100
Figura 5.18: Optimización de trayectoria de móviles en mapa de obstáculos rectangulares [Elaboración propia]. 100
Figura 5.19: Resultados del planeamiento y optimización de trayectoria [Elaboración propia]. 101
Figura 5.20: Tiempo de optimización vs número de vehículos [Elaboración propia]. 102
Figura 5.21: Visualización de simulación del control [Elaboración propia]. 103
Figura 5.22: Visualización de simulación del movimiento del móvil [Elaboración propia]. 103
Figura 5.23: Control de seguimiento de trayectoria de móvil en mapa de obstáculos rectangulares [Elaboración propia]. 104
Figura 5.24: Gráfico de posición X vs tiempo [Elaboración propia]. 104
Figura 5.25: Gráfico de posición Y vs tiempo [Elaboración propia]. 105

Figura 5.26: Gráfico de orientación vs tiempo [Elaboración propia].....105
Figura 5.27: Gráfico de velocidad vs tiempo [Elaboración propia].....106
Figura 5.28: Gráfico de aceleración vs tiempo [Elaboración propia].....106
Figura 5.29: Gráfico de giro del timón vs tiempo [Elaboración propia].....107
Figura 5.30: Simulación del movimiento del móvil en mapa de obstáculos rectangulares [Elaboración propia].....107
Figura 5.31: Seguimiento de trayectoria en mapa tipo H con situación de pasillo ocupado [Elaboración propia].....108
Figura 5.32: Resultados del planeamiento de móviles en mapa tipo H [Elaboración propia].....108
Figura 5.33: Gráficas de posición y orientación vs tiempo del control del agente 1 en mapa tipo H [Elaboración propia].....109
Figura 5.34: Gráfica de velocidad vs tiempo del control del agente 1 en mapa tipo H [Elaboración propia].....110
Figura 5.35: Gráficas de aceleración y giro del timón vs tiempo del control del agente 1 en mapa tipo H [Elaboración propia].....110
Figura 5.36: Gráficas de posición, orientación y velocidad vs tiempo del control del agente 0 en mapa tipo H [Elaboración propia].....111
Figura 5.37: Gráficas de aceleración y giro del timón vs tiempo del control del agente 0 en mapa tipo H [Elaboración propia].....112
Figura 5.38: Simulación del movimiento de los móviles en mapa tipo H [Elaboración propia].....112
Figura 5.39: Seguimiento de trayectoria de 10 móviles en espacio vacío [Elaboración propia].....113
Figura 5.40: Error de seguimiento de trayectoria Vs tiempo [Elaboración propia]. 113
Figura 5.41: Seguimiento de trayectoria de 10 móviles en mapa de puntos dispersos [Elaboración propia].....114
Figura 5.42: Seguimiento de trayectoria de 15 móviles en mapa de barras [Elaboración propia].....115
Figura 5.43: Error de seguimiento de trayectoria en mapa de obstáculos redondos [Elaboración propia].....115
Figura 5.44: Seguimiento de trayectoria con exclusión de puntos de curvatura máxima en mapa de obstáculos redondos [Elaboración propia].....116
Figura 5.45: Corrección de posición del punto final en móviles sin curvaturas pronunciadas cerca al punto final [Elaboración propia].....117
Figura 5.46: Seguimiento de trayectoria de 10 móviles en mapa de obstáculos no convexos [Elaboración propia].....117

Figura 5.47: Seguimiento de trayectoria de 10 móviles en mapa de obstáculos rectangulares [Elaboración propia]. 118
Figura 5.48: Seguimiento de trayectoria de 10 móviles en mapa de estructuras curvas [Elaboración propia]. 118
Figura 5.49: Seguimiento de trayectoria de 8 móviles en mapa de estructuras rectas [Elaboración propia]. 119

ÍNDICE DE TABLAS

Tabla 1: Enfoque solución de planeadores multi agente [Elaboración propia].....35
Tabla 2: Número de nodos generados por planeador multi agente [Elaboración propia]. 37
Tabla 3: Resultados del planeamiento en diferentes tipos de mapa, luego de 100 ejecuciones aleatorias por mapa [Elaboración propia]. 89
Tabla 4: Resultados de planeamiento ante variación de tamaño del mapa, luego de 10 ejecuciones por mapa [Elaboración propia]. 91
Tabla 5: Resultados del planeamiento ante incremento de agentes [Elaboración propia]. 92
Tabla 6: Resultados de planeamiento en entornos restringidos [Elaboración propia]. 94
Tabla 7: Resultados de planeamiento en mapas con obstáculos variados [Elaboración propia]. 96
Tabla 8: Resultados de pruebas de optimización de trayectoria [Elaboración propia]. 101
Tabla 9: Resumen de pruebas realizadas del sistema integrado [Elaboración propia]. 119

AGRADECIMIENTOS

Agradezco al Dr. Morán por su asesoramiento en esta tesis y por sus conocimientos compartidos durante mi etapa de estudios, así como a todos los profesores de esta maestría quienes han contribuido en mi formación académica. Agradezco también a mis compañeros de estudios que hicieron de esta una experiencia amena.

INTRODUCCIÓN

La utilización simultánea de múltiples robots móviles ha ido en incremento durante los últimos años en las diferentes industrias. La industria del comercio electrónico utiliza robots móviles para el almacenaje y despacho de productos; ejemplo de ello son Amazon, con más de 100000 robots móviles operando en sus 25 centros principales [1], y Alibaba, que en octubre de 2018 abrió un almacén con más de 700 robots móviles para despacho [2]. Así mismo, en la industria minera se tiene el caso de Rio Tinto [3], el cual utiliza más de 130 camiones autónomos para el transporte de carga. Mientras que, en la industria de electrodomésticos, se presenta el uso de múltiples robots móviles para la limpieza, como Dyson 360 [4] o Roomba [5]. De igual manera, en la industria del entretenimiento, empresas como Intel [6] o SKYMAGIC [7] utilizan múltiples drones para realizar shows de luces en el cielo.

La presente tesis tiene como finalidad diseñar un sistema de control y planeamiento de trayectoria multi robot con coordinación en el tiempo para robots móviles no holonómicos. Esta tesis está dividida en cuatro capítulos. En el capítulo 1, denominado “Marco Teórico”, se desarrolla el marco problemático de la tesis, el fundamento teórico, el estado del arte en el planeamiento de caminos multi-agente, el estado del arte en la coordinación en el tiempo, el estado del arte en generación de trayectoria multi robot y se desarrolla los objetivos de la tesis. En el capítulo 2, denominado “Modelamiento” se modela la cinemática y dinámica del modelo de robot móvil no holonómico a utilizar. En el capítulo 3, denominado “Planeamiento”, se diseña al planeador local, al planeador local en el tiempo y al planeador global. En el capítulo 4 denominado “Generación de Trayectoria y Control” se diseña al generador de trayectorias por optimización y se diseña el control de seguimiento de trayectoria del móvil. En el capítulo 5, denominado “Simulación”, se prueba el sistema en diferentes tipos de mapas y se verifica su funcionamiento.

Capítulo 1: Marco Teórico

1.1 Marco Problemático

Las aplicaciones mencionadas en la introducción implican la planificación del movimiento de múltiples robots móviles en simultáneo, en las cuales se debe mover a todos los robots de una posición origen a una final, tal que estos no colisionen entre sí, ni con los obstáculos del mapa. La solución a este problema varía según la complejidad del mismo [8][9][10], la cual aumenta acorde a las restricciones impuestas, como pueden ser las restricciones cinemáticas del móvil o las restricciones físicas del entorno. Las primeras restringen el movimiento del móvil, forzando a que este deba seguir caminos suaves, mientras que las segundas pueden ocasionar el atasco de los robots en el mapa. Por ejemplo, si los robots de Figura 1.1 a) planean de forma reactiva, usando tan solo un sistema de evasión de colisiones, ambos se desplazarían hacia su destino y quedarían atrapados en el centro del mapa, al no haber espacios libres para evitar la colisión; mientras que si los robots de Figura 1.1 b) planean por orden de prioridad, planeando primero el movimiento del robot 1 y luego del robot 2, solo el robot 1 llegaría a su destino. Nótese que, en ambos casos, la solución correcta implica la coordinación en el tiempo; en el caso de a), uno de los robots debe esperar a que el otro deje libre el centro del mapa y luego moverse hacia su destino; mientras que, en el caso de b), el robot 1 debe moverse a la parte superior del mapa y esperar a que pase el robot 2, para luego moverse hacia su destino.

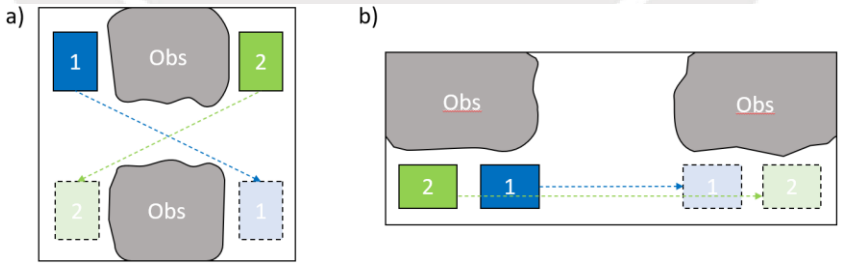


Figura 1.1: a) Robots en entorno tipo H, b) Robots en entorno tipo T [Elaboración propia].

Sabiéndose que la gran mayoría de robots móviles no pueden moverse de manera instantánea en cualquier dirección, ya que poseen restricciones cinemáticas; es decir, son no holonómicos; y que, además, las disposiciones y formas de obstáculos del mapa pueden generar situaciones de atasco, se busca diseñar un sistema de planeamiento de trayectoria multi robot coordinado en el tiempo para vehículos no holonómicos, tal que permita a los móviles navegar sin atascos en mapas con presencia de obstáculos variados.

1.2 Fundamento Teórico

1.2.1 Grafos

Un grafo se define como un par ordenado G = (V, E), donde V es un conjunto no vacío de elementos denominados vértices o nodos y E es un subconjunto de dos elementos de V, donde los elementos de E son denominados aristas [11]. Cuando dichas aristas poseen un valor numérico asociado al costo de transitar de un nodo a otro se le llama grafo ponderado. Mientras que, si las aristas de G se encuentran direccionadas, el grafo es llamado grafo dirigido. En Figura 1.2 se muestra un ejemplo de grafo dirigido ponderado.

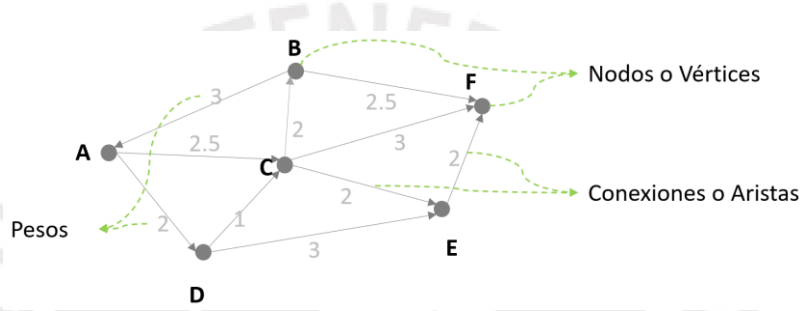


Figura 1.2: Representación de un grafo dirigido ponderado [Elaboración propia].

1.2.2 Algoritmos de Búsqueda

a) Algoritmo de Dijkstra

El algoritmo de Dijkstra es un algoritmo de búsqueda en grafos que consiste en obtener el camino de menor costo que conecta a dos nodos de un grafo, el cual hace uso de una lista ordenada de nodos, llamada lista de nodos abiertos para explorar el grafo.

La idea del algoritmo consiste en asignarle costos de transición o desplazamiento a los nodos del grafo, los cuales resultan de la acumulación del costo de transitar cada arista. El algoritmo explora cada nodo, partiendo del nodo inicial, propagándose por los nodos vecinos de cada nodo que explora y cuando reexplora un nodo y encuentra que el costo de llegar a ese nodo es menor al que había estimado antes actualiza el costo del nodo. El algoritmo explora nodo por nodo hasta encontrar al nodo destino, y va guardando al nodo padre de cada nodo para rastrear el camino que llevo hasta el nodo. Cuando el nodo destino es evaluado, se para la para la búsqueda y se reconstruye el camino recorrido, rastreando de manera iterativa a los nodos padre que llevaron a evaluar al nodo destino.

Por ejemplo, para encontrar el camino de menor costo del nodo A al F del grafo de Figura 1.3, el algoritmo empezaría inicializando todos los costos a infinito, a excepción del costo del nodo inicial A, el cual sería igual a 0, ya que el recorrido del nodo A hacia A es 0. Luego de inicializado los valores, se agregaría al nodo A la lista de abiertos (ver a), enseguida buscaría en la lista de abiertos al nodo de menor costo, el cual es el mismo nodo A, pues es el único nodo en la lista, y se lo retira de la lista de abiertos. Seguidamente, se explora los nodos vecinos del nodo elegido (ver b) y se calcula el costo de llegar a dichos nodos pasando por el nodo elegido (esto sería el costo de A más el costo de transición de pasar de A al nodo explorado indicado por el peso de la arista), luego se actualiza el costo de llegar a dichos nodos en caso los costos calculados sean menores a los costos estimados anteriores (como el costo anterior es igual a infinito, se reemplaza con los nuevos costos), luego se agrega dichos nodos a la lista de abiertos. Nuevamente, se busca al nodo de menor costo de la lista (el nodo D) y se lo retira de la lista de abiertos, luego se explora sus nodos vecinos (ver c), actualizando el costo de llegar a dichos nodos en caso el costo sea menor al previamente estimado (en el caso del nodo C no se reemplaza pues el costo es mayor).

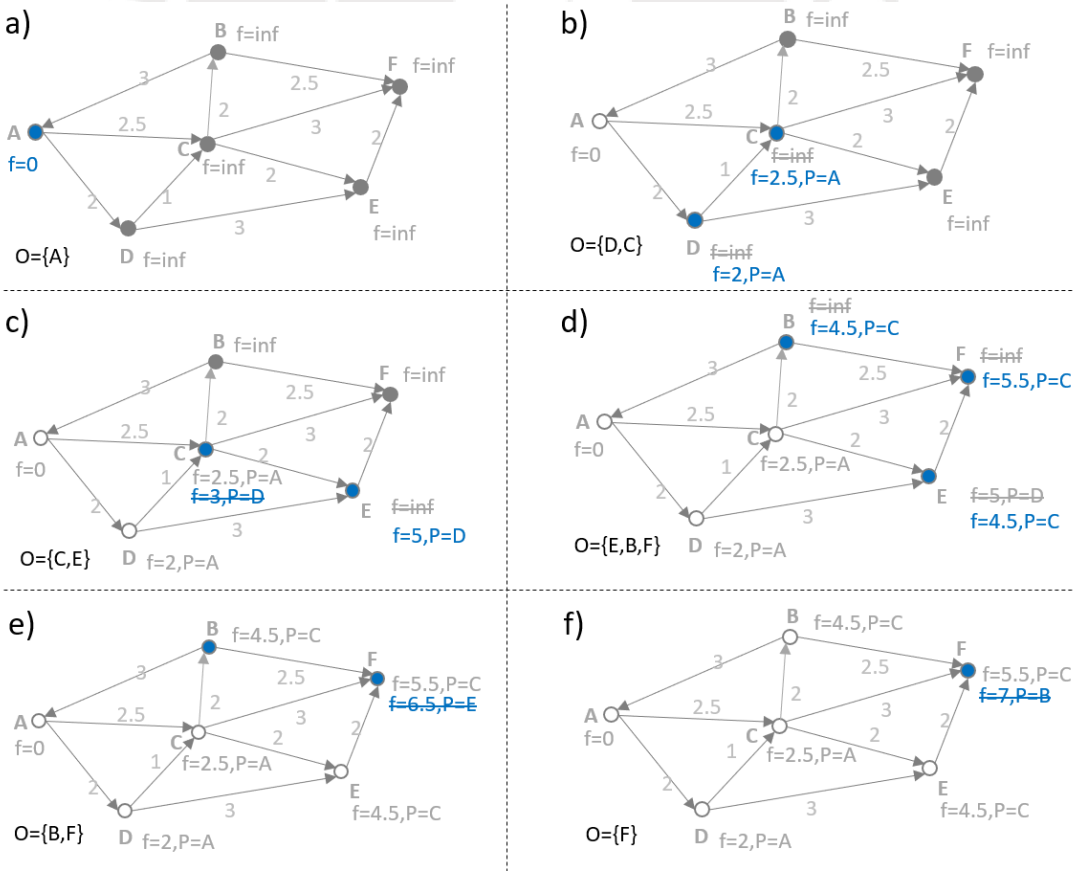


Figura 1.3: Pasos del algoritmo de Dijkstra para llegar del nodo A al F [Elaboración propia].

Luego el proceso se repite hasta que no queden más nodos en la lista de abiertos o hasta que se elija de la lista como nodo de menor costo al nodo destino. Una vez llegado al nodo destino se rastrea a los nodos padre que llevaron al nodo destino, ver Figura 1.4. Nótese de Figura 1.3, que los nodos padres son actualizados cada vez que se actualiza el costo del nodo.

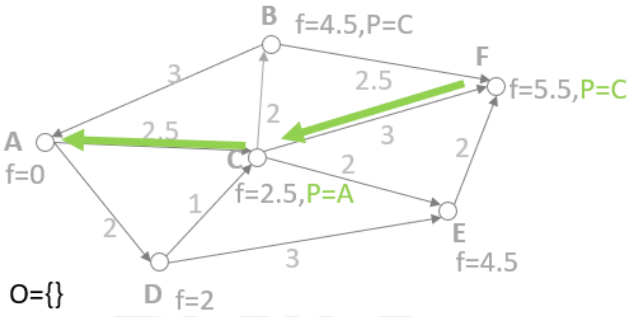


Figura 1.4: Reconstrucción de camino que lleva del nodo A al F [Elaboración propia].

El pseudocódigo del algoritmo de Dijkstra se muestra en Figura 1.5.

```
function Dijkstra(nodoInicial):
    crea lista de ABIERTOS vacía
    agrega nodoInicial a ABIERTOS
    mientras ABIERTOS no está vacio:
        nodoDeMenorCosto = obtenNodoDeMenorCostoDe(Abiertos)
        remueve nodoDeMenorCosto de ABIERTOS

        si nodoDeMenorCosto cumple con el objetivo:
            salir del bucle;

        nodosVecinos = obtenNodosVecinosDe(nodoDeMenorCosto)    // nodos hijos
        para cada nodoVecino de nodosVecinos:
            costoDeTransicion = obtenCostodeIrDe(nodoDeMenorCosto, nodoVecino)
            f_tentativoDeNodoVecino = f[nodoDeMenorCosto] + costoDeTransicion
            si f_tentativoDeNodoVecinoes es menor que f[nodoVecino]:
                f[nodoVecino] = f_tentativoDeNodoVecino
                rastreaNodoPadre[nodoVecino] = nodoDeMenorCosto
            agrega nodoVecino a ABIERTOS

    crea lista SOLUCION vacía
    reconstruyeSolucion(nodoDeMenorCosto,rastreaNodoPadre,SOLUCION)
    retorna SOLUCION
```

Figura 1.5: Pseudocódigo de algoritmo de Dijkstra [Elaboración propia].

Mientras que el pseudocódigo del algoritmo recursivo que reconstruye el camino se muestra en Figura 1.6.

```
function reconstruyeSolucion(nodo,rastreaNodoPadre,SOLUCION):
    agrega nodo a SOLUCION;
    si rastreaNodoPadre[node] no esta vacío:
        reconstruyeSolucion(rastreaNodoPadre[node],rastreaNodoPadre,SOLUCION)
    retorna
```

Figura 1.6: Pseudocódigo de algoritmo de reconstrucción de camino [Elaboración propia].

b) Algoritmo A estrella

El algoritmo de A estrella (A*) está basado en el algoritmo de Dijkstra, donde la única diferencia que existe entre este y el algoritmo de Dijkstra es que el costo total "f" del nodo se calcula como el costo "g" acumulado de llegar hasta el nodo evaluado más el costo "h" estimado de lo que tomaría llegar a la meta desde el nodo evaluado. El valor de "h" es calculado de manera heurística y no necesariamente es igual el costo real de lo que falta recorrer, pues solo es un valor estimado. El pseudocódigo del algoritmo de A estrella se muestra en Figura 1.7.

```
function aStar(nodoInicial):
    crea lista de ABIERTOS vacía
    agrega nodoInicial a ABIERTOS
    mientras ABIERTOS no está vacio:
        nodoDeMenorCosto = obtenNodoDeMenorCostoDe(Abiertos)
        remueve nodoDeMenorCosto de ABIERTOS

        si nodoDeMenorCosto cumple con el objetivo:
            salir del bucle;

        nodosVecinos = obtenNodosVecinosDe(nodoDeMenorCosto)    //nodos hijos
        para cada nodoVecino de nodosVecinos:
            costoDeTransicion = obtenCostodeIrDe(nodoDeMenorCosto, nodoVecino)
            g_tentativoDeNodoVecino = g[nodoDeMenorCosto] + costoDeTransicion
            si g_tentativoDeNodoVecinoes es menor que g[nodoVecino]:
                g[nodoVecino] = g_tentativoDeNodoVecino        // costo de llegar a nodo vecino
                h[nodoVecino] = obtenCostoHeuristicoEstimadoDeIrDe(nodoVecino,objetivo)
                f[nodoVecino] = g[nodoVecino] + h[nodoVecino]  // costo estimado de llegar al
                                                               // objetivo pasando por nodoVecino
                rastreaNodoPadre[nodoVecino] = nodoDeMenorCosto
            agrega nodoVecino a ABIERTOS

    crea lista SOLUCION vacía
    reconstruyeSolucion(nodoDeMenorCosto,nodoPadre,SOLUCION)
    retorna SOLUCION
```

Figura 1.7: Pseudocódigo de algoritmo A estrella [Elaboración propia].

Para ejemplificar el algoritmo, se utilizará el problema mostrado en Figura 1.8 (a). En este, se requiere que el móvil (el círculo azul) llegue a su destino (el cuadro naranja), por el camino más corto posible. La solución a este problema, por medio de dicho algoritmo se muestra en Figura 1.8 (b).

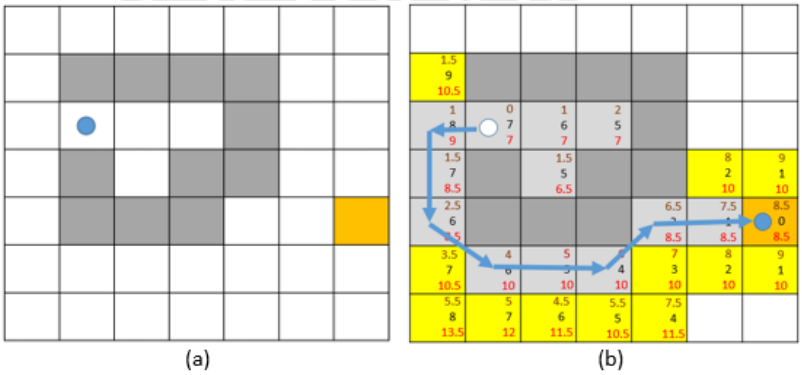


Figura 1.8:(a) Problema a resolver. (b) Solución de problema por A [Elaboración propia].*

El algoritmo, que resuelve dicho problema procede de la siguiente manera:

1) Crea una lista de nodos Abiertos (línea 2).
2) Coloca al nodo de inicio en la lista de nodos Abiertos (línea 3).
3) Busca al nodo de menor costo (nodoDeMenorCosto) en la lista de abiertos y se lo retira de la lista, visto como una celda en gris en Figura 1.9 (línea 5 y 6).

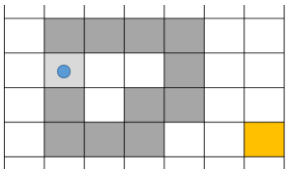


Figura 1.9: Paso 3 en algoritmo A con nodo retirado coloreado en gris [Elaboración propia].*

4) Si el nodo nodoDeMenorCosto es el nodo objetivo, termina el algoritmo (línea 5 y 6) y reconstruye el camino solución (línea 23 y 24).
5) Explora los nodos vecinos de nodoDeMenorCosto, posiciones adyacentes a este nodo (línea 11).
6) Para cada nodo vecino, calcula el costo g_tentativo de llegar al nodo vecino pasando por el nodo de menor costo, usando el costo de transición de ir del nodo de menor costo al nodo vecino (línea 12, 13 y 14):

$$g_{tentativo} = g[nodoDeMenorCosto] + costoDeTransición \quad (1)$$

7) Si $g_{tentativo}$ es menor a $g[nodoVecino]$ (línea 15 y 16), asigna:

$$g[nodoVecino] = g_{tentativo} \quad (2)$$

Luego calcula el costo total estimado de llegar al objetivo (línea 18), sumándole a $g[nodoVecino]$ el costo estimado h de ir desde el nodo vecino al objetivo:

$$f[nodoVecino] = g[nodoVecino] + h \quad (3)$$

Para garantizar que el camino obtenido sea el de menor costo, la heurística h debe ser admisible, es decir, no debe sobreestimar el costo real de llegar al objetivo. Luego, cualquier función que cumpla con ello puede ser utilizada como costo heurístico, como la distancia euclidiana o la de Manhattan. En este caso se usa la distancia de Manhattan al objetivo. En Figura 1.10, se muestra en negro, el costo h calculado por distancia de Manhattan; en marrón, el costo de transición de llegar del nodo de menor costo al vecino y en rojo, el costo total.

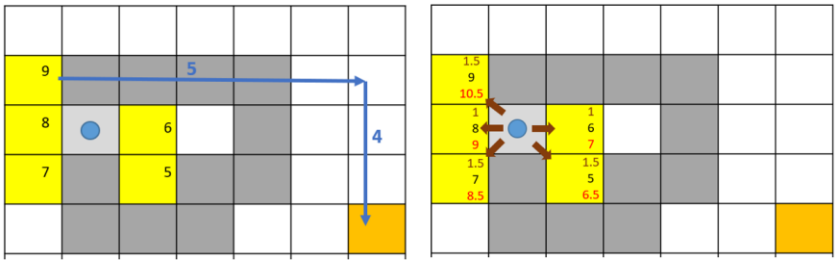


Figura 1.10: Distancia de Manhattan a objetivo y costos del nodo [Elaboración propia].

8) Agrega los nodos vecinos a la lista de abiertos (línea 21). En Figura 1.11, se muestran las 6 primeras iteraciones del algoritmo.

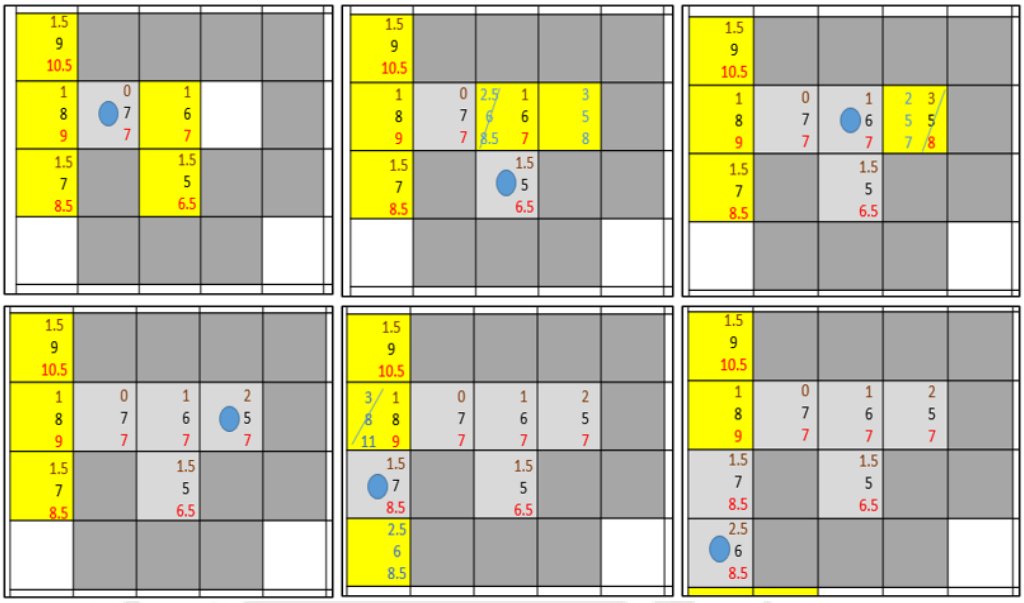


Figura 1.11: Primeras 6 iteraciones del algoritmo A. La siguiente estimación del costo se muestra en celeste [Elaboración propia].*

9) Volver al paso 3 (línea 4).

c) Algoritmo de Búsqueda Basada en Conflictos

El algoritmo de CBS, planteado en [10], se basa en planear los caminos de cada agente de manera independiente, cumpliendo con un árbol de restricciones de posiciones en el tiempo para cada agente. Estas restricciones son conflictos que se deben evitar, los cuales son las posiciones de los otros agentes en el tiempo. El algoritmo consta de dos partes, las cuales son llamadas “búsqueda de alto nivel” y “búsqueda de bajo nivel”. La búsqueda de alto nivel, es la búsqueda de restricciones de posiciones en el tiempo, la cual termina formando un árbol de restricciones; y la búsqueda de bajo nivel es la búsqueda de caminos en el tiempo que realiza cada agente de manera independiente para llegar a su objetivo, tal que estos caminos cumplan con las restricciones impuestas por el árbol de restricciones. El algoritmo de búsqueda de bajo nivel tiene como nodo la posición del agente en el tiempo; es decir, es una tupla de 3 valores (x, y, t). Este puede ser cualquier algoritmo de búsqueda, como A*, IDA* o algún otro. Por otro lado, los nodos del árbol de restricciones, son restricciones de posición y tiempo; es decir una tupla de la forma $\{(a_i, v, t), (a_j, v, t), (a_k, v, t)\}$ donde esto significaría que en el instante t, ninguno de los agentes a_i, a_j o a_k puede ocupar el vértice $v(x, y)$. El pseudocódigo del algoritmo CBS de alto nivel, se muestra en Figura 1.12.

Algorithm 1: high-level of CBS

Input: MAPF instance

1 $R.constraints = \emptyset$
2 $R.solution$ = find individual paths using the low-level()
3 $R.cost = SIC(R.solution)$
4 insert R to OPEN
5 **while** OPEN *not empty* **do**
6 P ← best node from OPEN // *lowest solution cost*
7 Validate the paths in P until a conflict occurs.
8 **if** *P has no conflict* **then**
9 **return** P.solution // *P is goal*
10 C ← first conflict (a_i, a_j, v, t) in P
11 **foreach** *agent* a_i *in C* **do**
12 A ← new node
13 A.constriants ← P.constriants + (a_i, s, t)
14 A.solution ← P.solution.
15 Update A.solution by invoking low-level(a_i)
16 $A.cost = SIC(A.solution)$
17 Insert A to OPEN

Figura 1.12: Pseudocódigo de algoritmo CBS de alto nivel [10].

Para explicar el algoritmo, utilizaremos el ejemplo de Figura 1.13, en la cual se muestra un par de ratones que deben llegar a sus objetivos respectivos, los cuales son el queso 1 y el queso 2.

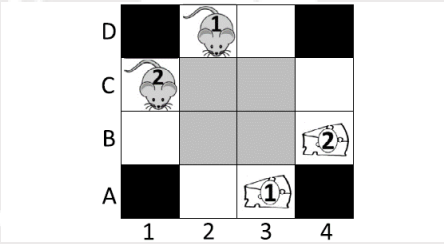


Figura 1.13: Problema para ejemplificación de algoritmo CBS [10].

El algoritmo procede de la siguiente manera:

1) Se crea el nodo CBS inicial sin ninguna restricción (línea 1).
2) Para cada agente, se halla el camino que lo lleva a su destino, usando su algoritmo de bajo nivel, sin considerar a otros agentes, ver Figura 1.14. Luego, se almacena ambos caminos en el nodo CBS creado (línea 2).

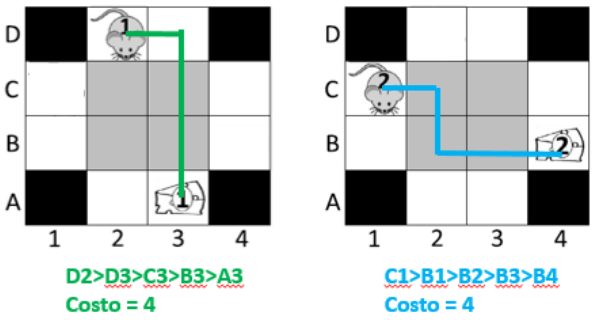


Figura 1.14: Solución individual de cada agente [10].

3) Se halla el costo total del nodo CBS sumando los costos de recorrer cada camino independiente (SIC) e incluirlo al nodo CBS (línea 3). En Figura 1.15 sería 4 del primer camino más 4 del segundo (8 en total).

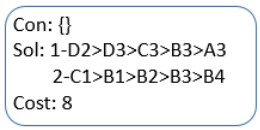


Figura 1.15: Nodo inicial del árbol de restricciones [Elaboración propia].

4) Se inserta el nodo CBS en la lista de abiertos (línea 4).
5) Mientras la lista de Abiertos no esté vacía se repite del paso 6 al 15 (línea 5).
6) Se toma el mejor nodo de menor costo “P” de la lista de abiertos y se lo retira de la lista (línea 6). De existir más de un nodo con el mismo costo, se elige entre ellos al de menor cantidad de restricciones. Ver Figura 1.16.

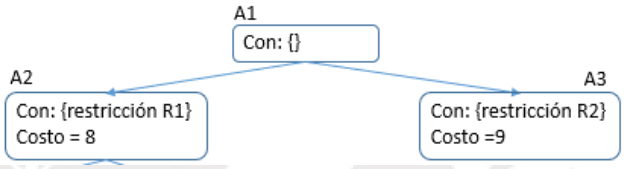


Figura 1.16: Ejemplo de elección de mejor nodo [Elaboración propia].

7) Se valida los caminos del nodo elegido hasta que ocurra un conflicto (línea 7). Ver Figura 1.17.

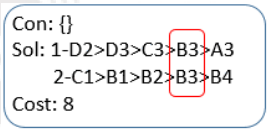


Figura 1.17: Búsqueda de conflictos entre caminos solución de los agentes [Elaboración propia].

8) Si el nodo elegido no tiene conflictos, se retorna el nodo elegido como el nodo que contiene la solución del problema (líneas 8 y 9).
9) De encontrase un conflicto “C”, se almacena en una estructura o vector de la forma $(a_i, a_j, \mathrm{p}, t)$, la cual indica que los agentes a_i y a_j colisionan en la posición p en el instante t (línea 10).
10) Para cada agente involucrado en el conflicto “C”, se crea un nuevo nodo “A” (línea 11). Ver Figura 1.18.
11) Para cada nodo “A” creado, se agrega la restricción de no estar en el instante y posición donde ocurre el conflicto (a_i, v, t) y se agrega las restricciones del nodo anterior (línea 13). Ver Figura 1.18.
12) Para cada nodo “A”, encontrar la solución (el camino del ratón 1 y del ratón 2 hallados de manera independiente con el algoritmo de bajo nivel). Y agregar la solución al nodo (líneas 14 y 15). Ver Figura 1.18.

13) Para cada nodo “A”, se halla el costo total de la solución usando SIC, suma independiente de costos (línea 16). Ver Figura 1.18.

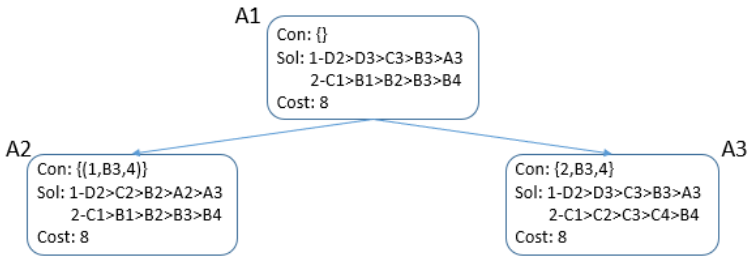


Figura 1.18: Generación de nuevos nodos con sus restricciones, caminos y costos [Elaboración propia].

14) Se inserta cada nodo “A” en la lista de abiertos (línea 17).
15) Se vuelve al paso 5 (línea 5).

De proseguir con los pasos especificados se llegaría al árbol de restricciones de Figura 1.19.

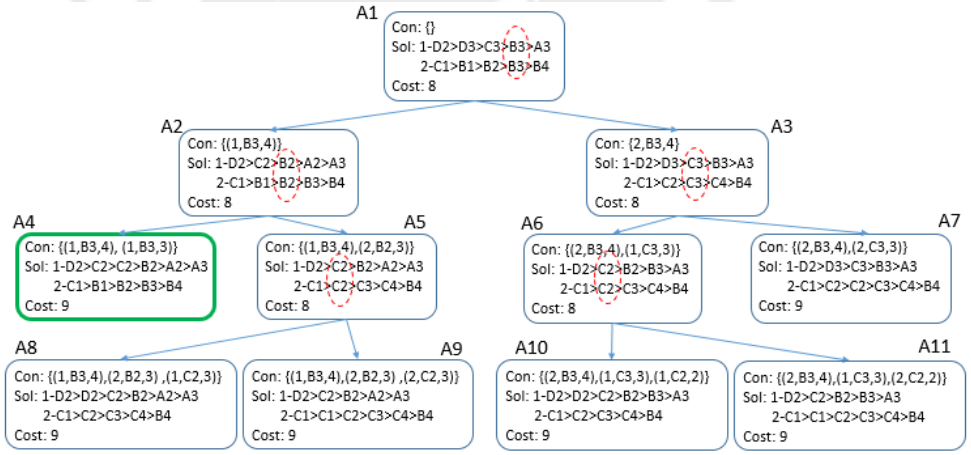


Figura 1.19: Solución del problema con algoritmo CBS [Elaboración propia].

Donde inicialmente, se crearía el nodo A1 y se mandaría a “Abiertos” (Abiertos = {A1}). Luego se elige el mejor nodo de “Abiertos” (P = A1). Se valida el nodo para ver si existen conflictos, pero como sí existen, se crean los nodos A2 y A3, heredando las restricciones de A1 y agregándolos a “Abiertos” (Abiertos = {A1, A2, A3}). De la lista de abiertos se elige al mejor nodo; como tanto A2 como A3 tienen el mismo costo y misma cantidad de restricciones, se procede por defecto con el nodo de la izquierda (P = A2). Como existen conflictos en A2, se crean A4 y A5, heredando las restricciones de A2 y añadiéndolos a “Abiertos” (Abiertos = {A1, A2, A3, A4, A5}). Luego se elige al mejor nodo de Abiertos. Como A3 tiene el mismo costo de A5, pero menos restricciones, se procede con A3 (P = A3). Como en A3 hay conflictos, se generan A6 y A7 y se los agrega a la lista de “Abiertos”. Como A5

y A6 tiene el mismo costo y mismo número de restricciones, se procede con A5 (P = A5). Como existen conflictos en A5 se crean A8 y A9. Luego se elige al mejor nodo de la lista de “Abiertos” (P = A6). Y como en este existen conflictos, se crean A10 y A11 y se los agrega a la lista de abiertos. Luego se elige al mejor nodo de la lista de abiertos. Como A4 y A7 tienen el mismo costo y misma cantidad de restricciones se elige al de la izquierda (A4). Luego se valida el nodo A4 y como no existen conflictos, se retorna ese nodo como la solución al problema.

De existir más de un agente que coincide en un mismo vértice, se crearía un nodo por agente, donde cada nodo tiene la restricción de que los otros agentes no ocupen la misma posición además de las heredadas. Por ejemplo, si fueran 3 agentes que quieren ocupar el nodo B3 en el instante 4, se crearía 3 nodos hijos. Ver Figura 1.20.

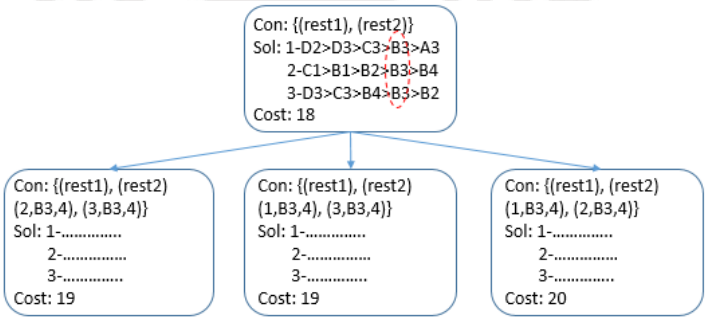


Figura 1.20: Árbol de restricciones para 3 agentes. [Elaboración propia].

Otra opción sería crear nodos para los dos primeros agentes que se encuentren en conflicto y restringir la posición del otro en el nodo. Ver Figura 1.21. Puesto que no se restringió para el tercer agente, este aparecería en conflicto en los nuevos nodos y se generarían nuevos nodos, terminando con los mismos nodos que los mostrados en Figura 1.20.

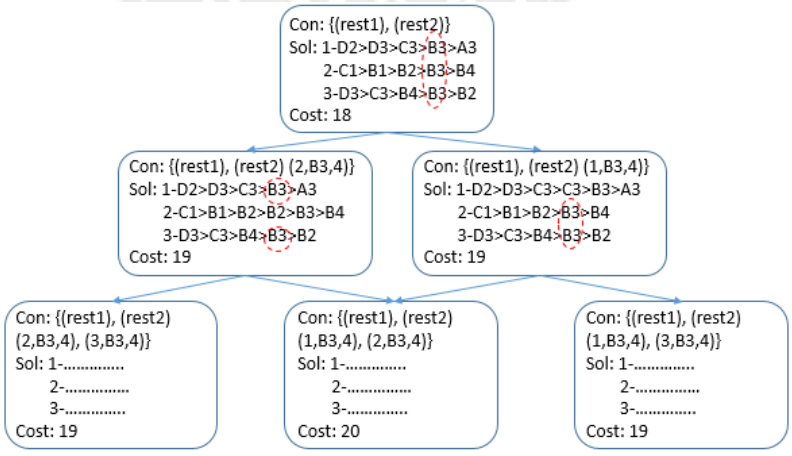


Figura 1.21: Árbol de restricciones alternativo para 3 agentes [Elaboración propia].

Dado que el problema empieza a crecer de manera exponencial cuando existen demasiados conflictos, en [12] se soluciona esto introduciendo el concepto de meta-agente y nombrando al algoritmo como MA-CBS. La idea de este algoritmo es la misma que la de detección de independencia [13], con la diferencia de que se aplica en el algoritmo de alto nivel y no el algoritmo de búsqueda. Es decir, si ciertos agentes entran en una cantidad de conflictos mayor a cierto número, ambos se fusionan en un meta agente y se comportan como si fuera uno solo. Esto ocurriría solo entre el paso 9 y el paso 10, el resto del algoritmo original de CBS permanece igual.

1.3 Estado del Arte

Existe dos tipos de planeamiento multi-robot. El primero, el centralizado, consiste en que un computador principal calcula los caminos o las trayectorias para todos los robots, asumiendo que los caminos a tomar por los mismos se encuentran acoplados o desacoplados; es decir que tienen o no dependencia entre ellos. Mientras que el segundo tipo de planeamiento multi-robot, el distribuido o descentralizado, consiste en que cada robot actúa como un computador que planea su propia trayectoria o camino, conociendo o no la trayectoria del resto de robots. El trabajo presente se enfocará en planeamiento de movimiento multi-robot centralizado. El estado del arte de dicho campo, se resume a continuación:

1.3.1 Planeamiento de Caminos Multi-Agente

Existen, al menos, tres tipos de solucionadores de planeamiento de caminos multi agente, según [10]. Los primeros son los solucionadores óptimos, los cuales se basan en algoritmos de búsqueda y siempre encuentran la mejor solución al problema de existir alguna (son completos y óptimos). Los segundos son los solucionadores sub-óptimos, los cuáles se basan en algoritmos de búsqueda, en reglas, o en una mezcla de ambas (solucionadores híbridos) y no siempre hallan la solución al problema; y de encontrar alguna, puede no ser la mejor solución posible. Finalmente, se tiene los solucionadores por reducción, los cuales se basan en reducir (transformar) el problema de planeamiento multi-agente a algún otro problema conocido, como SAT (problema de satisfacción booleana), y aplicar las soluciones ya existentes a dicho problema. Dado que este último es más bien un problema de ciencia computacional teórica y que los solucionadores sub-óptimos no garantizan la solución del problema, se ahondará solo en los solucionadores óptimos existentes hasta la actualidad.

a) Algoritmo A Estrella

La solución directa para aplicar el algoritmo A estrella en múltiples móviles, es tratar a todos los móviles como si fueran un único agente, así cada nodo representaría un conjunto de posiciones resultante de alguna combinación de movimiento de todos los móviles. Por ejemplo, para los móviles de Figura 1.22, el móvil verde tiene 2 siguientes posibles movimientos y el azul 5 siguientes posibles movimientos. Si se combina dichos movimientos, se puede obtener hasta 10 posibles combinaciones movimientos en el siguiente instante; por lo tanto, la cantidad de nodos sucesores sería también 10 (ver Figura 1.23).

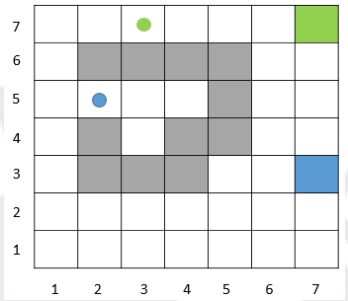


Figura 1.22: Problema de planeamiento de caminos multi agente [Elaboración propia].

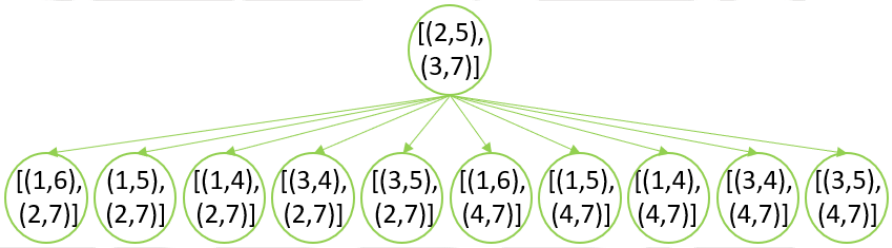


Figura 1.23: Primera iteración según solución estándar de algoritmo A [Elaboración propia].*

En el peor de los casos, cuando los n agentes pueden realizar todos sus M movimientos, se generaría M^n siguientes nodos. En el caso genérico de que cada agente pueda realizar sus 8 posibles movimientos, se generaría hasta 64 nodos sucesores solo para un instante. Notar que ejecutar este algoritmo con un alto número de vehículos, lo hace ineficiente, debido a la cantidad de tiempo que crear numerosos nodos y buscar la solución entre ellos. De ser este el caso, se puede utilizar variantes del algoritmo A*, como PEA* [14][15], o EPEA* [16] para reducir el número de nodos. Donde la idea en ambos algoritmos es que se agregue a la lista de abiertos solo los nodos hijos con un costo igual al del nodo padre, descartado al resto de hijos y se mantenga el nodo padre en la lista de abiertos, en caso se deba utilizar los hijos descartados previamente. Donde la diferencia entre ambos es que en EPEA* los hijos son generados una sola vez, cuando se hace uso de ellos; mientras que, en PEA* se los genera cada vez que se expande su nodo padre.

b) Algoritmo de Detección de Independencia

El algoritmo de detección de independencia (ID), planteado en [13], se basa en dividir el espacio de trabajo en grupos y buscar los caminos de los móviles en cada grupo. En caso de que se detecte un conflicto de caminos entre dos grupos diferentes, recalcula los caminos para alguno de los dos grupos y si se sigue presentando el conflicto une a los dos grupos en un nuevo grupo y realiza la búsqueda de caminos en este nuevo grupo. Técnicamente, se puede utilizar cualquier algoritmo de búsqueda con el algoritmo ID. El pseudocódigo del algoritmo se muestra en Figura 1.24.

Algorithm 2 Independence Detection

```
1:  assign each agent to a group
2:  plan a path for each group
3:  fill conflict avoidance table with every path
4:  repeat
5:      simulate execution of all paths until a conflict between two
        groups G1 and G2 occurs
6:      if these two groups have not conflicted before then
7:          fill illegal move table with the current paths for G2
8:          find another set of paths with the same cost for G1
9:          if failed to find such a set then
10:             fill illegal move table with the current paths for G1
11:             find another set of paths with the same cost for G2
12:         end if
13:     end if
14:     if failed to find an alternate set of paths for G1 and G2 then
15:         merge G1 and G2 into a single group
16:         cooperatively plan new group
17:     end if
18:     update conflict avoidance table with changes made to paths
19: until no conflicts occur
20: solution ← paths of all groups combined
21: return solution
```

Figura 1.24: Pseudocódigo del algoritmo de detección de Independencia [13].

c) Algoritmo de Descomposición de Operador

El algoritmo de descomposición de operador (OD), planteado en [13], pretende reducir el número exponencial de nodos sucesores que se generan en la solución estándar del algoritmo A*. La idea es que en vez de crear en un solo instante de tiempo M^n nodos sucesores de todas las posibles combinaciones de posiciones, crea M nodos de los posibles movimientos de un agente individual y se le asigna uno de esos movimientos al agente (sin actualizar la posición), luego repite lo mismo con cada agente hasta que los “n” agentes tengan asignado un movimiento. Cuando al último agente se le asigna el movimiento, se incrementa el siguiente instante de tiempo y se actualiza la posición de todos los agentes. Así en T instantes de tiempo, se tendría $T * M * n$ nodos resultantes en vez de $T * M^n$ nodos. Es decir, el algoritmo de A* pasaría a tener una complejidad lineal O(n) en vez de una exponencial O(M^n).

En el algoritmo, a cada nodo generado al empezar un nuevo instante de tiempo se le llama nodo estándar y a cada nodo generado en algún paso intermedio antes de llegar al nuevo instante de tiempo se le llama nodo intermedio.

El algoritmo OD es el mismo algoritmo de A*, pero con un par de diferencias:

- La primera es que el vector de estado de un solo nodo ahora contiene todas las posiciones y los movimientos a asignar a los agentes. Por ejemplo, si fueran 3 agentes, el vector de estado de un nodo sería de la forma X = [xy1, xy2, xy3, mov1, mov2, mov3]
- La segunda es que los nodos sucesores se generan por cada agente y no en conjunto.

Por ejemplo, si el problema fuera el de Figura 1.25. En el cual los móviles son los círculos con objetivos de cuadros del mismo color y solo se está permitido los movimientos arriba, abajo, izquierda y derecha y no se está permitido que dos móviles vayan en direcciones opuestas o que ocupen el mismo casillero.

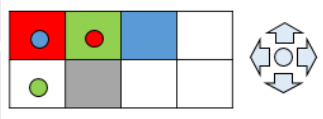


Figura 1.25: Problema para ejemplificación de algoritmo de descomposición de operados [Elaboración propia].

La transición del primer instante de tiempo al siguiente sería como se muestra en Figura 1.26.

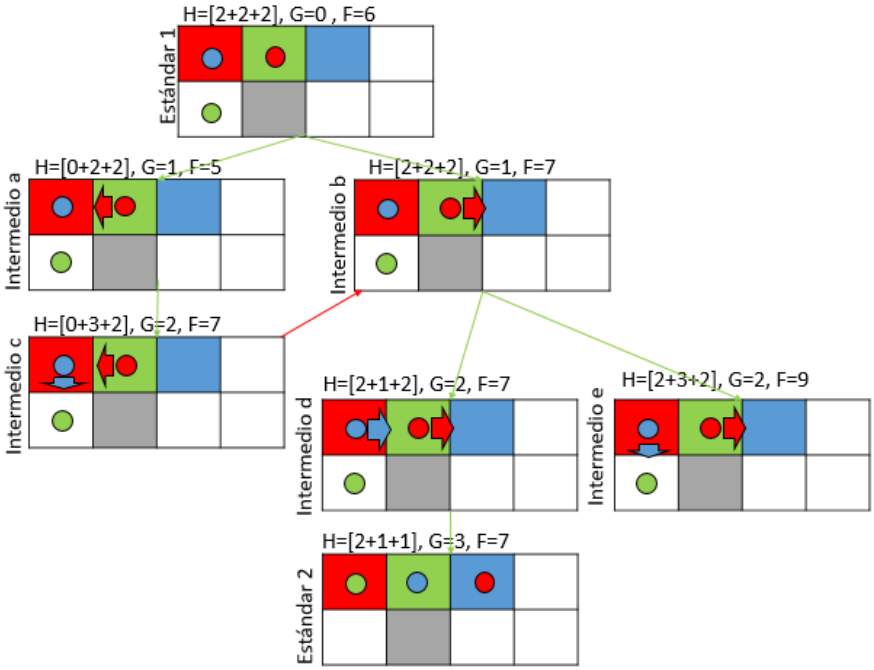


Figura 1.26: Solución del problema según A con descomposición de operador [Elaboración propia].*

d) Búsqueda en Árbol de Costo Incremental

El algoritmo ICTS, planteado en [17], es un algoritmo de búsqueda de dos niveles. En el nivel superior crea un árbol de costos, como el mostrado en Figura 1.27.

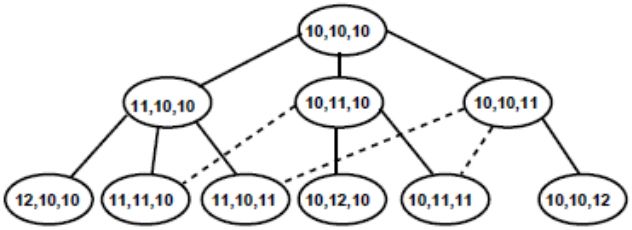


Figura 1.27: Ejemplo de árbol de costos para 3 agentes [17].

En este árbol, cada nodo contiene un vector de los costos de los caminos individuales que lleva a cada agente a su destino. En el primer nivel, son los costos de los caminos planeados sin considerar al resto de agentes, en el segundo nivel, se incrementa el costo de uno de los caminos en 1, en el tercer nivel, se incrementa en 1 el costo de uno de los caminos y así sucesivamente. Adicionalmente al vector de costos, cada nodo tiene asociado un diagrama de decisión multi-valor (MDD) de todos los agentes en conjunto. Este diagrama es una representación de todos los posibles caminos a tomar por los agentes restringido a un costo específico. Por ejemplo, si el problema fuera el de Figura 1.28 (i). El MDD del agente 1 ´que describe todos los posibles caminos a tomar por este con la restricción de llegar a su objetivo con un costo de 3 (MDD_1^3) sería el diagrama de la derecha de Figura 1.28 (ii), mientras que para llegar con un costo de dos (MDD_1^2) sería el de la izquierda de Figura 1.28 (ii). Así mismo los MDD para el agente 2 serían los de Figura 1.28 (iii). Para construir un MDD, se utiliza búsqueda en anchura (breadth-first-search), que lo que hace es elegir el nodo inicial (nodo a en el ejemplo), luego agregar a los nodos que están conectados al mismo (a, b, c) y luego agregar a los nodos conectados a los últimos nodos agregados y así sucesivamente.

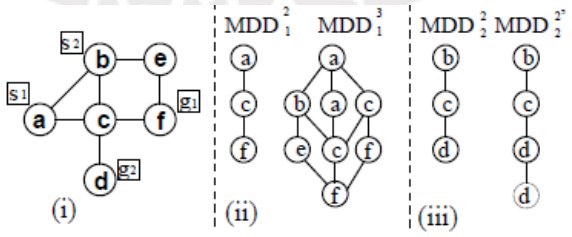


Figura 1.28:(i) Problema a resolver. (ii) MDDs para agente 1. (iii) MDDs para agente 2 [17].

Un MDD de cualquier nodo del árbol de costos es la fusión de varios MDDs. Por ejemplo, si el nodo del árbol de costos fuera [3, 2]; esto indicaría que el MDD asociado a ese nodo es la fusión de MDD_1^3 y $MDD_2^{2'}$. Esta fusión de MDDs se muestra en Figura 1.29.

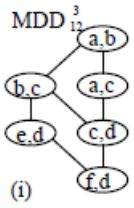


Figura 1.29: (i) Fusión de MDDs. [17].

Para lograr la fusión, se debe realizar una prueba de objetivo, la cual busca los caminos que no posean conflicto alguno; por ejemplo MDD_1^2 y MDD_2^2 no pueden fusionarse porque existe un conflicto en el nodo c. El algoritmo de bajo nivel es quien realiza la prueba de objetivo y si se logra encontrar al menos un camino válido que una el nodo inicial con el nodo objetivo de los agentes, retorna verdadero. Por otro lado, el algoritmo de alto nivel empieza creando los MDD individuales de cada agente y como la suma de costos individuales (SIC) de cada nodo de un mismo nivel es la misma, el algoritmo de alto nivel procede a buscar una solución válida en cada nodo de manera ordenada, ejecutando el algoritmo de bajo nivel, empezando por el primer nivel. Y tan pronto como el algoritmo de bajo nivel retorne un verdadero, cesa al algoritmo y retorna la solución. El pseudocódigo del algoritmo, se presenta en Figura 1.30.

Algorithm 1: The ICT-search algorithm

Input: (k, n) MAPF
Build the root of the ICT
foreach *ICT node in a breadth-first manner* **do**
  foreach *agent* a_i **do**
    Build the corresponding MDD_i
  end
  [**foreach** *pair of agents* a_i, a_j **do**
    perform pairwise search **//optional**
    if *pairwise search failed* **then**
      break *//Conflict found. Next ICT node*
    end
  end
  ] search the k-agent MDD *// low-level search*
    if *goal node was found* **then**
      return Solution
  end
end

Figura 1.30: Pseudocódigo de algoritmo ICTS [17].

Opcionalmente, se puede agregar una búsqueda en pares de los MDD previo a realizar la prueba de objetivo del conjunto de todos los k agentes. Esto es beneficioso, puesto que de encontrarse algún conflicto en un par de MDDs, ya no será necesario realizar la prueba de objetivo de los k agentes puesto que ese nodo sería inválido.

1.3.2 Coordinación en el Tiempo

Dado que las soluciones al problema de planeamiento de caminos multi-agente asumen que la velocidad de los móviles es constante y la misma para todos, se pueden generar colisiones en una implementación real. Por ello, es necesario introducir el tiempo en el planeamiento de caminos. En la coordinación, los caminos pueden ser variables en espacio y tiempo o pueden ser caminos fijos, modificables solamente en el tiempo.

a) Planeamiento en espacio de configuración en el tiempo

De trabajarse el sistema de planeamiento multi-robot con prioridades, los vehículos de mayor prioridad, con trayectoria previamente fijada, serían vistos como obstáculos móviles. Dichos obstáculos, pueden ser evitados en el espacio de configuración en tiempo [9], [18], el cual se explica a continuación:

Dado, un camino fijado τ en el espacio libre para un robot A:

$$\tau: [0\ ,1] \rightarrow C_{free} \qquad (4)$$

El espacio de configuración en tiempo (CT-Space) describe los obstáculos móviles que se han de presentar en el tiempo durante la transición de dicho camino. Por ejemplo, si en Figura 1.31 (a) el bloque amarillo es un robot; las líneas punteadas, el camino fijado para el mismo; y el bloque verde, un obstáculo móvil; el espacio de configuración en tiempo para dicho camino sería como Figura 1.31 (b), en la cual se muestra la presencia del obstáculo en el tiempo para dicho camino.

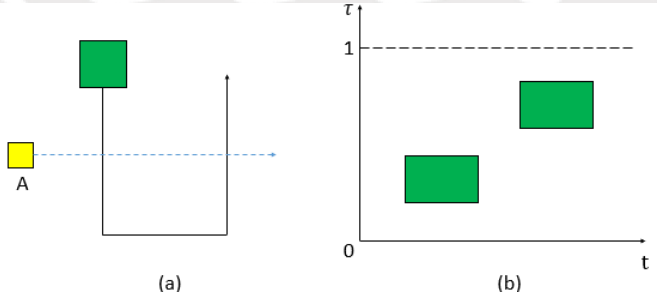


Figura 1.31: (a) Camino prefijado de un móvil y un obstáculo móvil. (b) Espacio de configuración en tiempo del camino prefijado del móvil [Elaboración propia].

Para que el robot, pueda evitar dicho obstáculo, deberá planear su movimiento en el espacio de configuración de tiempo, tal como se muestra en Figura 1.32.

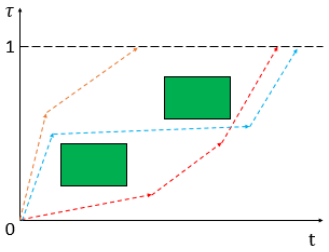


Figura 1.32: Planeamiento en espacio de configuración en tiempo [Elaboración propia].

Este planeamiento, puede ser realizado con algún planeador de caminos como el de Roadmap probabilístico, el de descomposición de celda o algún otro.

De no encontrarse fijado el camino, el espacio de configuración en tiempo, incluiría las dimensiones móviles del robot además del tiempo [18]. Por ejemplo, en Figura 1.33 se observa el espacio de configuración de tiempo de 3 dimensiones (2 de espacio y una de tiempo) con un obstáculo móvil y uno estático.

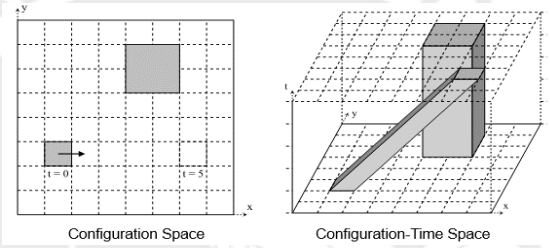


Figura 1.33: Espacio de configuración en tiempo para plano bidimensional [18].

Para planear a través de este espacio, bastaría con usar cualquier planeador de caminos, que lleve desde la coordenada (xi, yi, ti=0) a algún punto destino (xg, yg, tg = t).

b) Optimización Entera-Mixta

Cuando se tiene un camino fijo, una forma de parametrizar en el tiempo dicho camino, tal que se evite la colisión entre múltiples robots móviles, es utilizando optimización discreta, tal como se plantea en [19], [20] y [21]. En estos, se detecta en el espacio de configuración compuesta, los segmentos de colisión y se coordina el movimiento de los móviles usando programación entera mixta lineal (MILP) o no lineal (MINLP).

En [19], se plantea dos modelos para resolver el problema. El primero, llamado modelo instantáneo, asume que los móviles pueden cambiar de velocidad de manera instantánea. El segundo, llamado modelo de velocidad continua, incluye las ecuaciones cinemáticas del móvil. Ambos modelos, se encuentran restringidos a las velocidades máximas de los móviles.

Para explicar ambos modelos, tomaremos como ejemplo el problema mostrado en Figura 1.34. En esta se muestra dos móviles con sus caminos respectivos. El móvil A_1 va del punto "a1" al "a4" y A_2 va de "b1" a "b4". De estos se puede notar que las zonas de posible colisión para el robot son las definidas entre los segmentos [a1, a2] con [b3, b4] y la definida entre los segmentos [a3, a4] y [b1, b2].

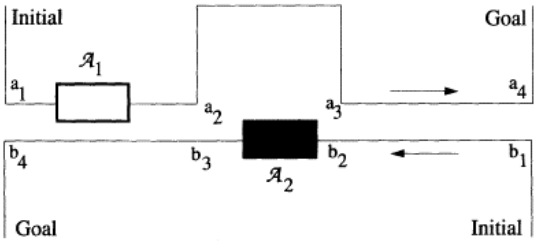


Figura 1.34: Ejemplo de movimiento de dos robots con caminos prefijados [19].

Definiéndose C_{max} como el máximo tiempo para que ambos robots lleguen a su destino; $t_{i,k}$ como el tiempo del robot "i" en el que llega al punto de inicio del segmento "k" y $\tau_{i,k}$ el tiempo que le toma a robot "i" transitar por el segmento "k". Entonces:

$$C_{max} \geq t_{i,last} + \tau_{i,last} \qquad (5)$$

Considerando un robot ""i" y un robot "j", con segmentos "k" y "h" respectivos de cada uno que definen una zona de colisión. Para evitar dicha colisión, los tiempos deberían ser $t_{jh} \geq t_{i(k+1)}$ o bien $t_{ik} \geq t_{j(h+1)}$, como no se puede cumplir ambas restricciones al mismo tiempo, se crea una variable binaria δ_{ijkh} por zona de colisión "kh". Así, definiendo un número M muy grande, se modifican las restricciones para que siempre se cumplan:

$$t_{jh} - t_{i(k+1)} + M(1 - \delta_{ijkh}) \geq 0$$
$$t_{ik} - t_{j(h+1)} + M\delta_{ijkh} \geq 0 \qquad (6)$$

Ya que se está optimizando tiempos, se define el tiempo máximo y mínimo de transición del robot móvil "i" por un segmento "k" de longitud S_{ik}. Dado que el robot puede realizarlo con una velocidad cercana a cero, el tiempo máximo sería infinito, $\Delta T_{ik}^{max} = \infty$, mientras que el mínimo estaría limitado por la velocidad máxima del móvil y sería $\Delta T_{ik}^{max} = \frac{S_{ik}}{v_{i,max}}$. Luego, si se deseara reducir el tiempo en el que todos los robots llegan a su destino, bastaría con definir un problema de optimización lineal entera mixta (MILP) en el que se minimice el tiempo total C_{max}, y se cumpla con la restricción de velocidad y evasión de colisiones de cada robot "i":

$$
\begin{aligned}
&Minimizar\ C_{max} \\
&sujeto\ a: \\
&C_{max} \geq t_{i,last} + \tau_{i,last} \qquad for\ i = 1, \dots, n \\
&t_{i,k} \geq 0 \\
&t_{i,(k+1)} = t_{ik} + \tau_{ik} \\
&\Delta T_{ik}^{max} \geq \tau_{ik} \geq \Delta T_{ik}^{min} \\
&t_{jh} - t_{i(k+1)} + M\left(1 - \delta_{ijkh}\right) \geq 0 \\
&t_{ik} - t_{j(h+1)} + M\delta_{ijkh} \geq 0 \\
&\delta_{ijkh} \in \{0,1\}
\end{aligned}
\qquad (7)
$$

Por otro lado, el modelo de velocidad continua, busca que las velocidades de los móviles sean consistentes a las aceleraciones de los mimos. De las ecuaciones de movimiento rectilíneo uniforme de un móvil, sabemos que:

$$v_{fin}^2 = v_{ini}^2 + 2 * a_{max} * t \qquad (8)$$

Por ende, si un segmento tiene una longitud S, y la velocidad del móvil al inicio del segmento es v_{start} y la del final del segmento es v_{end}, entonces dicho perfil de velocidad no es posible si:

$$\frac{\left|v_{end}^2 - v_{start}^2\right|}{2 * a_{max}} > S \qquad (9)$$

Además, el tiempo mínimo de transición por el segmento, tiene dos posibles casos, los cuales están sujetos a dos perfiles de velocidad (ver Figura 1.35).

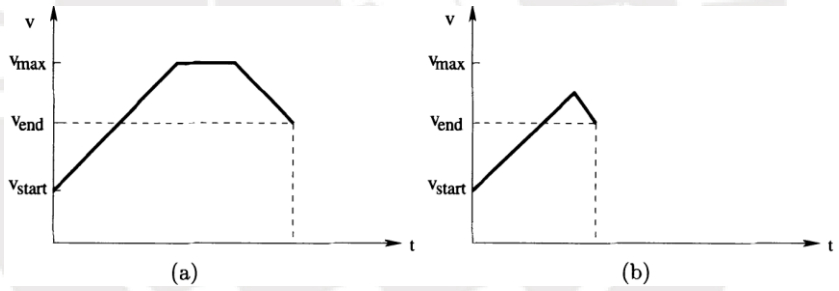


Figura 1.35: Perfiles de velocidad cuando: (a) Se alcanza Vmax. (b) No se alcanza vmax [19].

Para el caso (a) y (b) de Figura 1.35, el tiempo mínimo queda definido de la siguiente manera:

(a) If $S \geq \frac{1}{2} * \left(\frac{(v_{max}^2 - v_{start}^2)}{a_{max}} + \frac{(v_{max}^2 - v_{end}^2)}{a_{max}}\right)$,

$$\Delta T^{min} = \frac{S}{v_{max}} - \frac{\left((v_{max}^2 - v_{start}^2) + (v_{max}^2 - v_{end}^2)\right)}{2 * a_{max} * v_{max}} + \frac{v_{max} - v_{start}}{a_{max}} + \frac{v_{max} - v_{end}}{a_{max}}$$

(b) If $\frac{1}{2} * \left(\frac{(v_{max}^2 - v_{start}^2)}{a_{max}} + \frac{(v_{max}^2 - v_{end}^2)}{a_{max}}\right) > S \geq \frac{1}{2} * \frac{\left|v_{max}^2 - v_{end}^2\right|}{a_{max}}$,

$$\Delta T^{min} = \frac{(v_{middle} - v_{start})}{a_{max}} + \frac{(v_{middle} - v_{end})}{a_{max}}$$

where $v_{middle} = \frac{1}{2} * \left(2 * v_{start}^2 + 2 * v_{end}^2 + 4 * S * a_{max}\right)^{0.5}$ (10)

De manera similar, el tiempo máximo, también está limitado por dos perfiles. Ver Figura 1.36.

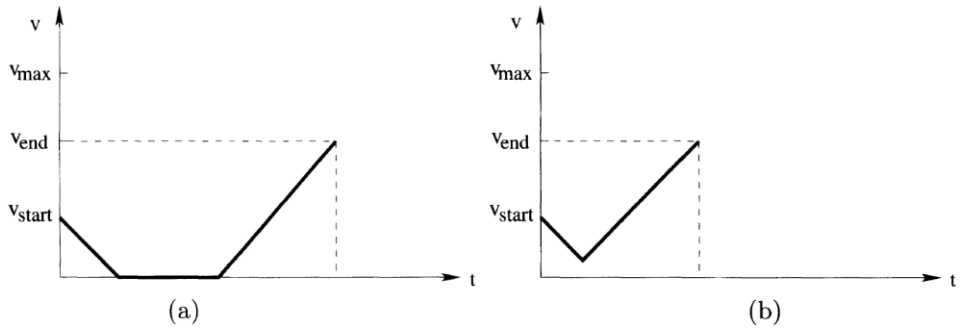


Figura 1.36: Perfiles de velocidad cuando: (a) Se alcanza v=0. (b) No se alcanza v=0 [19].

Luego, el tiempo máximo para el caso (a) y (b) de Figura 1.36 quedaría definido de la siguiente manera:

(a) If $S \geq \frac{1}{2} * \frac{(v_{start}^2 + v_{end}^2)}{a_{max}}$, $\Delta T^{max} = \infty$

(b) If $\frac{1}{2} * \frac{(v_{start}^2 + v_{end}^2)}{a_{max}} > S \geq \frac{1}{2} * \frac{|v_{end}^2 - v_{start}^2|}{a_{max}}$,

$$\Delta T^{max} = \frac{(v_{start} - v_{middle})}{a_{max}} + \frac{(v_{end} - v_{middle})}{a_{max}}$$

where $v_{middle} = \frac{1}{2} * \left(2 * v_{start}^2 + 2 * v_{end}^2 + 4 * S * a_{max}\right)^{0.5}$ *(11)*

Y al igual que en el modelo anterior, para que se puedan cumplir ambas restricciones al mismo tiempo del tiempo máximo y del tiempo mínimo, se definen las variables binarias $y_{ik,1}$, $y_{ik,2}$, $z_{ik,1}$ y $z_{ik,2}$.

Así mismo, puesto que ahora existe una aceleración, se debe garantizar que el perfil de velocidad es posible:

$$S \geq \frac{v_{end}^2 - v_{start}^2}{2 * a_{max}} \geq -S \qquad (12)$$

Luego, integrando todas las restricciones, se obtiene un problema de programación no lineal entera mixta (MINLP). El cual es linealizado posteriormente para obtener un problema de programación lineal entera mixta (MILP).

c) Planeamiento Temporal

c.1) Redes Temporales Simples

Para casos en donde la cantidad de vehículos y caminos son masivos, es preferible usar planeamiento temporal sobre los caminos previamente hallados. Este tipo de planeamiento, permite incluir restricciones de tiempo en los puntos del mapa, como instantes de llegada o salida, lo cual es propicio para sistemas automatizados. Dichos caminos pueden ser hallados por cualquier método que implique posiciones discretas; es decir, métodos que usen grillas o grafos.

En [22], se hace uso de redes simples temporales, para evitar la colisión entre robots, en esta solución, se toma en consideración la velocidad de los móviles, las cuales pueden variar para un mismo móvil; así como una distancia δ de seguridad entre robots. Para resolver el problema, se toma como referencia Figura 1.37. En esta se muestran dos móviles "1" y "2" que deben llegar a su objetivo (figuras traslúcidas) en un mapa en forma de T.

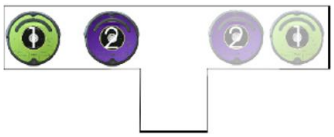
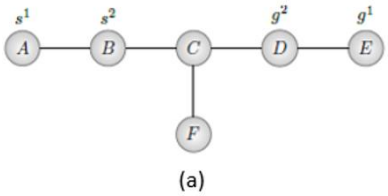


Figura 1.37: Problema de ejemplo para planeamiento temporal [22].

Para resolver este problema, se describe al mapa como el grafo mostrado en Figura 1.38 (a) y luego de aplicar algún algoritmo para solución de planeamiento de caminos multi-agente se llega a la solución mostrada en Figura 1.38 (b).



(a)

Agent	$t=1$	$t=2$	$t=3$	$t=4$
1	$A \to B$	$B \to C$	$C \to D$	$D \to E$
2	$B \to C$	$C \to F$	$F \to C$	$C \to D$

(b)

Figura 1.38: (a) Mapa del ejemplo representado en grafo. (b) Solución del problema, obtenida por algún método de planeamiento multi-agente, representado en una tabla [22].

Dicha solución es transformada en un grafo de planeamiento temporal como en Figura 1.39. Donde cada línea horizontal describe los caminos que debe tomar cada agente y cada línea trasversal, describe la dependencia de un mismo vértice del grafo en ambos caminos.

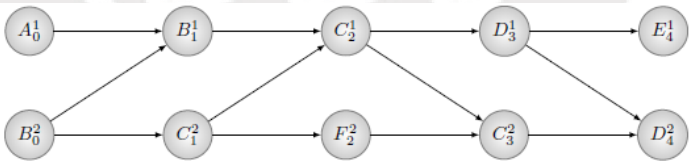


Figura 1.39: Grafo de planeamiento temporal del ejemplo [22].

Luego, se agrega dos vértices intermedios entre cada uno de los vértices originales, ver Figura 1.40. Estos vértices representan, posiciones que se encuentran a una distancia de seguridad δ del vértice más cercano.

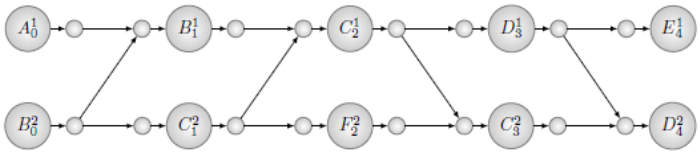


Figura 1.40: Grafo de planeamiento temporal aumentado del ejemplo [22].

Enseguida, se agrega un vértice inicial y final del grafo y se agrega información del tiempo máximo y mínimo que tomaría transitar cada arista, convirtiendo al grafo de planeamiento temporal en una red temporal simple. Como un robot, puede transitar una arista con una velocidad cercana a cero, se puede decir que el tiempo máximo es infinito. Por otro lado, el tiempo mínimo se obtiene dividiendo la distancia de la arista entre la velocidad máxima de cada robot. En Figura 1.41, se muestra dichos tiempos, considerando que la velocidad máxima de R1 es 0.25 m/s, la velocidad máxima de R2 es 0.625 m/s, la distancia de seguridad δ es 0.25 m y la distancia entre los vértices originales es 1 m.

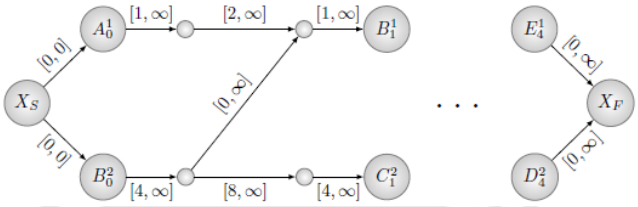


Figura 1.41: Red simple temporal del ejemplo [22].

Dado, que lo ideal es que se realicen estos movimientos, en el mínimo tiempo posible, se plantea un problema de optimización lineal en el que se busca minimizar el tiempo $t(v^j)$ que toma en llegar desde el vértice de inicio X_s al vértice v^j, el cual representa a cualquier vértice en el grafo. Sujeto a las restricciones mínimas y máximas de tiempo que toma en llegar de un vértice v cualquiera a un vértice v' adjunto. Esto de manera matemática, se expresa de la siguiente manera.

$$\begin{aligned} &Minimizar \sum_{j=1}^{K} t(v^j) \\ &tal\ que\ t(X_s) = 0 \\ &y, para\ todo\ e = (v, v') \in \mathcal{E}', \\ &t(v') - t(v) \geq LB(e) \\ &t(v') - t(v) \geq UB(e) \end{aligned} \qquad (13)$$

Esto mismo, se puede representar como un grafo de distancia, como en Figura 1.42 y resolver con el algoritmo de Floyd-Warshall's.

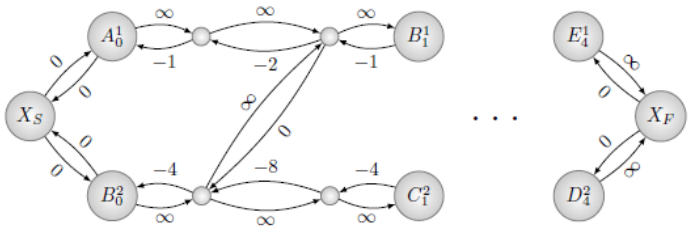


Figura 1.42: Grafo de distancia resultante de la red simple temporal del ejemplo [22].

Sabiendo los tiempos en los que se debe llegar a cada posición, se puede variar la velocidad del móvil, tal que se cumpla con los tiempos planeados.

c.2) Redes de Flujo en el Tiempo

Uno método que incluye, tanto planeamiento temporal como planeamiento de caminos es el método de redes de flujo en el tiempo [23], [24] y [25]. Este método ha sido probado con resultados exitosos en robots móviles [26] y [27]. El método de redes de flujos, parte de la suposición que se tiene un grafo que describe al espacio transitable por el robot (el espacio libre) y transformando este a un grafo de flujo [28] de vehículos en el tiempo, describe los posibles desplazamientos del robot. Asumiendo que el grafo original es el que se muestra en Figura 1.43 (a), donde la notación S_i^+ indica el punto de partida del robot i-ésimo y la notación S_i^-, indica el punto de llegada del mismo, se procedería como se detalla a continuación.

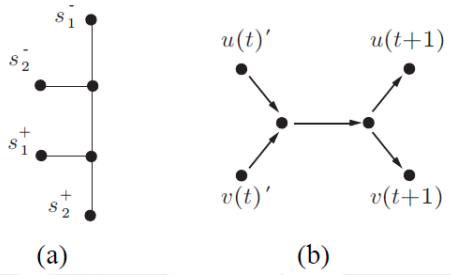


Figura 1.43: (a) Problema a resolver. (b) Estructura de grafo con forma de H [24].

Se debe replicar el grafo original múltiples veces y unir las transiciones de posición en el tiempo con una sección de grafo en forma de H, Figura 1.43 (b). El resultado de dicha operación sería como las líneas negras de Figura 1.44, estas servirían para ir de un vértice a otro. Para mantenerse en el mismo vértice, se agregan aristas entre instantes diferentes del mismo vértice (líneas verdes en la figura). Así mismo, para mantener el orden, se replica el último instante (en la figura denotado con apóstrofe) y se une el vértice a sí mismo con otra arista (línea azul). Finalmente, para reducir el número de ecuaciones, se agrega una arista de retorno de flujo (línea roja).

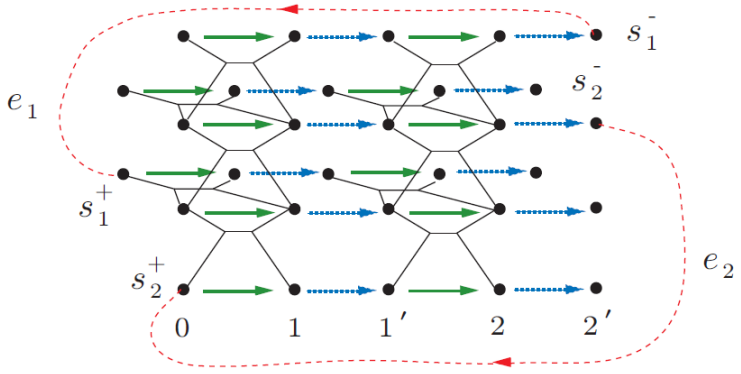


Figura 1.44: Grafo de Flujo de vehículos en el tiempo [24].

El flujo al que se hace alusión, es el flujo de vehículos. Dado que dos vehículos no pueden estar en una misma posición al mismo tiempo, se limita el flujo de vehículos (capacidad) de todas las aristas a 1 (denotado con “/1” en Figura 1.45). Puesto que por cada arista azul solo sale un vehículo, esto implicaría que al vértice de origen de dicha arista solo llega un vehículo. Así mismo el grafo en forma de H, al tener capacidad de 1 en todas sus aristas, evita que dos vehículos viajes en direcciones opuestas. Adicionalmente a la capacidad, se agrega un costo por trasporte de flujo en cada arista, donde solo la arista horizontal del grafo en forma de H y la arista azul tienen costo 1, mientras que el resto de aristas tienen costo 0.

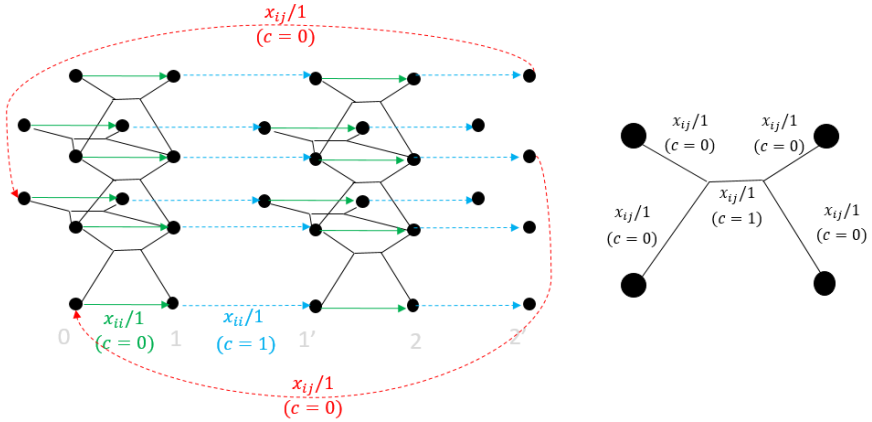


Figura 1.45: Capacidad y costo de las aristas del grafo de flujo [Elaboración propia].

El costo de cada arista, está a la distancia a recorrer del móvil. Mientras que el flujo X, indica que por ahí pasó o no un vehículo; es decir X solo puede tomar el valor de 1 o 0. En este grafo, se puede resolver dos problemas, el primero está asociado a encontrar los caminos que minimizan el tiempo de llegada de todos los vehículos a su destino y el segundo asociado a minimizar la distancia recorrida por todos los mismos.

La primera opción de problema a resolver es uno de minimización de “makespan”, que es el máximo tiempo entre los mínimos tiempos que le toma a cada robot llegar a su destino.

$$\text{makespan} = \max(TR1_{min}, TR2_{min}) \qquad (14)$$

De Figura 1.45 se puede notar que, si un vehículo llega a su destino en el instante t, en los siguientes instantes t+k pasará solo por las líneas azules y verdes. Así, mientras más rápido llegue un vehículo a su destino, mayor será el número de líneas azules y verdes que recorra luego de llegar a su destino. Por ello, el problema de minimización de makespan, puede plantearse como un problema de maximización de pasadas (Xii) por las líneas azules y verdes, donde todo X solo

puede tomar valores de 1 o 0 (pasó o no pasó por la arista) y por ende restringida a que la suma de flujos X que llegan a un vértice solo pude ser 0 o 1 y así mismo restringido a que la cantidad de vehículos que sale por cada vértice es igual a la cantidad de vehículos que entran al mismo. Esto resulta en un problema de optimización lineal de enteros (debido a los valores enteros de X), el cual puede ser descrito de la siguiente manera:

$$\max \sum_{1 \le i \le n}^{n} x_{ij}$$
$$\forall\, e_j \in G', \sum_{i=1}^{n} x_{ij} \le 1$$
$$\forall\, 1 \le i, j \le n, i \ne j, \quad x_{ij} = 0$$
$$\forall\, v \in G'\ y\ 1 \le i \le n, \quad \sum_{e_j \in \delta^+(v)} x_{ij} = \sum_{e_j \in \delta^-(v)} x_{ij} \qquad (15)$$

Donde δ^+ indica los flujos de llegada al vértice j y δ^- los flujos de salida del mismo.

La segunda opción de problema a resolver es el de minimización de la distancia recorrida por todos los robots, el cual se puede plantear cambiando la función objetivo por la suma de costos de transitar por cada arista multiplicado por la variable X que indica si el móvil paso o no por dicha a arista. Donde el costo vendría a ser la distancia a recorrer por el móvil durante su transición por dicha a arista.

$$\min \sum_{e_j \in G', j > n, 1 \le i \le n} c_2(e_j) * x_{ij} \qquad (16)$$

1.3.3 Generación de Trayectoria Multi-Robot

Existen al menos tres formas de diseñar un sistema de planeamiento de trayectorias multi robot. El primero es generando las trayectorias para todos los robots por algún método de optimización como un sistema acoplado, el segundo es generando las trayectorias a partir del planeamiento de caminos y el tercero es generando las trayectorias por prioridades en un sistema desacoplado. Dado que este último, no ofrece una solución óptima, nos enfocaremos en las otras dos opciones; ejemplo de este último se tiene en [29].

a) Optimización

Una forma acoplada de resolver el problema de planeamiento de trayectoria multi-robot es generando las trayectorias de todos los robots a partir de métodos de optimización. Un problema de optimización de este tipo, suele ser planteado como un problema de programación lineal o cuadrática en su función de costo; es decir, como un problema de control óptimo. Sin embargo, la inclusión de restricciones

para la evasión de obstáculos y evasión de colisiones termina transformado el problema y la solución del mismo en otro tipo de optimización, o bien porque el problema deja de ser convexo o porque se incluyen variables enteras. A continuación, se explica ambos casos.

a.1) Optimización Entera-Mixta

Una manera de integrar obstáculos con formas no uniformes en un problema de optimización lineal o cuadrático es describiéndolos por sus vértices y agregando restricciones que impidan que la posición de alguno de los móviles se encuentre entre las coordenadas de dichos vértices. Para la inclusión de dichas restricciones se suele utilizar variables binarias, las cuales son enteras. Este tipo de problemas de optimización que involucran tanto variables continuas como variables enteras se les conocen como programación entera mixta.

En [30] se describe al problema como un problema de optimización lineal sujeto a restricciones de igualdad definidas por la dinámica del móvil y a restricciones de desigualdad que limitan la entrada de control, al cual se le debe agregar la evasión de obstáculos y la evasión de colisiones entre robots. La función de costo del problema original es la siguiente:

$$\min_{w_{pi}, v_{pi}} \sum_{p=1}^{K} \left(\sum_{i=1}^{N} q_p' w_{pi} + \sum_{i=0}^{N-1} r_p' v_{pi} + p_p' w_{pN} \right) \quad (17)$$

Al problema original, se agregan obstáculos rectangulares, los cuales son descritos por su vértice superior derecho (xmax, ymax) y su vértice inferior izquierdo (xmin, ymin). Para evadir estos obstáculos, se agregan restricciones que impiden que un móvil “i” se encuentre dentro del obstáculo, las cuales son de la siguiente manera:

$$\begin{aligned} \forall i \in [1, \dots, N]: x_i &\leq x_{min} \\ or\ x_i &\geq x_{max} \\ or\ y_i &\geq y_{min} \\ or\ y_i &\geq y_{max} \end{aligned} \quad (18)$$

Sin embargo, estas incluyen el operador lógico OR, por lo cual no pueden ser integradas de esa en el problema lineal. Por ello, se crean variables de holgura t_{pcik} de tipo binarias, las cuales son multiplicadas por un número M muy grande y agregadas en las restricciones, de manera que las restricciones previamente planteadas se cumplan en todo instante de tiempo. Así mismo, se agrega una restricción adicional, que impide que las 4 variables binarias sean 1, pues esto implicaría que el móvil se encuentra dentro del obstáculo.

$$\forall c \in [1, \dots, L], i \in [1, \dots, N]:$$
$$x_{pi} \leq x_{c,min} + Mt_{pci1}$$
$$-x_{pi} \leq -x_{c,max} + Mt_{pci2}$$
$$y_{pi} \leq y_{c,min} + Mt_{pci3}$$
$$-y_{pi} \leq -y_{c,max} + Mt_{pci4}$$
$$\sum_{k=1}^{4} t_{pcik} \leq 3$$
$$t_{pcik} = 0 \ or \ 1 \qquad (19)$$

Para agregar la evasión de colisiones, se procede de manera similar. Si la distancia mínima entre un robot "p" y uno "q" para evitar la colisión fuera d_x en el eje X y d_y en el eje Y, las restricciones serían de la siguiente manera:

$$\forall\, i \in [1, \dots, N]: \forall p, q | q > p: |x_{pi} - x_{qi}| > d_x$$
$$or \ |y_{pi} - y_{qi}| > d_y \qquad (20)$$

Nuevamente, para poder integrar estas restricciones, se crean variables de holgura binarias. Y se agrega una restricción que impide que todas tengan como valor 1 al mismo tiempo, pues esto implica que estarían en colisión.

$$\forall\, i \in [1, \dots, N]: \ q \in [p+1, \dots, K],$$
$$x_{pi} - x_{qi} \leq d_x - Mb_{pqi1}$$
$$x_{qi} - x_{pi} \leq d_x - Mb_{pqi2}$$
$$y_{pi} - y_{qi} \leq d_y - Mb_{pqi3}$$
$$y_{qi} - y_{pi} \leq d_y - Mb_{pqi4}$$
$$\sum_{k=1}^{4} b_{pqik} \leq 3$$
$$b_{pqik} = 0 \ or \ 1 \qquad (21)$$

Finalmente, se indica dos maneras de resolver el problema. la primera es usando control con horizonte deslizante. Y la segunda es modificando la función de costo para fijar el tiempo de llegada, La simulación para un solo vehículo, por ambos métodos, se muestran en Figura 1.46.

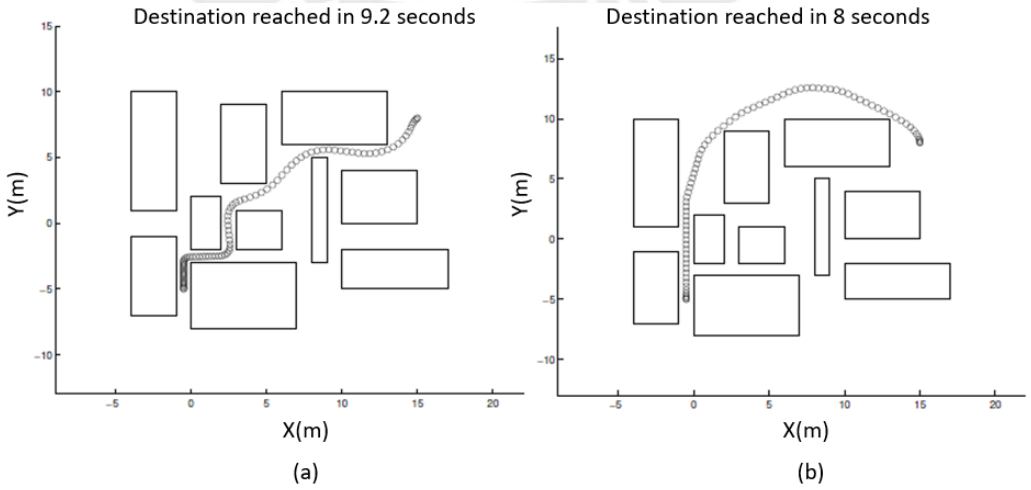


Figura 1.46: (a) Solución por horizonte deslizante. (b) Solución por tiempo de llegada fijo [30].

Este tipo de problemas, permiten agregar otras restricciones, como por ejemplo, puntos de pasada, orden de visita de los puntos, tiempo entre puntos de pasada, restricciones de cuáles moviles pueden visitar ciertos puntos de pasada y cuáles no. Estas restricciones adicionales son las que se agregan en [31].

En [31], la función de costo a optimizar par a "Nv" vehículos "p" con fuerzas de entrada f_{xtp} y f_{ytp} junto con las restricciones de evasión de obstáculos "j" con vértices (Z_{j1}, Z_{j2}) y (Z_{j3}, Z_{j4}) son como sigue:

$$J_1 = \min_{s,f,b,c} \bar{t} + \epsilon_1 \sum_{p=1}^{N_V} \left[t_p + \epsilon_2 \sum_{t=1}^{N_r - 1} \left(\left| f_{x_{tp}} \right| + \left| f_{y_{tp}} \right| \right) \right]$$

$$\forall t \in [1, \dots, N_T], \forall p \in [1, \dots, N_V], \forall m \in [1, \dots, N_C]$$

$$\dot{x}_{tp} \sin\left(\frac{2\pi m}{N_C}\right) + \dot{y}_{tp} \cos\left(\frac{2\pi m}{N_C}\right) \leq v_{max_p}$$

$$\forall t \in [0, \dots, N_T - 1], \forall p \in [1, \dots, N_V], \forall m \in [1, \dots, N_C]$$

$$f_{x_{tp}} \sin\left(\frac{2\pi m}{N_C}\right) + f_{y_{tp}} \cos\left(\frac{2\pi m}{N_C}\right) \leq f_p$$

$$\forall p \in [1, \dots, N_V], \forall t \in [0, \dots, N_T - 1]$$

$$S_{(t+1)p} = A s_{tp} + B f_{tp}$$

$$\forall t \in [1, \dots, N_T], \forall p \in [1, \dots, N_V], \forall j \in [1, \dots, N_Z]$$

$$\begin{aligned} x_{tp} - Z_{j3} &\geq -R c_{jpt1} \\ and\ Z_{j1} - x_{tp} &\geq -R c_{jpt2} \\ and\ y_{tp} - Z_{j4} &\geq -R c_{jpt3} \\ and\ Z_{j2} - y_{tp} &\geq -R c_{jpt4} \\ and \sum_{z=1}^{4} c_{jptz} &\leq 3 \end{aligned} \qquad (22)$$

Mientras que la restricciones de llegada a los "i" puntos de paso con coordenadas (W_{i1}, W_{i2}) son como sigue:

$$\forall p \in [1, \dots, N_V], \forall t \in [1, \dots, N_T], \forall i \in [1, \dots, N_W]$$

$$\begin{aligned} x_{tp} - W_{i1} &\geq \ R(1 - b_{ipt}) \\ and\ x_{tp} - W_{i1} &\geq -R(1 - b_{ipt}) \\ and\ y_{tp} - W_{i2} &\geq \ R(1 - b_{ipt}) \\ and\ y_{tp} - W_{i2} &\geq -R(1 - b_{ipt}) \\ \forall i \in [1, \dots, N_W] \sum_{t=1}^{N_T} \sum_{p=1}^{N_V} K_{pi} b_{ipt} &= 1 \end{aligned} \qquad (23)$$

Donde K_{pi} es una variable pre definida como 0 o 1 que indica si el robot "p" puede visitar o no el punto de paso "i".

Así mismo se agrega las restricciones de tiempo entre puntos de paso.

$$\forall k \in [1, \dots, N_C] \sum_{i=1}^{N_W} \Delta_{ki} \sum_{t=1}^{N_T} \sum_{p=1}^{N_V} t b_{ipt} \geq t_{D_k}$$

$$\forall p \in [1, \dots, N_V], \forall i \in [1, \dots, N_W]\ t_p \geq \sum_{t=1}^{N_r} t\ b_{ipt} \qquad (24)$$

$$\forall p \in [1, \dots, N_V]\ \bar{t} \geq t_p$$

Donde Δ es una variable predefinida como +1 o -1 que indica que se debe pasar primero por un punto de paso i.

La solución del problema en mención para dos vehículos móviles, donde ambos vehículos deben cubrir todos los puntos de paso, pero el vehículo 1 no puede pasar por el punto B, se muestra en Figura 1.47 (a). El problema que el punto D debe ser visitado antes que en punto A, se muestra en Figura 1.47 (b)y este mismo problema pero con un obstáculo añadido, se muestra en Figura 1.47 (c).

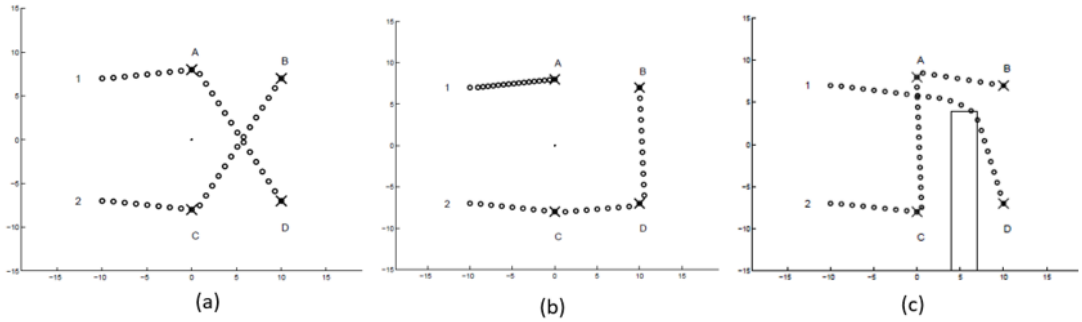


Figura 1.47:(a) Móvil 1 restringido a no pasar por B. (b) Restricción de pasar por D antes de A. (c) Restricción de pasar por D antes de A con un obstáculo [31].

Otro ejemplo más de uso de programación mixta para planeamiento multi-robot, se puede encontrar en [32], donde se utiliza programación mixta cuadrática (MIQP) para insertar evasión de obstáculos al problema de optimización cuadrática.

a.2) Optimización Continua No Convexa

Una manera alternativa de optimizar con funciones objetivo o restricciones no convexas es utilizando programación convexa secuencial (SCP) [33]. En este tipo de optimización, se modela a la función o restricción no convexa como una función convexa para una región de operación, la cual varía por cada iteración. Una forma usual de modelar dicha función o restricción es usando series de Taylor, para convertirla en convexa en una región definida. Dicha región es llamada región de confianza y se actualiza en cada iteración de manera heurística; por ende, este tipo de optimización no garantiza llegar a un mínimo global.

En [34] se plantea el problema inicialmente como un problema convexo con una función objetivo cuadrática (donde se desea disminuir en pares la aceleración de múltiples drones “i” y “j” en todos los instantes de tiempo “k”), sujeto a las restricciones cinemáticas de los mismos. Donde p_i es la posición, v_i la velocidad y a_i la aceleración.

$$\begin{gathered}
\underset{a_i[k]}{\operatorname{argmin}} \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} w_{ij} a^T[k] a_j[k], \\
p_i[k+1] = p_i[k] + h v_i[k] + \frac{h^2}{2} a_i[k] \\
v_i[k+1] = v_i[k] + h a_i[k] \\
j_i[k] = \frac{a_i[k+1] - a_i[k]}{h} \quad \forall i,k \\
p_i[K] = p_{fi}, \;\; v_i[K] = v_{fi}, \;\; a_i[K] = a_{fi}, \quad \forall i \\
p_i[K] \in [p_{min}, p_{max}], \;\; a_i[K] \in [a_{min}, a_{max}], \quad \forall i,k
\end{gathered} \tag{25}$$

Luego, se desea insertar la evasión de obstáculos rectangulares definidos por sus vértices. Y a diferencia de la sección anterior, se utiliza como restricción la lejanía a cada uno de los vértices, expresado de la siguiente manera:

$$\exp\left(c(p_i^x[k] - x_{cor})\right) + \exp\left(c\left(p_i^y[k] - y_{cor}\right)\right) \geq 2, \quad \forall i,k \tag{26}$$

Esta restricción es convexa. Por otro lado, para integrar la evasión de colisiones se utiliza la siguiente restricción no convexa:

$$\left\| p_i[k] - p_j[k] \right\|_2 \geq R \qquad \forall i,j \qquad i \neq j, \qquad \forall k \tag{27}$$

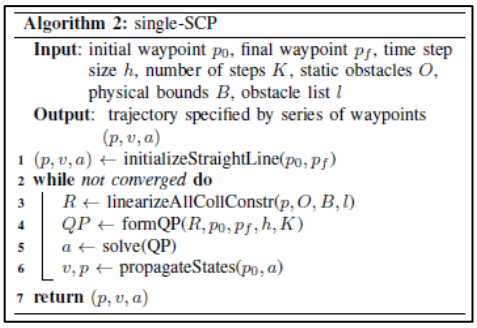
Para resolverse el problema por SCP, se linealiza la restricción por series de Taylor alrededor de una posición p^q de cada móvil:

$$\left\| p_i^q[k] - p_j^q[k] \right\| + \frac{\left(p_i^q[k] - p_j^q[k]\right)^T}{\left\| p_i^q[k] - p_j^q[k] \right\|} \left(p_i[k] - p_j[k]\right) \geq R \tag{28}$$

Haciendo este cambio, el problema vuelve a ser un problema de programación cuadrática, pero solo para un instante de tiempo.

La solución planteada es la del problema acoplado, ya que la linealización de la restricción incluye a todos los móviles. De aquí bastaría con resolver el problema por SCP, linealizando la restricción en cada instante de tiempo, resolviendo el problema de programación cuadrática y reutilizando el resultado para volver a linealizar alrededor de otro punto (nueva región de confianza).

A partir de este punto se propone resolver el problema de manera desacoplada, el cual consiste en resolver la trayectoria de cada móvil por separado. Es decir, cada móvil se comporta como un obstáculo estático para el siguiente. Para un solo móvil, el algoritmo de programación convexa secuencial se muestra en Figura 1.48.



Algorithm 2: single-SCP

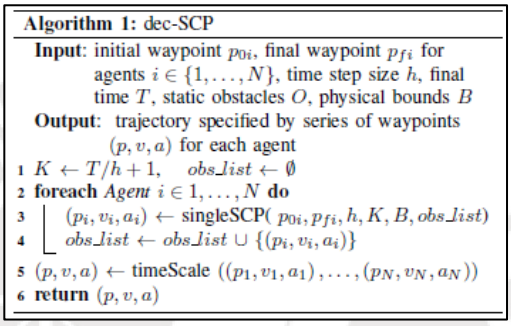
Input: initial waypoint p_0, final waypoint p_f, time step size h, number of steps K, static obstacles O, physical bounds B, obstacle list l

Output: trajectory specified by series of waypoints (p, v, a)

$(p, v, a) \leftarrow$ initializeStraightLine(p_0, p_f)
while *not converged* **do**
  $R \leftarrow$ linearizeAllCollConstr(p, O, B, l)
  $QP \leftarrow$ formQP(R, p_0, p_f, h, K)
  $a \leftarrow$ solve(QP)
  $v, p \leftarrow$ propagateStates(p_0, a)
return (p, v, a)

Figura 1.48: Algoritmo SCP para un solo móvil [34].

Dado que se debe resolver para cada móvil, el algoritmo solución del problema es el de Figura 1.49.



Algorithm 1: dec-SCP

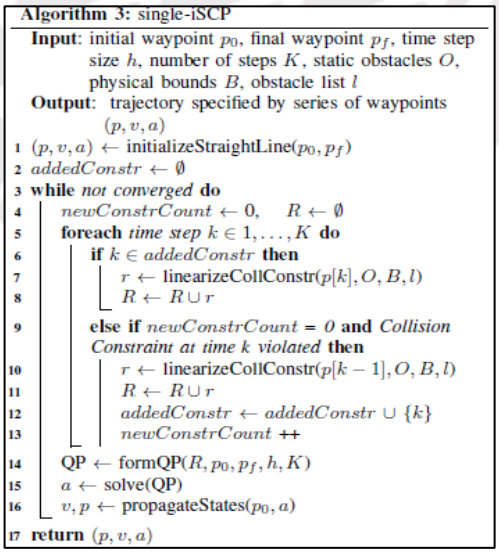
Input: initial waypoint p_{0i}, final waypoint p_{fi} for agents $i \in \{1, \ldots, N\}$, time step size h, final time T, static obstacles O, physical bounds B

Output: trajectory specified by series of waypoints (p, v, a) for each agent

$K \leftarrow T/h + 1, \quad obs\_list \leftarrow \emptyset$
foreach *Agent* $i \in 1, \ldots, N$ **do**
  $(p_i, v_i, a_i) \leftarrow$ singleSCP$(p_{0i}, p_{fi}, h, K, B, obs_list)$
  $obs\_list \leftarrow obs\_list \cup \{(p_i, v_i, a_i)\}$
$(p, v, a) \leftarrow$ timeScale $((p_1, v_1, a_1), \ldots, (p_N, v_N, a_N))$
return (p, v, a)

Figura 1.49: Algoritmo solución desacoplado basado en SCP [34]..

Finalmente, se plantea cambiar el algoritmo SCP por uno incremental. El cual es llamado programación convexa secuencial incremental (iSCP). Este consiste en agregar las restricciones de un solo móvil de manera secuencial. El algoritmo es el de Figura 1.50.



Algorithm 3: single-iSCP

Input: initial waypoint p_0, final waypoint p_f, time step size h, number of steps K, static obstacles O, physical bounds B, obstacle list l

Output: trajectory specified by series of waypoints (p, v, a)

$(p, v, a) \leftarrow$ initializeStraightLine(p_0, p_f)
$addedConstr \leftarrow \emptyset$
while *not converged* **do**
  $newConstrCount \leftarrow 0, \quad R \leftarrow \emptyset$
  foreach *time step* $k \in 1, \ldots, K$ **do**
    if $k \in addedConstr$ **then**
      $r \leftarrow$ linearizeCollConstr$(p[k], O, B, l)$
      $R \leftarrow R \cup r$
    else if $newConstrCount = 0$ **and** *Collision Constraint at time k violated* **then**
      $r \leftarrow$ linearizeCollConstr$(p[k-1], O, B, l)$
      $R \leftarrow R \cup r$
      $addedConstr \leftarrow addedConstr \cup \{k\}$
      $newConstrCount$ ++
  QP $\leftarrow$ formQP(R, p_0, p_f, h, K)
  $a \leftarrow$ solve(QP)
  $v, p \leftarrow$ propagateStates(p_0, a)
return (p, v, a)

Figura 1.50: Algoritmo iSCP [34].

Otro ejemplo de uso de programación convexa para solución de planeamiento de trayectoria multi-robot se tiene en [35].

b) Suavizado de Caminos y Parametrización en Tiempo

De haberse utilizado algún planeador de movimiento en grafos, bastaría con suavizar los caminos resultantes de los móviles y parametrizar en el tiempo dichos caminos. Existen muchos métodos de suavizado [36], entre ellos los basado en interpolación polinomial, haciendo uso de curvas de Bézier, splines cúbicos, B-splines o curvas de nurbs; los basados en curvas especiales, como curvas de Dubins, hipocicloide o clotoide. Además de los basados en optimización. Por otro lado, la parametrización en el tiempo, se debe realizar según las limitaciones del móvil.

1.4 Propuesta de Solución

Los planeadores multi agente revisados, pueden ser clasificados de 3 formas según su enfoque solución. Los cuales son acoplado, si resuelve todos los caminos en conjunto; desacoplado, si resuelve cada camino de forma individual sin considerar al resto de agentes; y dinámicamente acoplado, si empieza resolviendo los caminos de forma individual (desacoplada) y luego resuelve de manera conjunta (acoplada) los caminos en colisión.

En la Tabla 1, se muestra de forma resumida, la lógica de los algoritmos utilizados en los planeadores multi agente revisados, basándose en [10], [13] y [17]; así como su clasificación.

Tabla 1: Enfoque solución de planeadores multi agente [Elaboración propia].

MAPF	Tipo	Resumen de Pasos Solución
A*	Acoplado	1. Planea el primer paso de todos los agentes en conjunto combinando sus movimientos. 2. Busca la combinación de menor costo. 3. Planea el siguiente paso de todos los agentes. 4. De llegar al objetivo, reconstruye solución.
ODA*	Desacoplado (por paso)	1. Para un agente se obtiene los siguientes posibles pasos sin considerar al resto y se agregan a una lista de pasos. 2. Busca el paso de menor costo en la lista de pasos. 3. Para otro agente se obtiene los siguientes posibles pasos sin considerar al resto de agentes y se agregan a la lista. 4. Busca el paso de menor costo en la lista de pasos. 5. Compara pasos y descarta combinación si hay colisión. 6. De llegar al objetivo reconstruye camino.

CBS	Desacoplado (por camino)	1. Para cada agente en un nodo CBS planea su camino sin considerar al resto de agentes. 2. Compara los caminos y encuentra los agentes en conflicto. 3. Crea los siguientes nodos CBS, imponiendo restricciones a cada agente en conflicto; y agregarlos a lista de nodos CBS. 4. Busca en la lista el nodo CBS de menor costo. 5. De no haber conflictos en nodo elegido, retorna solución. 6. De haber conflictos en el nodo elegido, para cada agente en el nodo planea su camino con las restricciones impuestas sin considerar al resto de agentes.
IDA*	Dinámicamente Acoplado	1. Para cada agente planea su camino sin considerar al resto. 2. Si hay colisión, agrupa a los agentes en colisión. 3. Planea el camino conjunto de cada grupo en colisión. 4. De llegar al objetivo reconstruye camino conjunto del grupo. 5. Planea el camino individual de cada agente en no colisión. 6. De llegar al objetivo reconstruye camino del agente.
ICTS	Dinámicamente Acoplado	1. Define, en un nodo ICTS, el número de pasos a dar por cada agente para llegar al objetivo. 2. Para cada agente en el nodo ICTS planea su camino sin considerar al resto. 3. Para cada agente construye un grafo MDD que describe todos los caminos solución para el numero de pasos definido, usando los caminos hallados. 4. Fusiona los grafos MDD de todos los agentes en el nodo ICTS que compartan mismos pasos. 5. Busca en el grafo fusionado el camino conjunto con un número de pasos igual al definido. 6. De existir camino conjunto, reconstruye caminos y retorna solución 7. De no existir camino, crea los siguientes nodos ICTS, incrementando el número de pasos por agente y agregarlos la lista de nodos ICTS. 8. Luego obtén el siguiente nodo ICTS de la lista. 9. Para cada combinación de número de pasos repite los pasos del 2 al 7.

De esta se puede observar que, el algoritmo que requiere mayor número de pasos a implementar y que crea más estructuras (grafos ICTS, MDD y MDD fusionado) es el de ICTS, mientras que el algoritmo más sencillo de implementar es el A*. En orden de simplicidad de implementación se tiene al A*, seguido de ODA*, luego IDA*, luego CBS y finalmente al ICTS.

Por otro lado, la Tabla 2, muestra el número de nodos generados en el mejor y peor caso de cada planeador multi agente, asumiéndose que cada uno de los n agentes puede realizar M movimientos y llega en T pasos a su destino, basándose en [10], [13] y [17]

Tabla 2: Número de nodos generados por planeador multi agente [Elaboración propia].

MAPF	Tipo	Número de Nodos en Mejor Caso	Número de Nodos en Peor Caso
A*	Acoplado	$N = T * M^n$	$N = \sum_{k=0}^{T} (M^n)^k$
ODA*	Desacoplado	$N = M * T * n + 1$	$N = \sum_{k=0}^{T*n} M^k = \sum_{k=0}^{T} (M^n)^k$
CBS	Desacoplado	$N = N_L + 1 = (M * T + 1) * n + 1$	$N = N_L * N_H + N_H = n * \sum_{k=0}^{T} M^k * \sum_{h=0}^{DH} 2^h + \sum_{h=0}^{DH} 2^h$
IDA*	Dinámicamente Acoplado	$N = N_d = (M * T + 1) * n$	$N = N_d + N_c = nd * \sum_{k=0}^{T} (M)^k + \sum_{g=0}^{G} \sum_{k=0}^{T} (M^{nc_g})^k$
ICTS	Dinámicamente Acoplado	$N = N_L + 1 = n * \sum_{k=0}^{T} M^k + 1$	$N = N_L * N_H + N_H = (N_{MDD} + N_F) * N_H + N_H = \left(n * \sum_{k=0}^{T} M^k + f(M^{nc_g})\right) * \sum_{h=0}^{DH} n^h + \sum_{h=0}^{DH} n^h$

Donde:

- G: Número de grupos acoplados.
- nc_g: Número de agentes por grupo g acoplado.
- nd: Número de agentes desacoplados.
- n: Número total de agentes.
- M: Movimientos por agente.
- T: Número de pasos solución.
- N: Número total de nodos.
- N_c: Número de nodos creados por grupos acoplados.
- N_d: Número de nodos creados por agentes desacoplados.
- N_H: Número de nodos generados en algoritmo de alto nivel.
- N_L: Número de nodos generados en algoritmo de bajo nivel.
- DH: Profundidad de árbol generado en algoritmo de alto nivel.
- N_{MDD}: Número de nodos generados para construir todos los grafos MDD.
- N_F: Número de nodos generados de todos los grafos fusionados.
- $f()$: Función de número de nodos acoplados.

De esta se observa que, en el mejor de los casos (cuando no hay colisiones) los algoritmos desacoplados y dinámicamente acoplados generan un número de nodos lineal con respecto al número total de agentes. Así mismo, se observa que, en el peor de los casos, los algoritmos dinámicamente acoplados generan un número de nodos exponencial con respecto al número de agentes en colisión, mientras que los algoritmos acoplados generan un número de nodos exponencial con respecto al número total de agentes.

Dado que el tiempo de ejecución de los algoritmos depende del número de nodos generados, se busca utilizar en esta tesis a aquellos algoritmos que generen una menor cantidad de nodos en el mejor de los casos (por ejemplo, cuando se planea un solo móvil y no hay colisiones) y en el peor de los casos (por ejemplo, cuando el móvil debe esperar en una región el mapa para evitar la colisión). Por ello, se ha de descartar al algoritmo A* y ODA*, ya que producen una cantidad exponencial de nodos con respecto al número de agentes en el peor caso; además de que el A* produce un número elevado de nodos en el mejor caso. Así mismo, se descarta los algoritmos ICTS e IDA*, ya que generan un número exponencial de nodos con respecto al número de agentes acoplados en el peor caso, además de que el ICTS produce un número elevado de nodos en el mejor caso. Quedando así el algoritmo CBS, como el algoritmo elegido para la implementación de esta tesis, pues este produce un número lineal de nodos con respecto al número de agentes en el peor y mejor caso.

El algoritmo CBS está diseñado para ser utilizado en espacios discretos; mientras que la tesis requiere de un planeador que funcione en espacios continuos; por ello, se planea adaptar el algoritmo de búsqueda basada en conflictos (CBS), para que funcione en espacios continuos y así se pueda realizar el planeamiento de múltiples móviles no holonómicos. Para ello, será necesario el uso de un planeador local para robots móviles no holonómicos, el cual sirva como algoritmo de bajo nivel del algoritmo CBS. Este planeador local deberá ser adaptado a su vez para incluir el planeamiento en el tiempo, el cual debe ser capaz de agregar acciones de espera en puntos del mapa, y así poder ser utilizado con el algoritmo CBS.

Existen al menos tres tipos de planeadores no holonómicos. Los primeros, se basan en construir un árbol de configuraciones que describen la orientación y posición del móvil. Entre estos encontramos al algoritmo A Estrella Híbrido (Hybrid A*) [37][38], el cual construye dicho árbol usando las ecuaciones cinemáticas del móvil con un

número de entradas de control discretas. Los segundos, se basan en planear sobre una estructura tipo lattice [39][40], esta es una estructura que se crea sobre los espacios libres del mapa y describe los posibles caminos que puede transitar el móvil y que respetan la cinemática del mismo. Mientras que, los terceros, son los basados en optimización, en los cuales se optimiza una función de costo sujeto a restricciones cinemáticas o dinámicas del móvil, ejemplos de estos se pueden encontrar en [41] y [42].

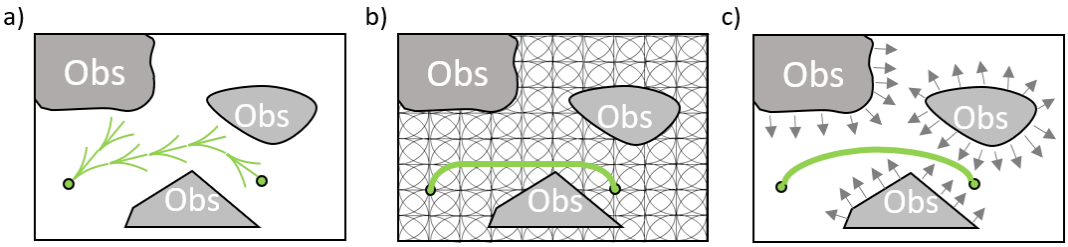


Figura 1.51: Planeamiento no holonómico con: a) Árbol de configuración, b) Lattice, c) Optimización [Elaboración propia].

Entre estos tres tipos de planeadores, los basados en optimización suelen ser los más lentos debido a que deben cumplir con restricciones de igualdad que respeten la cinemática o dinámica del móvil. Mientras que los planeadores que utilizan una estructura tipo lattice limitan el rango de movimiento del móvil, ya que estos solo pueden transitar sobre el lattice predefinido. Por ello, se opta por usar un planeador no holonómico basado en la creación de árbol de configuración. Para esta tesis se utilizará el algoritmo de A Estrella Híbrido (Hybrid A*).

Por otro lado, la forma en que se suele incluir el tiempo en un planeador local, es simplemente extendiendo en el vector de configuración con la variable tiempo, donde las acciones de espera, son realizadas con nodos de retardo; es decir si el móvil posee M movimientos, generará M+1 nodos, donde dicho “1” es la acción de retardo. Sin embargo, si el móvil debe esperar largos tiempo en un punto del mapa, esto conlleva a que se generen un número exponencial de nodos con respecto al tiempo de espera; por ejemplo, si cada retardo toma un segundo y el móvil debe esperar diez segundos en un punto del mapa, se generará en el primer segundo $M+1$ nodos, luego en el siguiente segundo se generará $M+1$ nodos para cada nodo del segundo anterior, así hasta generarse $\sum_{k=0}^{10}(M+1)^k$ nodos, donde diez de estos son los nodos de espera y el resto no serán parte de la solución, pues son nodos que implican que el móvil se mueve antes de los diez segundos. Una manera alternativa a planear en el tiempo es utilizando nodos SIPP, los cuales son los nodos utilizados en el algoritmo de planeamiento de caminos en intervalos seguros

[43]. Donde cada nodo describe el intervalo de tiempo en el que se puede ocupar una configuración de manera segura sin colisionar con ningún objeto. De utilizarse dichos nodos, no se generan nodos de espera, pues la espera se realiza de forma implícita; por ejemplo, si un móvil llega en el instante 9 a un nodo SIPP que tiene un intervalo seguro de [0, +∞) (es decir, es seguro en todo instante) y este tiene un siguiente nodo SIPP con un intervalo seguro de [19, 25], la transición del nodo actual al siguiente indica implícitamente que el móvil debe esperar 10 segundos en el nodo actual antes de pasar al siguiente. Dado que la utilización de nodos SIPP reduce el número de nodos generados, se combinará al planeador local elegido con el de planeamiento SIPP.

Resumiendo, para la etapa de planeamiento, se planea utilizar como planeador global multi agente al algoritmo CBS, y se planea utilizar como planeador local no holonómico, al algoritmo de A Estrella Híbrido e incluirá el planeamiento en el tiempo en dicho algoritmo combinando a este con el de planeamiento de caminos en intervalos seguros. Todos los algoritmos de planeamiento serán explicados a detalle en el capítulo de planeamiento.

Dado que el algoritmo CBS, retorna ya de por sí los caminos en el tiempo de cada móvil, bastaría con optimizar dichos caminos, tal que se reduzca la longitud de los mismos y sean transitables de manera suave por los móviles. Para ello, se propone utilizar optimización por descenso de gradiente de manera desacoplada para cada móvil, es decir optimizando de manera semi simultánea los caminos de cada uno por separado, pero compartiendo las posiciones de los mismos para realizar la evasión de colisiones entre móviles. Mientras que, para el controlador de cada móvil, se propone utilizar un controlador LQR basado en el modelo linealizado del móvil no holonómico

1.5 Objetivos de la Tesis

Sabiéndose que la gran mayoría de robots móviles no pueden moverse de manera instantánea en cualquier dirección, ya que poseen restricciones cinemáticas; es decir, son no holonómicos; y que, además, las disposiciones y formas de obstáculos del mapa pueden generar situaciones de atasco, se busca diseñar un sistema de planeamiento de trayectoria multi robot coordinado en el tiempo para vehículos no holonómicos, tal que permita a los móviles navegar sin atascos en mapas con presencia de obstáculos variados.

- **Objetivo General:**

  Diseñar un sistema de planeamiento de trayectorias multi robot con coordinación en el tiempo para robots móviles no holonómicos, capaz de evitar colisiones con obstáculos y evitar atascos entre los móviles, utilizando algoritmos de optimización y de planeamiento (CBS, SIPP y Hybrid A*), y diseñar el sistema de control de seguimiento de trayectoria de cada robot.

- **Objetivos Específicos:**

  - Obtener el modelo matemático del robot móvil no holonómico.
  - Diseñar el planeador local para robots no holonómicos.
  - Diseñar el planeador global para solución de conflictos entre robots.
  - Diseñar el generador de trayectorias utilizando los caminos hallados por el planeador.
  - Diseñar el sistema de control de seguimiento de trayectoria.
  - Integrar el planeador multi robot, el generador de trayectorias y el sistema de control de seguimiento de trayectorias.
  - Validar el funcionamiento del sistema en mapas variados.

Capítulo 2: Modelamiento

El presente capítulo desarrolla el modelamiento cinemático y dinámico de un móvil tipo Ackerman, usando el modelo bicicleta. En este modelo, el móvil es controlado por dos parámetros, los cuales son la aceleración de avance y el ángulo de giro del timón. Este móvil es de tipo no holonómico, pues las ecuaciones que describen su movimiento lo limitan a moverse en un espacio restringido, impidiendo la posibilidad de desplazarse en cualquier dirección de manera instantánea. Los modelamientos realizados a continuación toman como referencia los modelamientos en [44] y [45].

2.1 Modelo Cinemático

En Figura 2.1, se muestra la imagen del modelo del robot móvil no holonómico a modelar, donde β es el ángulo de deslizamiento, θ es la orientación del móvil, δ_f el ángulo de giro de la rueda frontal, δ_r el ángulo de giro de la rueda posterior, R el radio de giro, V la velocidad del móvil y C es el centro de gravedad del mismo.

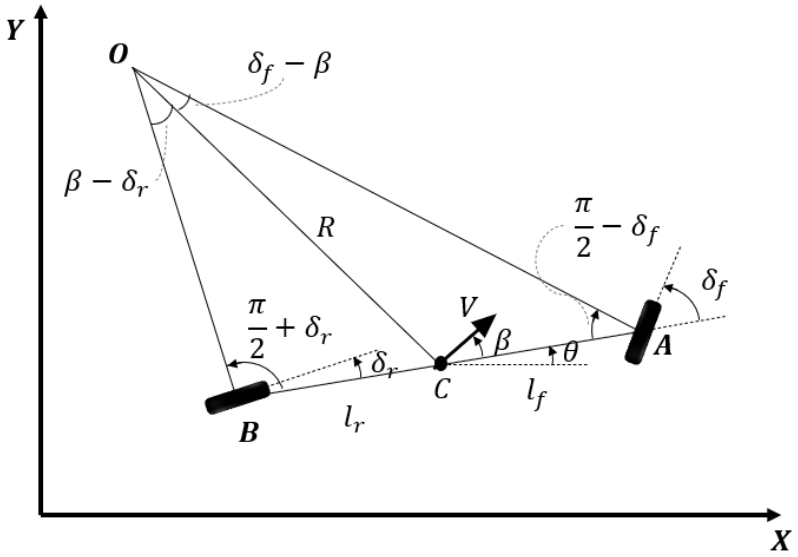


Figura 2.1: Representación de modelo cinemático del móvil [Elaboración propia].

Para el modelo mostrado, los parámetros conocidos son el ángulo de giro frontal y posterior, mientras que los parámetros que se desean conocer son la posición del móvil, el ángulo de giro y el ángulo de deslizamiento. Estos parámetros pueden ser estimados de manera discreta si se conoce sus derivadas, a continuación, se procede a hallar dichas derivadas.

Aplicando la ley de senos en el triángulo OCA se obtiene lo siguiente:

$$\frac{\sin(\delta_f - \beta)}{l_f} = \frac{\sin\left(\frac{\pi}{2} - \delta_f\right)}{R} \quad (29)$$

$$\frac{\cos(\delta_r) * \sin(\beta) - \cos(\beta) * \sin(\delta_r)}{l_r} = \frac{\cos(\delta_r)}{R} \quad (30)$$

$$\sin(\beta) - \cos(\beta) * \tan(\delta_r) = \frac{l_r}{R} \quad (31)$$

Mientras que aplicando la ley de senos en el triángulo OCB se tiene lo siguiente:

$$\frac{\sin(\beta - \delta_r)}{l_r} = \frac{\sin\left(\frac{\pi}{2} + \delta_r\right)}{R} \quad (32)$$

$$\frac{\sin(\delta_f) * \cos(\beta) - \sin(\beta) * \cos(\delta_f)}{l_f} = \frac{\cos(\delta_f)}{R} \quad (33)$$

$$\tan(\delta_f) * \cos(\beta) - \sin(\beta) = \frac{l_f}{R} \quad (34)$$

Luego, sumando ambos se llega a la siguiente ecuación:

$$\sin(\beta) - \cos(\beta) * \tan(\delta_r) + \tan(\delta_f) * \cos(\beta) - \sin(\beta) = \frac{l_f + l_r}{R} \quad (35)$$

$$\cos(\beta) * (\tan(\delta_f) - \tan(\delta_r)) = \frac{l_f + l_r}{R} \quad (36)$$

Sabiendo que, a bajas velocidades, la velocidad angular del móvil es igual a la velocidad del cambio de orientación, se propone la siguiente igualdad:

$$\dot{\theta} = \omega \quad (37)$$

$$\dot{\theta} = \frac{V}{R} \quad (38)$$

Luego reemplazando esto en la ecuación (36), se tiene:

$$\cos(\beta) * (\tan(\delta_f) - \tan(\delta_r)) = \frac{\dot{\theta}}{V} * (l_f + l_r) \quad (39)$$

$$\dot{\theta} = \frac{V}{l_f + l_r} * \cos(\beta) * (\tan(\delta_f) - \tan(\delta_r)) \quad (40)$$

En Figura 2.1, se puede notar que la velocidad de desplazamiento en los ejes X e Y son los siguientes:

$$\dot{X} = V * cos(\theta + \beta) \quad (41)$$

$$\dot{Y} = V * sin(\theta + \beta) \quad (42)$$

Mientras que el ángulo de deslizamiento β se puede hallar de la siguiente manera:

$$l_f * (\sin(\beta) - \cos(\beta) * \tan(\delta_r)) = l_r * (\tan(\delta_f) * \cos(\beta) - \sin(\beta)) \quad (43)$$

$$\beta = \operatorname{atan}\left(\frac{l_r * \tan(\delta_f) + l_f * \tan(\delta_r)}{l_f + l_r}\right) \quad (44)$$

Luego, las ecuaciones cinemáticas del móvil quedan definidas por:

$$\dot{X} = V * cos(\theta + \beta) \quad (45)$$

$$\dot{Y} = V * sin(\theta + \beta) \quad (46)$$

$$\dot{V} = a \quad (47)$$

$$\dot{\theta} = \frac{V}{l_f + l_r} * \cos(\beta) * \left(\tan\left(\delta_f\right) - \tan(\delta_r)\right) \quad (48)$$

$$\beta = \text{atan}\left(\frac{\left(l_r * tan\left(\delta_f\right) + l_f * tan(\delta_r)\right)}{\left(l_f + l_r\right)}\right) \quad (49)$$

2.2 Modelo Dinámico

El modelamiento de la dinámica del móvil es dividido en modelamiento lateral y longitudinal, a continuación, se detalla cada uno de ellos.

2.2.1 Modelamiento Lateral

En Figura 2.2, se muestra a las fuerzas que actúan de manera lateral en el móvil, en el eje "y" de su marco de referencia local. Donde F_{yf} y F_{yr} son las fuerzas resultantes en el la rueda frontal y posterior respectivamente, causadas por el movimiento del móvil.

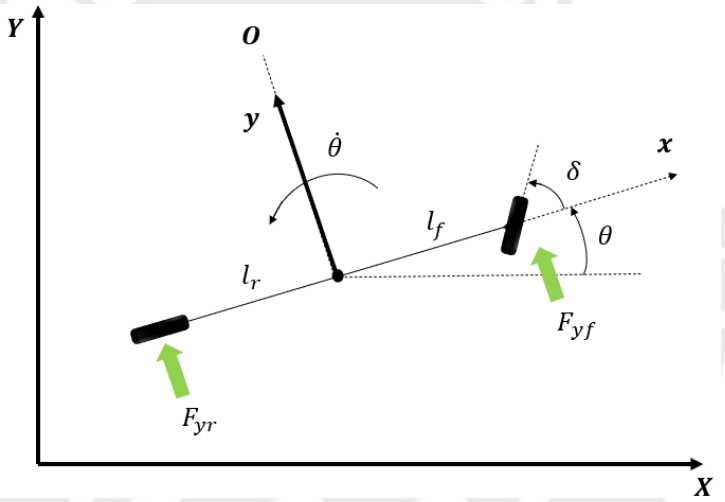


Figura 2.2: Representación de fuerzas laterales que actúan en el móvil [Elaboración propia].

Haciendo un balance de fuerzas, se llega a los siguiente:

$$m * a_y = F_{yf} + F_{yr} \quad (50)$$

Así mismo, la aceleración resultante en la dirección Y resulta de la aceleración del desplazamiento en Y más la aceleración centrípeta. Así se tiene lo siguiente:

$$a_y = \ddot{y} + \frac{V_x^2}{R} = \ddot{y} + V_x * \omega \quad (51)$$

Como $\omega = \dot{\theta}$, se tiene lo siguiente:

$$a_y = \ddot{y} + V_x * \dot{\theta} \quad (52)$$

Luego, reemplazando esto en la ecuación de fuerzas, se tiene lo siguiente:

$$m * \left(\ddot{y} + V_x * \dot{\theta}\right) = F_{yf} + F_{yr} \quad (53)$$

Por otro lado, la ecuación de momentos del móvil queda definida de la siguiente manera:

$$I_Z\dot{\omega} = l_f * F_{yf} - l_r * F_{yr} \quad (54)$$

$$I_Z\ddot{\theta} = l_f * F_{yf} - l_r * F_{yr} \quad (55)$$

Luego, se procede a modelar las fuerzas laterales en cada rueda, en función al ángulo de deslizamiento α que existe en cada una de ellas, ver Figura 2.3.

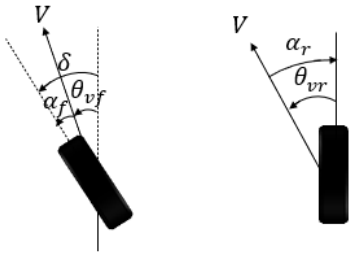


Figura 2.3: Ángulos de deslizamiento en rueda frontal y posterior [Elaboración propia].

De la figura, se puede notar que, el ángulo de deslizamiento de la rueda frontal y posterior son de la siguiente manera:

$$\alpha_f = \delta - \theta_{vf} \quad (56)$$

$$\alpha_r = -\theta_{vr} \quad (57)$$

Asumiendo un modelo de rueda lineal, la fuerza lateral en cada rueda del móvil será proporcional a sus ángulos de deslizamiento y a sus coeficientes de rigidez lateral $C_{\alpha f}$ y $C_{\alpha r}$ de la rueda frontal y posterior respectivamente.

$$F_{yf} = 2 * C_{\alpha f} * \alpha_f = 2 * C_{\alpha f} * (\delta - \theta_{vf}) \quad (58)$$

$$F_{yr} = 2 * C_{\alpha r} * \alpha_r = 2 * C_{\alpha r} * (-\theta_{vr}) \quad (59)$$

Así mismo, los ángulos θ_{vf} y θ_{vr}, se pueden obtener de la relación de la velocidad lateral V_y, longitudinal V_x y de giro $l * \dot{\theta}$ de la siguiente forma:

$$\tan(\theta_{vf}) = \frac{V_y + l_f * \dot{\theta}}{V_x} \quad (60)$$

$$\tan(\theta_{vr}) = \frac{V_y - l_r * \dot{\theta}}{V_x} \quad (61)$$

Luego, si los ángulos de deslizamiento θ_{vf} y θ_{vr} son pequeños, se puede aproximar la tangente al ángulo, así se tendría lo siguiente:

$$\theta_{vf} = \frac{V_y + l_f * \dot{\theta}}{V_x} = \frac{\dot{y} + l_f * \dot{\theta}}{V_x} \quad (62)$$

$$\theta_{vr} = \frac{V_y - l_r * \dot{\theta}}{V_x} = \frac{\dot{y} - l_r * \dot{\theta}}{V_x} \quad (63)$$

Luego, reemplazando los valores hallados en la ecuación de fuerza:

$$m * (\ddot{y} + V_x * \dot{\theta}) = 2 * C_{\alpha f} * \left(\delta - \frac{\dot{y} + l_f * \dot{\theta}}{V_x}\right) + 2 * C_{\alpha r} * \left(-\frac{\dot{y} - l_r * \dot{\theta}}{V_x}\right) \quad (64)$$

$$\ddot{y} = -\frac{2 * (C_{\alpha f} + C_{\alpha r})}{m * V_x} * \dot{y} + \left(-V_x - \frac{2 * (C_{\alpha f} * l_f - C_{\alpha r} * l_r)}{m * V_x}\right) * \dot{\theta} + 2 * \frac{C_{\alpha f}}{m} * \delta \quad (65)$$

Luego, reemplazando los valores hallados en la ecuación de momentos:

$$I_Z \ddot{\theta} = l_f * 2 * C_{\alpha f} * \left(\delta - \frac{\dot{y} + l_f * \dot{\theta}}{V_x}\right) - l_r * 2 * C_{\alpha r} * \left(-\frac{\dot{y} - l_r * \dot{\theta}}{V_x}\right) \quad (66)$$

$$\ddot{\theta} = -\frac{2 * (l_f * C_{\alpha f} - l_r * C_{\alpha r})}{I_Z * V_x} * \dot{y} - \frac{2 * (l_f^2 * C_{\alpha f} + l_r^2 * C_{\alpha r})}{I_Z * V_x} * \dot{\theta} + \frac{2 * l_f * C_{\alpha f}}{I_Z} * \delta \quad (67)$$

Finalmente, expresando las ecuaciones en una matriz de espacio estado, se tiene:

$$\begin{bmatrix} \dot{y} \\ \ddot{y} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2 * (C_{\alpha f} + C_{\alpha r})}{m * V_x} & 0 & -V_x - \frac{2 * (C_{\alpha f} * l_f - C_{\alpha r} * l_r)}{m * V_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2 * (l_f * C_{\alpha f} - l_r * C_{\alpha r})}{I_Z * V_x} & 0 & -\frac{2 * (l_f^2 * C_{\alpha f} + l_r^2 * C_{\alpha r})}{I_Z * V_x} \end{bmatrix} * \begin{bmatrix} y \\ \dot{y} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 2 * \frac{C_{\alpha f}}{m} \\ 0 \\ \frac{2 * l_f * C_{\alpha f}}{I_Z} \end{bmatrix} * \delta \quad (68)$$

2.2.2 Modelamiento Longitudinal

En Figura 2.4, se muestra las fuerzas longitudinales que actúan sobre el móvil cuando este se encuentra en una superficie inclinada. Donde m es la masa del vehículo, g es la gravedad, γ es el ángulo de inclinación de la superficie, F_{aero} es la fuerza longitudinal de arrastre aerodinámico, F_{xf} y F_{xr} son las fuerzas longitudinales de las ruedas frontales y posteriores respectivamente y R_{xf} y R_{xr} son las fuerzas de resistencia al rodamiento de las ruedas frontales y posteriores respectivamente.

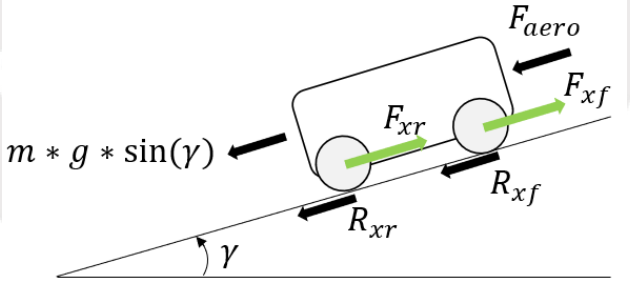


Figura 2.4: Representación de fuerzas longitudinales que actúan en un móvil [Elaboración propia].

Realizando el balance de fuerzas para el móvil de la figura, se tiene lo siguiente:

$$m * \ddot{x} = F_{xf} + F_{xr} - F_{aero} - R_{xf} - R_{xr} - m * g * \sin(\gamma) \quad (69)$$

Luego, se define la fuerza de tracción F_x y de resistencia R_x como:

$$F_x = F_{xf} + F_{xr} = m * a_x \quad (70)$$

$$R_x = R_{xf} + R_{xr} \quad (71)$$

Asumiendo que el ángulo de inclinación γ es pequeño, se llega a lo siguiente:

$$m * \ddot{x} = F_x - F_{aero} - R_x - m * g * \gamma \quad (72)$$

Donde, la fuerza aerodinámica depende de la velocidad del móvil, del área frontal del mismo y de la densidad del aire, y se rige por la siguiente ecuación:

$$F_{aero} = 0.5 * \rho * C_\alpha * A_F * \dot{x}^2 = c_\alpha * \dot{x}^2 \quad (73)$$

Mientras que la resistencia de rodamiento depende de la fuerza normal que se ejerce en la rueda, la presión de la misma y la velocidad del móvil.

$$R_x \approx c_r |\dot{x}| \quad (74)$$

Finalmente reemplazado estos valores en la ecuación de fuerzas, se obtiene:

$$\ddot{x} = a_x - \frac{c_\alpha}{m} * \dot{x}^2 - \frac{c_r}{m} |\dot{x}| - g * \gamma \quad (75)$$

2.3 Selección del modelo

La presente tesis es diseñada para móviles que se desplazan a bajas velocidades en un plano bidimensional y con deslizamiento nulo, por ello, bastará con utilizar el modelo cinemático del móvil para verificar que el sistema funciona.

Capítulo 3: Planeamiento

3.1 Representación del Mapa

Los mapas pueden ser representados principalmente de tres maneras [9]. La primera es la representación del mapa en grafos (Figura 3.1 b), la segunda es la representación discretizada del mapa en una grilla (Figura 3.1 c) y la tercera es la representación con funciones potenciales (Figura 3.1 d).

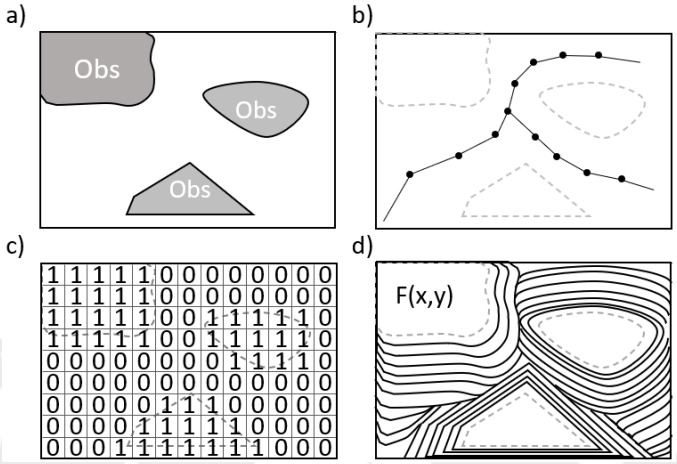


Figura 3.1: a) Mapa con obstáculos, b) Representación de mapa en grafos, c) Representación de mapa en grilla, d) representación de mapa con funciones potenciales [Elaboración propia].

Para realizar el planeamiento con múltiples móviles, se requiere de un tipo de mapa que permita conocer con certeza si estos son capaces de navegar en el mapa sin colisionar entre ellos ni con el entorno. En esta tesis, el mapa utilizado para verificar las colisiones es un mapa de tipo grilla, obtenido a partir de la imagen binaria del mismo. Mientas que, el mapa utilizado para el planeamiento no holonómico, es un árbol de configuraciones (un mapa de grafos) que se forma conforme se desplaza el móvil, utilizando las ecuaciones de movimiento circular del mismo [37][38].

3.2 Planeamiento Local

Para elegir el planeador local, se debe considerar el tipo de robot móvil a utilizar. En esta tesis se trabaja con vehículos no holonómicos, por ello, se elige un planeador que respete la cinemática del móvil, el cual lleve al móvil de un estado origen definido por la posición y orientación inicial del móvil a un estado final definido por la posición y orientación final del móvil. Así mismo, en esta tesis el planeador local servirá para hallar el mejor camino que puede transitar el móvil para llegar a su destino.

3.2.1 Algoritmo Hybrid A*

Para vehículos no holonómicos, el planeador más utilizado en la industria automotriz es el de Hybrid A star [37][38], el cual consiste en utilizar el algoritmo de A estrella con nodos que representan la posición y orientación del móvil en el mapa, los cuales son calculados usando las ecuaciones cinemáticas del móvil. Teniéndose así un vector de estado de cada nodo de la forma (x, y, theta). Ver Figura 3.2.

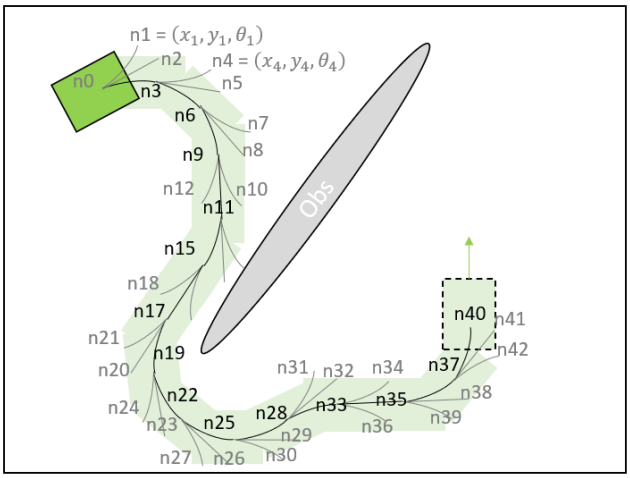


Figura 3.2: Representación de la exploración de un móvil usando Hybrid A Star [Elaboración propia].

Así mismo, el algoritmo Hybrid A star se caracteriza por utilizar dos heurísticas, una holonómica y otra no holonómica, para determinar el costo “h” de lo que falta para llegar a la meta. La heurística holonómica, describe la distancia que debe recorrer un móvil no holonómico para llegar desde un punto cualquiera del mapa al punto de destino, considerando los obstáculos del mapa, pero sin considerar las restricciones físicas del móvil. Ver Figura 3.3 (a). Mientras que la heurística no holonómica, describe la distancia o tiempo que debe recorrer un móvil no holonómico para llegar al punto de destino, sin considerar los obstáculos del mapa, pero considerando las restricciones físicas del móvil ver Figura 3.3 (b).

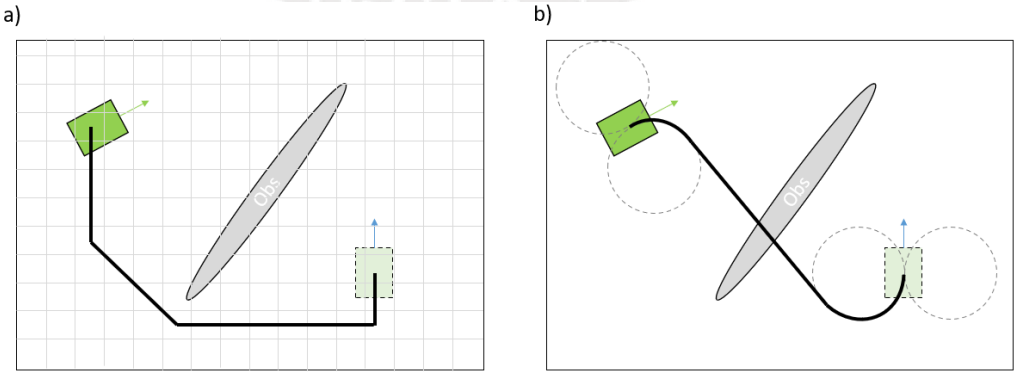


Figura 3.3: a) Distancia holonómica entre un móvil y su destino, b) Distancia no holonómica entre un móvil y su destino [Elaboración propia].

La razón de utilizar dos heurísticas es con el fin de tener un costo heurístico más próximo al costo real de lo que falta recorrer para llegar al destino. Así, cada nodo tendrá un costo “f” más cercano al real. Esto tiene como consecuencia que el algoritmo de A estrella sea mucho más veloz, ya que este algoritmo de búsqueda se basa en explorar el mapa usando los nodos de menor costo, y al ser los costos más cercanos a la realidad, se evita que se evalúen nodos innecesarios.

Cada nodo es el resultado de aplicar un valor de giro distinto (steering) partiendo de la configuración actual del móvil. Por ejemplo, si el móvil tiene un rango de giro entre -40 y 40 y se discretiza dicho rango en n valores, se generarán n siguientes nodos. Ver Figura 3.4

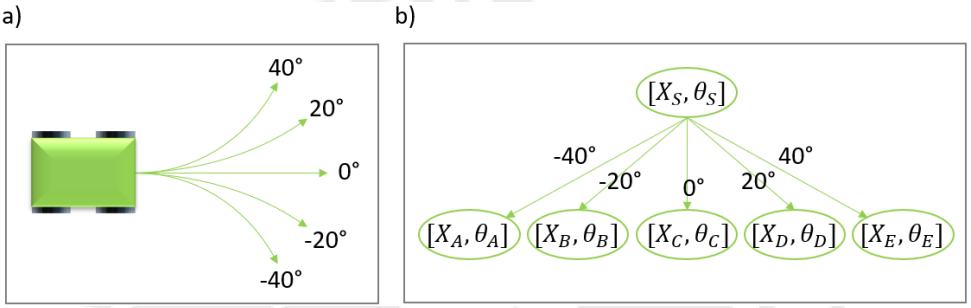


Figura 3.4: a) Discretización de giros de robot no holonómico, b) Representación en nodos de configuraciones resultantes de giros de robot no holonómico [Elaboración propia].

Por otro lado, generar nodos por cada posición y orientación del móvil en el espacio cartesiano, resulta computacionalmente costoso, pues se generarían infinitos nodos, entre los cuales se tiene que buscar la solución. Por ello, se discretiza el espacio y la orientación del móvil, navegándose así en una grilla con dimensiones definidas, donde cada celda describe al móvil solo en “m” orientaciones distintas ver Figura 3.5. Tanto la orientación como la posición del móvil son discretizados a estos valores. Así, si el movimiento del móvil genera una configuración discretizada que es igual a una previamente calculada, se omite la creación de dicho nodo, pues ya existiría uno que representa dicha configuración.

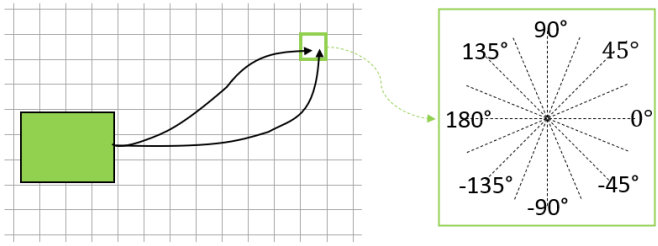


Figura 3.5: Representación de discretización de la posición y orientación de un móvil en el espacio cartesiano [Elaboración propia].]

3.2.2 Diseño del Planeador Local

a) Cálculo de la distancia holonómica

Para calcular la distancia heurística holonómica, se utilizó el algoritmo de Dijkstra de manera offline; es decir, antes de ejecutar el algoritmo de Hybrid A star. Proveyendo como nodo de entrada el nodo que describe la celda que contiene al punto destino del móvil y como nodo objetivo a la celda que contiene al punto de origen del móvil. Ver Figura 3.6.

```
funcion creaMatrizDeDistanciaHolonomica(nodoCelda):
    crea lista Q vacía
    agrega nodoCelda a Q
    mientras Q no esté vacía:
        nodoDeMenorCosto = obtenNodoDeMenorCostoDe(Q)
        remueve nodoDeMenorCosto de Q
        nodosVecinos = obtenCeldasVecinasDe(nodoDeMenorCosto)
        para cada nodoVecino en nodosVecinos:
            costoGtentativo = g[nodoDeMenorCosto] +
                              costoDeIrDe(nodoDeMenorCosto,nodoVecino)
            si g[nodoVecino] > costoGtentativo:
                g[nodoVecino] = costoGtentativo
                h_holonomico[nodoVecino.y][nodoVecino.x] = costoGtentativo
            agrega nodoVecino a Q
    retorna h_holonomico
```

Figura 3.6: Pseudocódigo de creación de matriz de distancia holonómica [Elaboración propia].

Donde los siguientes nodos describen a las celdas adjuntas. Ver Figura 3.7.

```
funcion obtenCeldasVecinasDe(nodoCelda):
    crea lista CELDASVECINAS vacía
    desviacionesXY = {{-1,-1},{-1,0},{-1,1},{0,1},{1,1},{1,0},{1,-1},{0,-1}}
    para cada desviacionXY en desviacionesXY:
        x = nodoCelda.x + desviacionXY(1)
        y = nodoCelda.y + desviacionXY(2)
        nuevoNodoCelda = creaNodoCelda(x,y)
        agrega nuevoNodoCelda a CELDASVECINAS
    retorna CELDASVECINAS
```

Figura 3.7: Pseudocódigo de creación de nodos que describen celdas adjuntas [Elaboración propia].

La matriz h_holonomic resultante de ejecutar el código de Figura 3.6, resultaría de la forma de Figura 3.8 y por ende para saber la distancia al punto destino, solo bastaría con leer el valor de h_holonomic sobre el cual se encuentra el móvil.

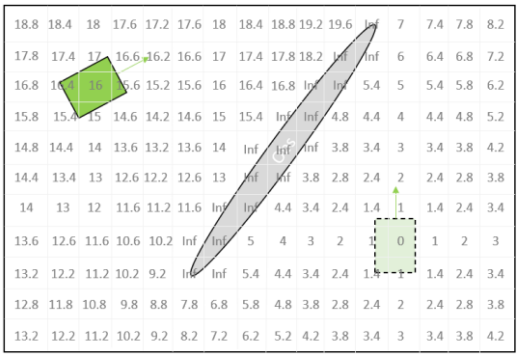


Figura 3.8: Visualización de matriz holonómica [Elaboración propia].

b) Cálculo de la distancia no holonómica

Por otro lado, la heurística no holonómica es calculada de manera “online”; es decir se calcula durante la creación del nodo, según la posición y orientación del móvil. Para esto, se calcula las curvas de Dubins, las cuales describen los caminos más cortos que puede recorrer un móvil no holonómico para llegar a una configuración de posición y orientación destino, conociéndose el radio mínimo de giro del móvil. Estas pueden ser de 6 tipos, las cuales son llamadas como RSR, LSL, RSL, LSR, LRL y RLR [46]. Donde R significa giro a la derecha, S significa movimiento recto y L significa giro a la izquierda.

Cuando la distancia entre los móviles es mayor a 4 veces el radio mínimo de giro, se utiliza las curvas RSR, LSL, RSL o LSR. Para hallar dichas curvas se debe dibujar circunferencias a cada lado del móvil, en el punto de origen y destino, las cuales representan el giro de menor radio del móvil, y luego unir las circunferencias del punto origen con el destino usando rectas tangentes, como en Figura 3.9, la cual muestra las tangentes entre dos de estas circunferencias. Así, Las curvas quedarán definidas por los puntos tangentes en cada circunferencia. Por ello, se debe hallar dichos puntos y luego calcular la longitud total de la curva para obtener la distancia no holonómica entre el punto origen y el punto destino.

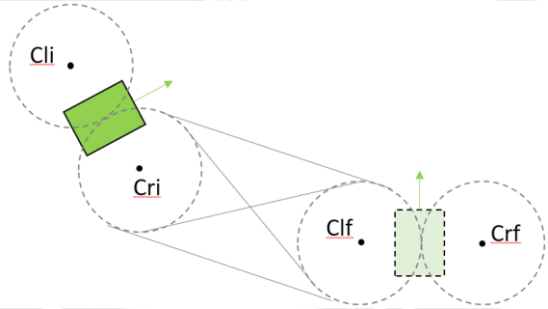


Figura 3.9: Visualización de circunferencias y tangentes de un móvil en su punto de inicio y destino [Elaboración propia].

Para el caso de las curvas RSR o LSL, se debe analizar las rectas tangentes exteriores de cada circunferencia, ver Figura 3.10.

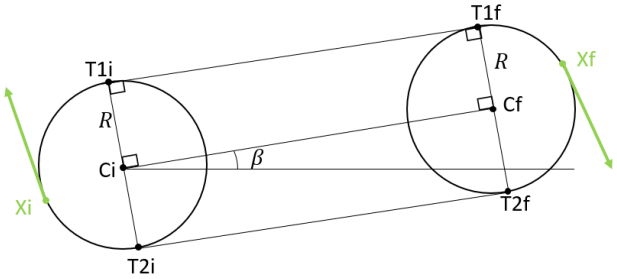


Figura 3.10: Representación geométrica de rectas tangentes exteriores [Elaboración propia].

El cálculo procede de la siguiente forma:

$$\beta = atan2\left(Cf_y - Ci_y, Cf_x - Ci_x\right) \quad (76)$$

$$T1i = Ci + R * \begin{bmatrix}\cos(\beta + 0.5 * \pi)\\ \sin(\beta + 0.5 * \pi)\end{bmatrix} \quad (77)$$

$$T2i = Ci + R * \begin{bmatrix}\cos(\beta - 0.5 * \pi)\\ \sin(\beta - 0.5 * \pi)\end{bmatrix} \quad (78)$$

$$dirVect = Cf - Ci \quad (79)$$

$$T1f = dirVect + T1i \quad (80)$$

$$T2f = dirVect + T2i \quad (81)$$

Mientras que, si la curva es de tipo RSL o LSR, se debe analizar las rectas tangentes interiores, ver Figura 3.11.

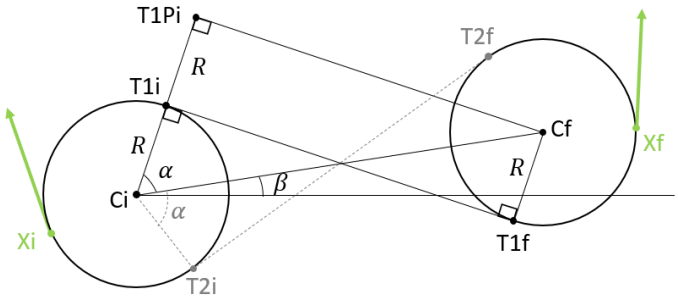


Figura 3.11: Representación geométrica de rectas tangentes interiores [Elaboración propia].

Luego, se procede de la siguiente forma, para la recta tangente superior:

$$D = norm(Cf - Ci) \quad (82)$$

$$\beta = atan2\left(Cf_y - Ci_y, Cf_x - Ci_x\right) \quad (83)$$

$$\alpha = \operatorname{acos}\left(2 * \frac{R}{D}\right) \quad (84)$$

$$T1i = Ci + R * \begin{bmatrix}\cos(\beta + \alpha)\\ \sin(\beta + \alpha)\end{bmatrix} \quad (85)$$

$$T1Pi = Ci + 2 * R * \begin{bmatrix}\cos(\beta + \alpha)\\ \sin(\beta + \alpha)\end{bmatrix} \quad (86)$$

$$dirVect = Cf - T1Pi \quad (87)$$

$$T1f = T1i + dirVect \quad (88)$$

Y de la misma manera para la otra recta tangente inferior:

$$T2i = Ci + R * \begin{bmatrix}\cos(\beta - \alpha)\\ \sin(\beta - \alpha)\end{bmatrix} \quad (89)$$

$$T2Pi = Ci + 2 * R * \begin{bmatrix}\cos(\beta - \alpha)\\ \sin(\beta - \alpha)\end{bmatrix} \quad (90)$$

$$dirVect = Cf - T2Pi \quad (91)$$

$$T2f = T2i + dirVect \quad (92)$$

Luego la distancia holonómica quedará definida por la curva de menor longitud, en el caso de Figura 3.11, esta es la curva RSL. La longitud de dicha curva, es calculada de la siguiente manera:

$$V0 = Xi - Ci \quad (93)$$

$$V1 = T1i - Ci \quad (94)$$

$$V2 = T1f - Cf \quad (95)$$

$$V3 = Xf - Cf \quad (96)$$

$$angi = atan2(V1_y, V1_x) - atan2(V0_y, V0_x) \quad (97)$$

$$angf = atan2(V3_y, V3_x) - atan2(V2_y, V2_x) \quad (98)$$

$$lengthRi = abs(angi) * R \quad (99)$$

$$lengthRf = abs(angf) * R \quad (100)$$

$$lengthS = norm(T1f - T1i) \quad (101)$$

$$h_{holonomic} = lengthRi + lengthRf + lengthS \quad (102)$$

Por otro lado, si la distancia entre los móviles es menor a 4 veces el radio mínimo de giro, se utiliza las curvas tipo RLR o LRL. Para la obtención de estas curvas, se utiliza las dos circunferencias más cercanas entre los móviles y se agrega circunferencias tangentes a estas mismas. Ver Figura 3.12.

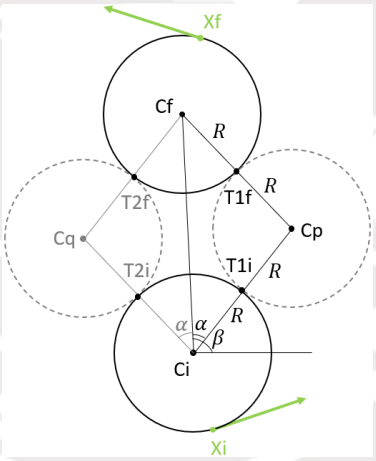


Figura 3.12: Representación geométrica de circunferencias tangentes agregadas [Elaboración propia].

Y luego, se procede de la siguiente manera:

$$D = norm(Cf - Ci) \quad (103)$$

$$\alpha = \text{acos}\left(\frac{D}{4 * R}\right) \quad (104)$$

$$V1 = Cf - Ci \quad (105)$$

$$\beta = atan2(V1_y, V1_x) \quad (106)$$

$$Cp = Ci + 2 * R * \cos(\beta - \alpha) \quad (107)$$

$$Cq = Ci + 2 * R * \cos(\beta + \alpha) \quad (108)$$

$$T1i = Cp + R * \frac{Ci - Cp}{norm(Ci - Cp)} \quad (109)$$

$$T1f = Cp + R * \frac{Cf - Cp}{norm(Cf - Cp)} \quad (110)$$

$$T2i = Cq + R * \frac{Ci - Cq}{norm(Ci - Cq)} \quad (111)$$

$$T2f = Cq + R * \frac{Cf - Cq}{norm(Cf - Cq)} \quad (112)$$

Luego la distancia holonómica quedará definida por la curva de menor longitud, en el caso de Figura 3.12 esta es la curva LRL. La longitud de dicha curva, es calculada de la siguiente manera:

$$V0 = Xi - Ci \quad (113)$$

$$V1 = T1i - Ci \quad (114)$$

$$V2 = T1i - Cp \quad (115)$$

$$V3 = T1f - Cp \quad (116)$$

$$V4 = T1f - Cf \quad (117)$$

$$V5 = Xf - Cf \quad (118)$$

$$angi = atan2(V1_y, V1_x) - atan2(V0_y, V0_x) \quad (119)$$

$$angp = atan2(V3_y, V3_x) - atan2(V2_y, V2_x) \quad (120)$$

$$angf = atan2(V5_y, V5_x) - atan2(V4_y, V4_x) \quad (121)$$

$$lengthLi = abs(angi) * R \quad (122)$$

$$lengthRf = abs(angp) * R \quad (123)$$

$$lengthLf = abs(angf) * R \quad (124)$$

$$h_{nonholonomic} = lengthLi + lengthRp + lengthLf \quad (125)$$

Ya que existen 6 tipos de curvas, se debe obtener la longitud de cada una de estas y luego compararlas para obtener la curva de menor longitud. La longitud de dicha curva es asignada como la distancia no holonómica. El pseudocódigo de la función que halla la distancia no holonómica del móvil, entre una configuración incial y una final se muestra en Figura 3.13.

```
1  funcion h_noholonomico(confInicial, confFinal):
2      l1 = obtenLongitud_RSR(confInicial, confFinal)
3      l2 = obtenLongitud_LSL(confInicial, confFinal)
4      l3 = obtenLongitud_RSL(confInicial, confFinal)
5      l4 = obtenLongitud_LSR(confInicial, confFinal)
6      l5 = obtenLongitud_RLR(confInicial, confFinal)
7      l6 = obtenLongitud_LRL(confInicial, confFinal)
8     menorLongitud = min(l1, l2, l3, l4, l5, l6)
9     retorna menorLongitud
```

Figura 3.13: Pseudocódigo de distancia holonómica entre dos configuraciones [Elaboración propia].

c) Cálculo de las siguientes configuraciones

Dado que, las ecuaciones cinemáticas del modelo, son válidas cuando el tiempo de muestreo es muy pequeño y que lo que se busca en esta tesis con el planeador local es verificar en el menor tiempo posible que el camino hallado es plausible de realizar por el móvil, se prefiere utilizar las ecuaciones de movimiento circular en vez de las cinemáticas, ya que realizar el cálculo de estas últimas con un tiempo pequeño requiere múltiples iteraciones y, por ende, mayor tiempo de cómputo. Las ecuaciones de movimiento circular se pueden definir a partir de Figura 3.14.

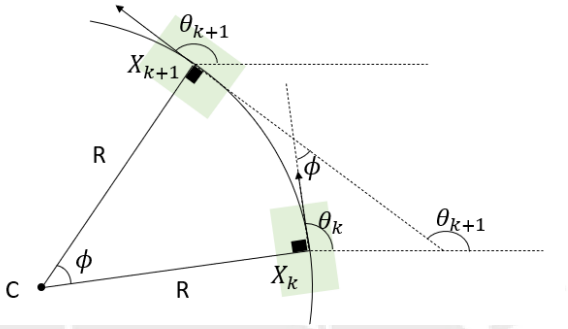


Figura 3.14: Representación geométrica del giro de un móvil [Elaboración propia].

Para un vehículo tipo bicicleta que se desplaza una distancia d con un ángulo de giro del timón δ, el radio de la circunferencia que describe se calcula como:

$$R = \frac{d}{\tan(\delta)} \quad (126)$$

Mientras que el ángulo ϕ descrito por su desplazamiento se calcula como:

$$\phi = \frac{d}{R} \quad (127)$$

Luego, la orientación final del móvil, se puede deducir por geometría, la cual quedaría definido por la suma de la orientación inicial y el ángulo de giro.

$$\theta_{k+1} = \theta_k + \phi \quad (128)$$

Mientras que, para hallar la posición final del móvil, bastará con encontrar el centro C de la circunferencia de radio de giro R y agregarle el vector que va de C a X_{k+1}.

$$C = X_k + R * \begin{bmatrix} \cos(0.5 * \pi + \theta_k) \\ \sin(0.5 * \pi + \ \theta_k) \end{bmatrix} \quad (129)$$

$$X_{k+1} = C - R * \begin{bmatrix} \cos(0.5 * \pi + \theta_{k+1}) \\ \sin(0.5 * \pi + \theta_{k+1}) \end{bmatrix} \quad (130)$$

Donde R es el radio de giro del móvil, (Cx, Cy) son las coordenadas del centro de giro, θ_k es la orientación del móvil en instante k, ϕ es el ángulo de giro del móvil del instante k al instante k+1 y d es el desplazamiento del móvil entre el instante k y el instante k+1. El pseudocódigo del algoritmo que calcula el desplazamiento circular del móvil se muestra en Figura 3.15.

```
funcion calculaSiguienteConfiguracion(nodo,d,angDeGiro):
    radio = d/angDeGiro
    cx    = nodo.x + radio*cos(pi/2 + nodo.th)
    cy    = nodo.y + radio*sin(pi/2 + nodo.th)
    x     = nodo.x - radio*cos(pi/2 + nodo.th + angDeGiro)
    y     = nodo.y - radio*sin(pi/2 + nodo.th + angDeGiro)
    th    = nodo.th + angDeGiro
    conf  = {x,y,th}
    retorna conf
```

Figura 3.15: Pseudocódigo de cálculo de la siguiente configuración del móvil [Elaboración propia].

d) Creación de nodos de configuración

Una vez conocidas las siguientes posibles configuraciones del móvil, que describen la posición y orientación del mismo, se debe descartar aquellas que colisionen con el entorno. Para ello, se debe hallar los vértices del móvil dada la configuración del mismo, luego obtener las posiciones discretas que describen la huella del móvil y comparar dicha huella con el entorno. Ver Figura 3.16 a.

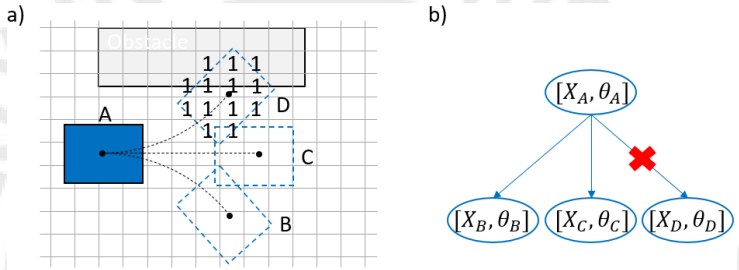


Figura 3.16: a) Representación de colisión de móvil con el entorno, b) Nodos de configuración resultantes de los posibles movimientos del móvil [Elaboración propia].

Cada configuración que se encuentre libre de colisión es almacenada en un nuevo nodo de configuración. Donde cada nodo contiene información de la orientación y posición del móvil, ver Figura 3.16 b. Luego estos nodos son asignados como los siguientes nodos de configuración del nodo que posee la configuración inicial. El pseudocódigo de dicho proceso se muestra en Figura 3.17.

```
funcion obtenSiguientesNodosCfgDe(nodo):
    crea lista SIGUIENTESNODOS vacía
    si nodo.dConf no se encuetra en cfgMap:
        d             = obtenDesplazamiento(nodo)
        angulosDeGiro = obtenPosiblesGiros(nodo)
        para cada angGiro en angulosDeGiro:
            conf      = calculaSiguienteConfiguracion(nodo,d,angGiro)
            vertices  = calculaVertices(x,y,th)
            huella    = calculaHuella(vertices)
            estaLibre = verificaSiEstaLibreDeColisiones(huella)
            si estaLibre es verdadero:
                dConf = discretizaConfiguracion(conf)
                nuevoNodoCfg = creaNodoCgf(conf)
                nuevoNodoCfg.dConf = dConf
                agrega nuevoNodoCfg a SIGUIENTESNODOS
                agrega dConf a cfgMap
    retorna SIGUIENTESNODOS
```

Figura 3.17: Pseudocódigo de obtención de siguientes nodos de configuración [Elaboración propia].

e) Algoritmo Hybrid A star

Una vez obtenidos la distancia holonómica, la distancia no holonómica y el cálculo de las siguientes configuraciones del móvil, se procede a implementar el algoritmo Hybrid A star y el planeador local.

Para empezar, se asigna un valor de costo total “f” a cada nodo, resultante de sumar el costo “g”, el cual es igual al desplazamiento que tomó en llegar al nodo de configuración, más el costo “h” igual al desplazamiento que se debería recorrer desde la configuración del nodo a la configuración final.

$$f = g + h \qquad (131)$$

Sabiéndose que, en espacios libres de obstáculos, el desplazamiento a recorrer para llegar a la configuración final es igual la distancia no holonómica y que en espacios con obstáculos, el desplazamiento para llegar a la meta es cercano a la distancia holonómica, se define a “h” como el máximo entre ambas distancias.

$$h = \max(h_{holonomico}, h_{no\ holonómico}) \qquad (132)$$

Mientras que el costo “g” del nodo actual se puede calcular como la suma del costo “g” del nodo anterior más el costo que toma llegar de la configuración anterior a la configuración actual.

Luego, teniendo cada nodo un costo, bastará con aplicar el algoritmo A estrella partiendo del nodo origen, que describe a la configuración inicial, para hallar el camino de nodos de menor costo que lleva desde la configuración inicial a la final; es decir, el camino de menor desplazamiento. El pseudocódigo del algoritmo Hybrid A star se muestra en Figura 3.18.

```
1  funcion hybridAStar(nodoInicial, confObjetivo):
2      crea lista Q vacía
3      agrega nodoIncial a Q
4      mientras Q no esté vacía:
5          nodoDeMenorCosto = obtenNodoDeMenorCostoDe(Q)
6          remueve nodoDeMenorCosto de Q
7          si configuración de nodoDeMenorCosto es similar a confObjetivo:
8              salir del bucle;
9          siguientesNodosCfg = obtenSiguientesNodosCfgDe(nodoDeMenorCosto)
10         para cada nodoCfg en siguientesNodosCfg:
11             costoGtentativo = g[nodoCfg] + costoDeIrDe(nodoDeMenorCosto, nodoCfg)
12             si g[nodoCfg] > costoGtentativo:
13                 g[nodoCfg] = costoGtentativo
14                 currentConf = {nodoCfg.x, nodoCfg.y, nodoCfg.th}
15                 f[nodoCfg] = g[nodoCfg] + max(h_holonomico[nodoCfg.y][nodoCfg.x],
16                                               h_noholonomico(currentConf,confObjetivo))
17                 rastreaNodoPadre[nodoCfg] = nodoDeMenorCosto
18             agrega nodoCfg a Q
19     crea lista SOLUCION vacía
20     reconstruyeSolucionCfg(nodoDeMenorCosto,rastreaNodoPadre,SOLUCION)
21     retorna SOLUCION
```

Figura 3.18: Pseudocódigo de algoritmo Hybrid A star [Elaboración propia].

Donde la reconstrucción de la secuencia de nodos que representa el camino a seguir, se obtiene con el algoritmo recursivo de Figura 3.19, el cual rastrea al nodo padre del último nodo evaluado.

```
funcion reconstruyeSolucionCfg(nodoFinal,rastreaNodoPadre,SOLUCION):
    agrega nodoFinal a SOLUCION;
    si rastreaNodoPadre[nodoFinal] no está vacío:
        nodoPadre = rastreaNodoPadre[nodoFinal]
        reconstruyeSolucionCfg(nodoPadre,rastreaNodoPadre,SOLUCION)
    retorna
```

Figura 3.19: Pseudocódigo de algoritmo de reconstrucción de solución [Elaboración propia].

Finalmente, el planeador local para un agente cualquiera (un robot) quedaría definido como se muestra en Figura 3.20.

```
funcion planeamientoLocal(agente):
    confFinal       = obtenPosicionYOrientacionFinal(agente)
    nodoCeldaFinal  = creaNodoCelda(confFinal)
    h_holonomico    = creaMatrizDeDistanciaHolonomica(nodoCeldaFinal)
    confInicial     = obtenPosicionYOrientacionInicial(agente)
    nodoCfgInicial  = creaNodoCgf(confInicial)
    agente.solucion = hybridAstar(nodoCfgInicial, confFinal)
    agente.camino   = obtenCaminoDeSolucion(agente.solucion)
    retorna
```

Figura 3.20: Pseudocódigo de planeador local [Elaboración propia].

3.3 Planeamiento en el Tiempo

El algoritmo a ser utilizado en el planeamiento global (sección 3.5), requiere que el planeador local permita al móvil esperar en el tiempo en una posición fija del mapa. Una forma de lograr esto es agregando, adicionalmente a los nodos de configuración del planeador local, un nodo de espera en el tiempo que resulta en la misma configuración inicial, ver Figura 3.21.

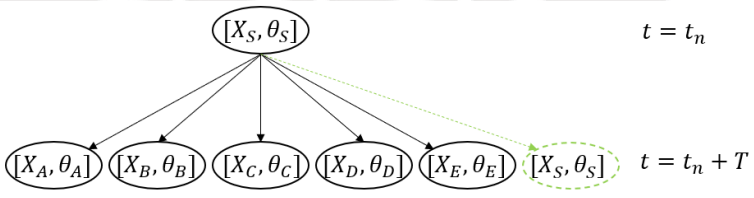


Figura 3.21: Adición de nodo de espera [Elaboración propia].

Sin embargo, esto implica la creación de nodos excesivos, siendo incluso infinitos, si el móvil se queda esperando de manera indefinida en su posición. Por ello, en vez de utilizar nodos de espera, se opta por usar nodos SIPP, los cuales son utilizados en el algoritmo SIPP (Safe Interval Path Planning) [43][47], para la evasión de obstáculos dinámicos. La razón es que estos describen el movimiento del móvil en rangos de tiempo en vez de instantes, evitándose así que se genere nodos de espera para cada instante.

3.3.1 Planeamiento SIPP

La idea del planeamiento SIPP es que para cada posición que puede ocupar el robot, se determine el intervalo de tiempo en el que este puede ocupar dicha posición de manera segura (sin colisionar con ningún obstáculo móvil) y luego determinar el instante de tiempo más temprano posible, dentro del intervalo seguro, en el que se puede llegar a dicha posición. Luego si cada nodo SIPP contiene información de la posición y tiempo de llegada a dicha posición (que además es libre de colisiones), bastaría con encontrar la secuencia de nodos SIPP que lleva al móvil a su destino en el menor tiempo posible.

Para explicar la creación de nodos SIPP, se usará el ejemplo de Figura 3.22 a, donde se muestra a un robot R ubicado en la casilla A y a dos obstáculos móviles Ob1 y Ob2, descritos en el instante 0. Donde el robot y los obstáculos se mueven a una velocidad de una celda por segundo. De la figura, se observa que la celda B sería ocupada por el obstáculo Ob1 en el instante 6 y por ende estaría libre entre los rangos [0, 5] y [7, ∞). En el algoritmo SIPP estos rangos son llamados intervalos seguros y se genera un nodo SIPP por cada intervalo seguro, ver Figura 3.22 b.

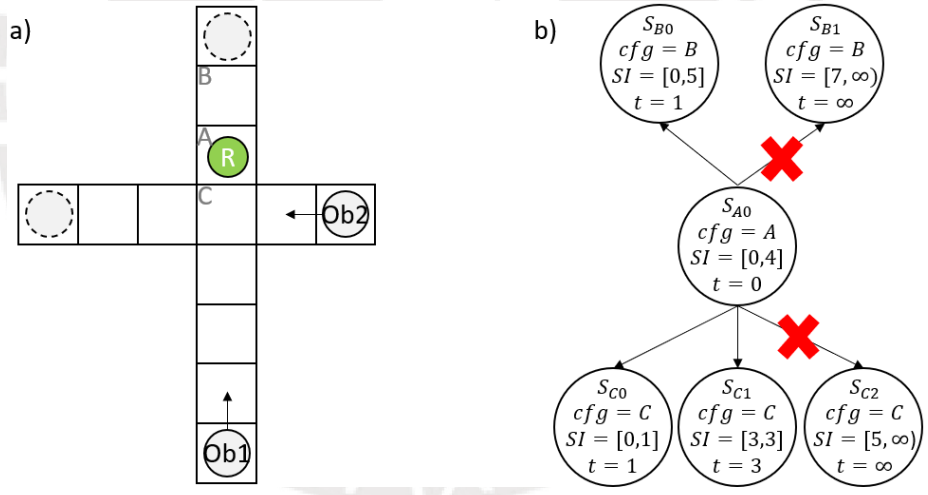


Figura 3.22: a) Problema de robot holonómico con dos obstáculos móviles, b) Representación de creación de nodos SIPP [Elaboración propia].

Para el nodo SB0 que representa al primer intervalo seguro de la celda B, el instante de tiempo "t" más temprano en el que el robot puede llegar a dicha celda en el rango [0 5], es 1, pues toma un segundo moverse de la celda A a la B. Mientras que para el nodo SB1, el instante más temprano en el rango [7, ∞) en el que el móvil puede llegar a la celda B es ∞, ya que el móvil nunca llegará a la celda B en ese intervalo. Pues si el rango seguro de SA0 va de 0 a 4 y el desplazamiento a B toma un segundo, el instante más tardío en el que el móvil pude llegar a B de

manera segura es 5, lo cual no es suficiente para entrar al rango seguro de B que empieza en el instante 7. Por otro lado, la celda C se encuentra ocupada en el instante 2 por el obstáculo Ob2 y en el instante 4 por el obstáculo Ob1 y, por ende, libre en los rangos de tiempo [0,1], [3,3] y [5, ∞), representados por los nodos SC0, SC1 y SC2 respectivamente. Dado que toma un segundo al robot trasladarse a la celda C, el tiempo más temprano en el que el móvil puede ocupar dicha celda para el primer rango sería 1; mientras que para el segundo rango sería 3, ya que solo en el instante 3 se encuentra libre la celda, y para el tercer rango, el tiempo más temprano sería ∞; ya que si bien el instante límite de llegada a C es 5 y este se encuentra dentro del rango seguro [5, ∞) de SC2, los movimientos de C a A del obstáculo y de A a C de móvil ocurren ambos en el mismo instante 5, lo cual no es un movimiento válido, pues implica que estos intercambian de posición.

El proceso explicado en el párrafo anterior es el realizado por el algoritmo que genera nuevos nodos SIPP, a partir de un nodo SIPP inicial. El pseudocódigo de dicho algoritmo se muestra en Figura 3.23.

```
1  getSuccessors(s)
2    successors = ∅;
3    for each m in M(s)
4      cfg =configuration of m applied to s
5      m_time =time to execute m
6      start_t = time(s) + m_time
7      end_t = endTime(interval(s)) + m_time
8      for each safe interval i in cfg
9        if startTime(i) > end_t or endTime(i) < start_t
10           continue
11       t =earliest arrival time at cfg during interval i with no collisions
12       if t does not exist
13         continue
14       s' = state of configuration cfg with interval i and time t
15       insert s' into successors
16   return successors;
```

Figura 3.23: Pseudocódigo de obtención de siguientes nodos SIPP [43].

La búsqueda del camino de nodos SIPP, se realiza agregándole un costo a cada nodo y luego utilizando el algoritmo de A estrella, para encontrar la secuencia de nodos de menor costo. Dado que cada nodo SIPP está asociado a una posición en el tiempo, el costo total de cada nodo es el costo “f” en tiempo, resultante de sumar el costo “g” del tiempo que tomó llegar al nodo SIPP actual más el costo “h” del tiempo que tomaría llegar a la meta desde el nodo SIPP actual. Donde “g” es igual al tiempo “t” más temprano de llegar al nodo SIPP actual. El pseudocódigo del algoritmo que realiza dicha búsqueda se muestra en Figura 3.24.

```
1  g(s_start) = 0; OPEN = ∅;
2  insert s_start into OPEN with f(s_start) = h(s_start);
3  while(s_goal is not expanded)
4    remove s with the smallest f-value from OPEN;
5    successors = getSuccessors(s);
6    for each s' in successors
7      if s' was not visited before then
8        f(s') = g(s') = ∞;
9      if g(s') > g(s) + c(s, s')
10       g(s') = g(s) + c(s, s');
11       updateTime(s');
12       f(s') = g(s') + h(s');
13       insert s' into OPEN with f(s');
```

Figura 3.24: Pseudocódigo de algoritmo SIPP [43].

3.3.2 Diseño del Planeador Local en el Tiempo

a) Obtención de intervalos seguros

Dado que, en la presente tesis, los móviles ocupan varias posiciones de una grilla en simultáneo y no una celda en específico, en vez de evaluar si una celda es segura, se evalúa si el conjunto de posiciones sobre el cual se encuentra el móvil es seguro. Para ello, se debe contar con una matriz que tenga almacenado el valor del rango de tiempo de ocupación de cada punto del mapa, con información del agente u obstáculo móvil que ocupa cada celda del mapa y el tiempo que ocupa la misma, ver Figura 3.25.

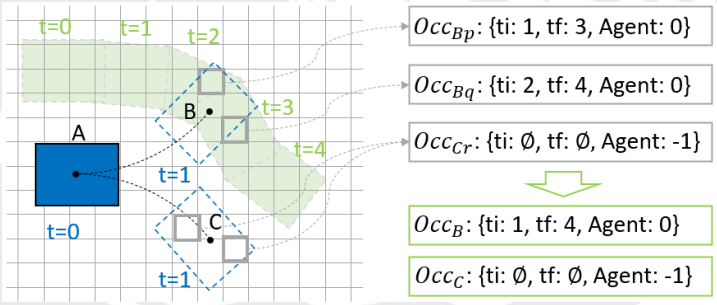


Figura 3.25: Representación de matriz de tiempo de ocupación por punto y de obtención de rango de ocupación de configuración [Elaboración propia].

Y luego usando dicha matriz se obtenga primero el rango de tiempo de ocupación de obstáculos sobre dicho conjunto de puntos, evaluando individualmente el rango de ocupación de cada punto y luego obteniendo los rangos de ocupación del conjunto de puntos, ver Figura 3.26.

```
1  funcion obtenRangosDeTiempoOcupados(huella):
2      para cada punto en huella:
3          rangoOcupado = mapaDeOcupacion[punto]
4          agrega rangoOcupado a rangosOcupados
5      rangosAgrupados = agrupaRangos(rangosOcupados)
6      retorna rangosAgrupados
```

Figura 3.26: Pseudocódigo de obtención de rango de ocupación de configuración [Elaboración propia].

Luego el intervalo seguro de la configuración del móvil se obtiene del complemento del tiempo de ocupación de la configuración. El pseudocódigo del algoritmo que halla el intervalo seguro de una configuración se muestra en Figura 3.27.

```
1  funcion obtenIntervalosSeguros(nodoCfg):
2      crea lista INTERVALOSSEGUROS
3      huella   = obtenHuella(nodoCfg)
4      rangosOc = obtenRangosDeTiempoOcupados(huella)
5      para cada rango en rangosOc:
6          si rango es el último elemento en rangosOc:
7              intervaloSeguro = {rango.fin,infinito}
8          otros:
9              intervaloSeguro = {rango.fin, siguienteRango.inicio}
10         agrega intervaloSeguro a INTERVALOSSEGUROS
11     retorna INTERVALOSSEGUROS
```

Figura 3.27: Pseudocódigo de obtención de intervalo seguro de configuración [Elaboración propia].

b) Verificación de movimiento válido

El algoritmo SIPP requiere que se descarte crear Nodos SIPP para configuraciones que resultan en movimientos inválidos. Esta verificación no es necesaria para el tipo de implementación realizada ya que el desplazamiento del móvil es menor a la longitud del mismo y, por ende, sea cual sea el movimiento que realice siempre se detecta la colisión comparando las huellas de los móviles.

c) Creación de nodos SIPP

El primer nodo SIPP que se crea, es el nodo que representa al intervalo seguro de la configuración inicial del móvil. Ver pseudocódigo de Figura 3.28.

```
1  funcion creaNodoSIPPInicial(nodoCfg):
2      nodoSIPP = creaNodoSIPP()
3      nodoSIPP.cfg = nodoCfg
4      nodoSIPP.tiempoDeLlegada = 0
5      nodoSIPP.intervaloSeguro = obtenIntervalosSeguros(nodoCfg)[0]
6      retorna nodoSIPP
```

Figura 3.28: Pseudocódigo de creación de nodo SIPP inicial [Elaboración propia].

Luego, a partir de este, se determina los siguientes nodos SIPP. Para ello, se debe determinar primero, las siguientes posibles configuraciones, de la configuración del nodo inicial, los cuales se puede obtener con la ejecución del algoritmo previamente implementado en la sección 3.2 para obtención de siguientes nodos de configuración. Una vez, obtenidos los nodos de configuración siguientes y el nodo SIPP inicial, se procede con el algoritmo de obtención de siguientes nodos SIPP tal como se explicó en 3.3.1, con la diferencia que los intervalos seguros se calculan de la forma explicada en a).

El pseudocódigo del algoritmo que obtiene los siguientes nodos SIPP, se muestra en Figura 3.29.

```
funcion obtenSucesores(nodoSIPP, siguientesNodosCfg):
    crea lista SUCESORES vacía
    t_transicion = miDesplazamiento / miVelocidad
    t_inicio     = nodoSIPP.tiempoDeLlegada + t_transicion
    t_fin        = nodoSIPP.IntervaloSeguro.fin + t_transicion
    para cada nodoCfg en siguientesNodosCfg:
        intervalosSeguros = obtenIntervalosSeguros(nodoCfg)
        para cada intervaloSeguro en intervalosSeguros:
            si intervaloSeguro.inicio > t_fin o intervaloSeguro.fin < t_inicio:
                continua
            t = max(t_inicio, nodoSIPP.intervaloSeguro.inicio)
            esFeasible  = verificaFeasibilidad(t,nodoCfg,intervaloSeguro.fin)
            si esFeasible es falso:
                continua
            nuevoNodoSIPP = creaNodoSIPP()
            nuevoNodoSIPP.tiempoDeLlegada = t
            nuevoNodoSIPP.intervaloSeguro.inicio = intervaloSeguro.inicio
            nuevoNodoSIPP.intervaloSeguro.fin = intervaloSeguro.fin
            agrega nuevoNodoSIPP a SUCESORES
    retorna SUCESORES
```

Figura 3.29: Pseudocódigo de algoritmo de obtención de siguientes nodos SIPP [Elaboración propia].

d) Algoritmo Hybrid A Star con planeamiento SIPP

El algoritmo Hybrid A Star con Planeamiento SIPP asigna costos a los nodos SIPP asociados al tiempo de desplazamiento del móvil. El costo total “f” de cada nodo, resulta de sumar el costo “g”, el cual es igual al tiempo que tomó en llegar al nodo SIPP (es decir, el tiempo más temprano de llegada al nodo), más el costo “h”, el cual es igual al tiempo que tomaría al móvil recorrer un camino desde la configuración del nodo a la configuración final.

$$f = g + h \quad (133)$$

El costo h del nodo queda definido por la división de la distancia heurística, obtenida del máximo de las distancias holonómica y no holonómica, con la velocidad del móvil.

$$h = \frac{\max(h_{holonomico}, h_{no\ holonómico})}{velocidad} \quad (134)$$

Mientras que el costo “g” del nodo actual se calcula como la suma del costo “g” del nodo SIPP anterior más el tiempo que toma llegar de dicho nodo a al nodo SIPP actual.

Luego, teniendo cada nodo SIPP un costo, bastará con aplicar el algoritmo A estrella en el nodo SIPP inicial, para hallar el camino de nodos de menor costo que lleva desde la configuración inicial a la final en el menor tiempo posible. El pseudocódigo del algoritmo Hybrid A Star con planeamiento SIPP se muestra en Figura 3.30.

```
funcion hybridAStarConSIPP(nodoInicio, confFinal):
    crea lista Q vacía
    agrega nodoInicio a Q
    mientras Q no esté vacía:
        mejorNodoSIPP = obtenNodoDeMenorCostoDe(Q)
        remueve mejorNodoSIPP de Q

        si configuración de mejorNodoSIPP.nodoCfg es similar a confFinal:
            salir del bucle;

        siguienteNodoCfg = obtenSiguientesNodosCfgDe(mejorNodoSIPP.nodoCfg)
        siguientesNodosSIPP = obtenSucesores(mejorNodoSIPP, siguienteNodoCfg)
        para cada nodoSIPP en siguientesNodosSIPP:
            tiempoDeTransision = nodoSIPP.tiempoDeLlegada - mejorNodoSIPP.tiempoDeLlegada
            costoGtentativo = g[mejorNodoSIPP] + tiempoDeTransision
            si g[nodoSIPP] > costoGtentativo:
                g[nodoSIPP] = costoGtentativo
                confActual  = {nodoSIPP.nodoCfg.x, nodoSIPP.nodoCfg.y,
                               nodoSIPP.nodoCfg.th}
                f[nodoSIPP] = g[nodoSIPP] +
                              max(h_holonomico[nodoSIPP.nodoCfg.y][nodoSIPP.nodoCfg.x],
                                  h_noholonomico(confActual, confFinal))/miVelocidad
                rastreaNodoPadre[nodoSIPP] = mejorNodoSIPP
            agrega nodoSIPP a Q

    crea lista SOLUCION
    reconstruyeSolucionSIPP(mejorNodoSIPP,rastreaNodoPadre,SOLUCION)
    retorna SOLUCION
```

Figura 3.30: Pseudocódigo de algoritmo Hybrid A star con planeamiento SIPP [Elaboración propia].

Donde la reconstrucción de la secuencia de nodos que representa el camino a seguir en espacio y tiempo, se obtiene con el algoritmo recursivo de Figura 3.31, el cual rastrea al nodo padre del último nodo evaluado.

```
funcion reconstruyeSolucionSIPP(nodoFinal,rastreaNodoPadre,SOLUCION):
    agrega nodoFinal a SOLUCION;
    si rastreaNodoPadre[nodoFinal] no está vacío:
        nodoPadre = rastreaNodoPadre[nodoFinal]
        reconstruyeSolucionSIPP(nodoPadre,rastreaNodoPadre,SOLUCION)
    retorna
```

Figura 3.31: Pseudocódigo de algoritmo de reconstrucción de solución [Elaboración propia].

Finalmente, se modifica al planeador local descrito en la sección 3.2, para incluir la lógica del algoritmo SIPP, resultando en un planeador en espacio tiempo. Ver Figura 3.32.

```
funcion planeamientoLocalEnElTiempo(agente):
    confFinal      = obtenPosicionYOrientacionFinal(agente)
    nodoCeldaFinal = creaNodoCelda(confFinal)
    h_holonomico   = creaMatrizDeDistanciaHolonomica(nodoCeldaFinal)

    confInicial        = obtenPosicionYOrientacionInicial(agente)
    nodoSIPPInicial    = creaNodoSIPPInicial(confInicial)
    agente.solucion    = hybridAstarConSIPP(nodoSIPPInicial, confFinal)
    agente.Trayectoria = reconstruirTrayectoria(agente.solucion)
    retorna
```

Figura 3.32: Pseudocódigo de algoritmo del planeador local en el tiempo [Elaboración propia].

3.4 Planeamiento Global

Al haber múltiples vehículos en el mapa, es necesario el uso de un planeador global que solucione los conflictos entre los robots, para evitar que estos colisionen o se queden atrapados en regiones del mapa.

3.4.1 Algoritmo CBS

El algoritmo de mejor performance para planeamiento de sistemas multi agentes es el de CBS (búsqueda en base a conflictos), el cual consiste en generar un árbol de restricciones para solucionar los conflictos entre múltiples agentes (entidades sin propiedades físicas), ver sección 1.2.2c).

En general la idea del algoritmo, es que cada agente planee su camino a seguir de manera independiente (usando cualquier planeador local) sin considerar a los otros agentes y luego, se compare los caminos en cada instante de tiempo para verificar si hay colisiones entre los agentes. De existir colisiones entre estos, digamos entre un agente A y uno B, se crean dos posibles realidades, representadas cada una por un nodo CBS, ver Figura 3.33, una realidad en donde se restringe solo al agente A de no estar en la posición de la colisión en el momento que ocurre (para que así no choque con el agente B) y otra en donde se restringe solo al agente B de no estar en la posición de colisión en el momento que ocurre (para que así no choque con el agente A). Luego se resuelven ambas realidades; es decir, ambos nodos CBS, usando el planeador local de cada agente, tal que sus soluciones cumplan con las restricciones impuestas.

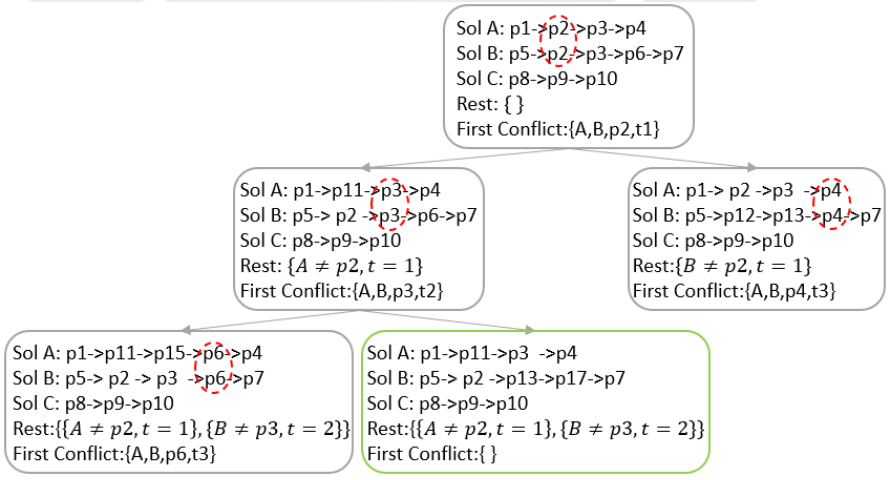


Figura 3.33: Árbol de restricciones formado con nodos CBS [Elaboración propia].

Enseguida, se procede a aplicar un algoritmo de búsqueda, para encontrar al nodo CBS de menor costo, agregando los nodos resueltos a una lista de posibles

soluciones y tomando al mejor nodo de esta lista. Si el nodo elegido presenta alguna colisión, se repite el proceso, generando más nodos CBS, los cuales heredan las restricciones de su nodo padre, creándose así un árbol de restricciones. Al algoritmo de búsqueda con imposición de restricciones se le llama algoritmo de alto nivel y al planeador local de cada agente se le llama algoritmo de bajo nivel.

3.4.2 Diseño del Planeador Global

a) Imposición de restricciones

En la presente tesis, se modifica al algoritmo CBS, para que pueda lidiar con robots no holonómicos, cuyas huellas ocupan en un instante más de una celda en el mapa tipo grilla. Para ello, en vez de restringir la posición y orientación del robot a no estar en una configuración específica, se restringe al robot que no colisione con la huella del otro robot (todos los puntos que el otro robot ocupa sobre la grilla) en el instante de la colisión. Ver Figura 3.34 y Figura 3.35.

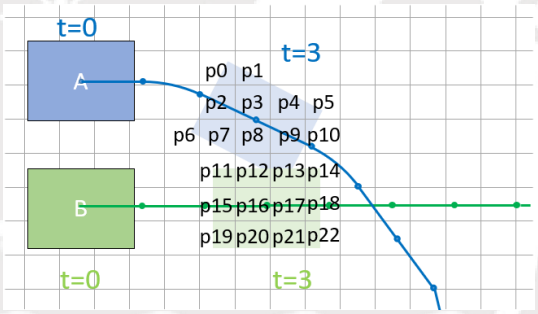


Figura 3.34: Colisión de móviles en instante t=5 [Elaboración propia].

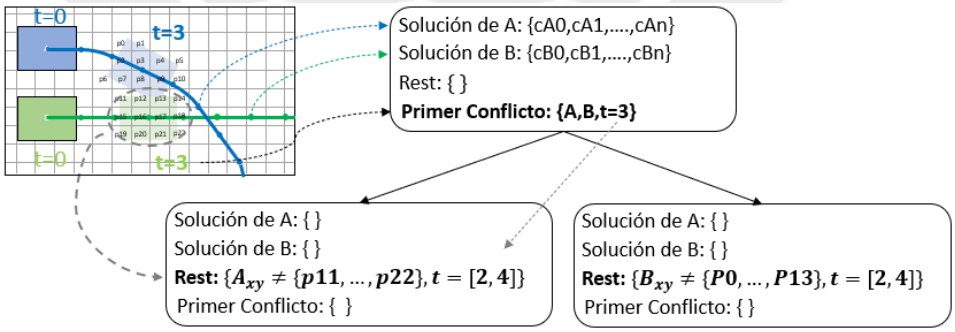


Figura 3.35: Árbol de restricciones modificado [Elaboración propia].

Las restricciones son almacenadas en un mapa de tiempos de ocupación por punto, el cual es usado posteriormente por el planeador local, ver sección 3.3.2a). Donde cada entrada del mapa corresponde a una celda (ver Figura 3.25) y contiene un vector que indica el tiempo de inicial y final de ocupación de dicha celda, y qué agente ocupó dicha celda.

Si un móvil A y uno B colisionan en un tiempo t, las restricciones a añadir al móvil A dependen del tiempo de ocupación del móvil B en un conjunto de celdas. El tiempo inicial de ocupación es calculado como el tiempo de colisión t menos el tiempo de desplazamiento del móvil para entrar en dicho conjunto de celdas. Mientras que el tiempo final de ocupación es el tiempo de colisión más el tiempo de desplazamiento para salir de dicho conjunto de celdas. El pseudocódigo del algoritmo que añade las restricciones se muestra en Figura 3.36.

```
funcion llenarMapadeOcupacionDeAgenteA(instanteDeColision,celdasRestringidas,agenteA,agenteB):
    tiempoDeDesplazamientoDeB = agenteB.desplazamiento/agenteB.velocidad
    ti = instanteDeColision - tiempoDeDesplazamientoDeB
    tf = instanteDeColision + tiempoDeDesplazamientoDeB
    para cada celda en celdasRestringidas:
        ocupacionB = {ti,tf,agenteB}
        agenteA.mapaDeOcupacion[celda] = ocupacionB
    retorna
```

Figura 3.36: Pseudocódigo de inserción de restricciones [Elaboración propia].

b) Búsqueda de primer conflicto

La implementación del algoritmo CBS requiere que se encuentre la primera colisión que ocurre entre las trayectorias de dos móviles. Para ello, se debe comparar la posición de cada uno de los móviles en el tiempo partiendo desde el instante cero. Sin embargo, debido al uso de nodos SIPP en planeador local, los tiempos de ambas trayectorias no coinciden, lo cual impide comparar las posiciones de una trayectoria con la otra de manera directa. Por ello, se debe crear un vector de tiempos base que permita la comparación de posiciones. Este vector es creado almacenando todos los instantes tiempos, no repetidos, de ambas trayectorias, y ordenándolo de forma ascendente, tal como se muestra en el pseudocódigo del algoritmo de búsqueda del primer conflicto en Figura 3.37.

```
funcion hallaPrimerConflicto(nodoCBS):
    crea set tiempos vacío
    para cada agente en nodoCBS.agentes:
        trayect = obtenTrayectoria(agente)
        agrega trayect.tiempo a tiempos
    ordena tiempos en forma ascendente
    para cada instante en tiempos:
        limpia histMap
        para cada agente en nodoCBS.agentes:
            huella = obtenHuellaEn(instante,agente)
            para cada celda en huella:
                si histMap[celda] no está vacio:
                    histMap[celda].agente = agente
                    histMap[celda].instante = instante
                otros:
                    nodoCBS.primerConflicto.agenteA  = histMap[celda].agente
                    nodoCBS.primerConflicto.agenteB  = agente
                    nodoCBS.primerConflicto.instante = histMap[celda].instante
                    retorna
    retorna
```

Figura 3.37: Pseudocódigo de búsqueda del primer conflicto [Elaboración propia].

Una vez obtenido este vector de tiempos, se procede a almacenar todos los puntos de la huella de cada móvil en un mapa que relaciona cada posición con el par tiempo-agente. Cuando alguna posición del mapa es visitada más de una vez, se verifica que hay una colisión, y se procede a guardar a los agentes implicados en la misma, así como el tiempo de colisión y se cesa la ejecución del algoritmo.

c) Creación de nodo CBS

Para aplicar el algoritmo CBS, se debe contar con el nodo CBS inicial. El cual es creado como un nodo sin restricciones que contiene solo a los agentes (los móviles), y usando el algoritmo de Figura 3.38, se obtiene los caminos iniciales de cada uno de los agentes, así como el primer conflicto encontrado entre ambas soluciones.

```
funcion resuelveNodoCBS(nodoCBS):
    para cada agente en nodoCBS.agentes:
        planeamientoLocalEnElTiempo(agente)

    costo = calculaCostoCBS(nodoCBS)
    si costo es distinto a infinito:
        nodoCBS.primerConflict = hallaPrimerConflicto(nodoCBS)
    retorna
```

Figura 3.38: Pseudocódigo de solucionador de nodo CBS [Elaboración propia].

Luego la creación de los dos nodos siguientes a este nodo o a cualquier otro nodo, procede como se muestra en el pseudocódigo de Figura 3.39.

```
funcion obtenSiguientesNodosCBSDe(nodoCBS):
    crea lista SIGUIENTESNODOS vacía
    conflicto         = nodoCBS.primerConflicto
    siguienteNodoCBS1 = creaNodoCBS(nodoCBS.agentes)
    siguienteNodoCBS2 = creaNodoCBS(nodoCBS.agentes)
    celdasA           = obtenHuellaDe(conflicto.agenteA)
    celdasB           = obtenCeldasOcupadas(conflicto.agenteB)
    agregaRestricciones(siguienteNodoCBS1, conflicto.agenteA.idx,
                        conflicto.agenteA.idx,celdasB,conflicto.instante)
    addConstraintTo(siguienteNodoCBS2, conflicto.agenteB.idx,
                    conflicto.agenteB.idx,celdasA,conflicto.instante)
    solvenodoCBS(siguienteNodoCBS1)
    solvenodoCBS(siguienteNodoCBS2)
    agrega siguienteNodoCBS1 a SIGUIENTESNODOS
    agrega siguienteNodoCBS2 a SIGUIENTESNODOS
    retorna SIGUIENTESNODOS
```

Figura 3.39: Pseudocódigo de obtención de siguientes nodos CBS [Elaboración propia].

En este, se crean dos nodos CBS vacíos, se obtiene la huella de cada agente en el momento de la colisión, y luego, se restringe en el primer nodo CBS a que el primer agente no ocupe ninguna posición de la huella del segundo agente en el momento de la colisión, mientras que en el segundo nodo CBS se restringe a que el segundo agente no ocupe ninguna posición de la huella del primer agente en el momento de la colisión, donde las restricciones son impuestas usando el pseudocódigo de

Figura 3.40. Teniendo las restricciones definidas, se utiliza el algoritmo de Figura 3.38 para encontrar los caminos de cada agente que respeten la restricción impuesta y encontrar el primer conflicto si es que existe alguno. Luego se retorna una lista que incluya a ambos nodos.

```
funcion agregaRestriccionA(nodoCBS, agenteAIdx, agenteBIdx, celdasB, instante):
    instanteDeColision = instante
    celdasRestringidas = celdasB
    agenteA            = nodoCBS.agentes[agenteAIdx]
    agenteB            = nodoCBS.agentes[agenteBIdx]
    llenarMapadeOcupacionDeAgenteA(instanteDeColision,celdasRestringidas,agenteA,agenteB)
    retorna
```

Figura 3.40: Pseudocódigo de obtención de siguientes nodos CBS [Elaboración propia].

Con el fin de hallar la solución de manera rápida, se plantea una función de costo del nodo CBS distinta al del algoritmo original. Sabiéndose que, para evitar la colisión, el móvil deberá tomar un camino ligeramente distinto al del nodo anterior, se prioriza aquellas soluciones donde se explore nuevas configuraciones y donde exista mayor número de pasos. Por otro lado, se debe penalizar aquellas soluciones que tomen caminos que demoren más en ser transitados. Y dado que existe múltiples agentes "i" se utiliza la sumatoria de los parámetros mencionados. Luego, el costo queda definido de la siguiente manera:

$$costoCBS = \frac{\sum_{i=0}^{N} costoDeCamino_i}{\sum_{i=0}^{N} n°pasosDeCamino_i * (1 + \sum_{i=0}^{N} n°nuevasConfiguraciones_i)} \quad (135)$$

El pseudocódigo del algoritmo que calcula dicho costo se muestra en Figura 3.41.

```
funcion calculaCostoCBS(nodoCBS):
    sumaDePasos      = 0
    sumaDeCostos     = 0
    numNuevosLugares = 0
    para cada agente en nodoCBS.agentes:
        numNuevosLugares += obtenNuevosLugaresExplorados(agente)
        sumaDePasos      += numeroDePasos(agente.trayectoria)
        sumaDeCostos     += agente.trayectoria.tiempo.back()

    nodoCBS.costo = sumaDeCostos / (sumaDePasos*(1+numNuevosLugares))
    retorna
```

Figura 3.41: Pseudocódigo de cálculo del costo de nodo CBS [Elaboración propia].

d) Algoritmo CBS

El algoritmo CBS desarrollado en esta tesis, posee una estructura distinta al original mostrado en Figura 1.12. En esta implementación, la búsqueda del primer conflicto y la ejecución del algoritmo de bajo nivel, para hallar los caminos de cada agente son implementados en la creación de los siguientes nodo CBS. De esta manera, se puede aplicar cualquier algoritmo de búsqueda en el algoritmo de alto nivel, para

encontrar al nodo CBS de menor costo. El pseudocódigo del algoritmo de alto nivel se muestra en Figura 3.42.

```
funcion CBSdeAltoNivel(nodoInicial):
    crea lista Q vacía
    agrega nodoInicial a Q
    mientras Q no esté vacía:
        mejorNodo = obtenNodoDeMenorCostoDe(Q)
        remueve mejorNodo de Q

        si no hay conflictos en mejorNodo:
            salir del bucle;

        siguientesNodosCBS = obtenSiguientesNodosCBSDe(mejorNodo)
        para cada nodo en siguientesNodosCBS:
            agrega nodo a Q

    retorna mejorNodo
```

Figura 3.42: Pseudocódigo del algoritmo de alto nivel [Elaboración propia].

Finalmente, el planeado global quería definido tal como se muestra en Figura 3.43.

```
funcion planeadorGlobal(infoDeVehiculos):
    creal lista TRAYECTORIAS
    agentes        = creaAgentes(infoDeVehiculos)
    nodoCBSInicial = creaNodoCBS(agentes)
    resuelveNodoCBS(nodoCBSInicial)
    nodoCBSSolucion = CBSdeAltoNivel(nodoCBSInicial)

    para cada agente en nodoCBSSolucion.agentes:
        agrega agente.trayectoria a TRAYECTORIAS

    retorna TRAYECTORIAS
```

Figura 3.43: Pseudocódigo del planeador global [Elaboración propia].

Capítulo 4: Generación de Trayectoria y Control

4.1 Generación de Trayectoria

El planeador global retorna ya de por sí las trayectorias que cada móvil debe seguir en espacio y tiempo para llegar a su destino. Sin embargo, estas al haber sido obtenidas mediante la exploración del mapa con giros discretos, no necesariamente posee las curvas más suaves o los caminos más cortos. Por ello, adicionalmente, se optimiza las posiciones de las trayectorias, con el fin de obtener el camino más corto.

Tomando como referencia la optimización realizada en [38], se optimiza directamente los puntos del camino, en vez de realizar la optimización de una parametrización por tramos del camino con curvas de Bézier u otros. Esto debido a que parametrizar el camino por tramos implicaría agregar más parámetros a la optimización, además de agregar restricciones de igualdad al inicio y final de cada tramo para que las curvas coincidan.

4.1.1 Propuesta de Optimización

Dado que las curvas de Dubins representan los caminos de menor longitud para el traslado de un móvil, se planea hacer uso de estas para la optimización. La idea es reducir la longitud del camino hasta que solo queden curvas de Dubins. Así mismo, se debe evitar la colisión con obstáculos del mapa, por ello, se debe considerar una distancia mínima entre el móvil y el obstáculo. Mientras que, para tener un camino uniforme se debe tener en cuenta el espaciamiento entre cada punto del mismo. Así mismo, se debe corregir la configuración final, obtenida por el planeador global, para que coincida con la configuración destino ideal.

En Figura 4.1, se muestra una representación de las fuerzas que deben mover a los puntos durante la optimización, donde R es el radio mínimo de giro, d_{min} la distancia mínima al obstáculo, F_l la fuerza que reduce el espacio entre puntos (y por ende la longitud del camino), F_c la fuerza que mantiene la curva de Dubins, F_o la fuerza que mueve al móvil a una distancia mínima con el obstáculo en caso de colisión, F_s la fuerza que mantiene la uniformidad del espacio entre los puntos y F_e la fuerza que mueve el punto final al punto destino ideal.

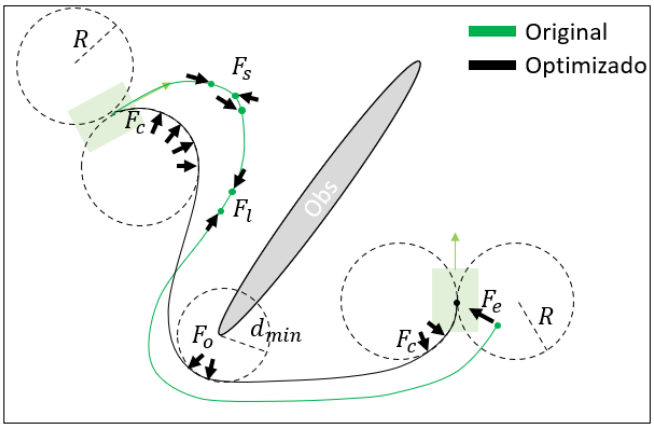


Figura 4.1: Representación de fuerzas durante la optimización [Elaboración propia].

Dado de que existe más de un móvil en el mapa, se debe optimizar la distancia entre los móviles para los mismos instantes de tiempo. Así el punto de la trayectoria de un móvil 1 en un instante t se verá afectada por una fuerza aplicada por el punto de la trayectoria móvil 2 en el mismo instante t, tal como se muestra en Figura 4.2, donde fd_{tn} representa la fuerza que aplica la trayectoria de un móvil 2 sobre los puntos de la trayectoria del móvil 1 en cada instante n y d_{car} la distancia mínima que deben mantener los dos móviles.

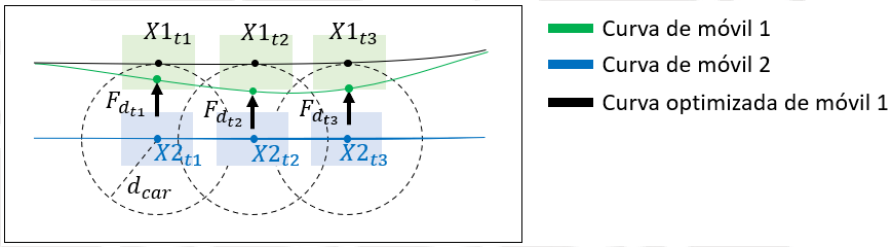


Figura 4.2: Representación de las fuerzas aplicadas por un móvil 2 a un móvil 1 [Elaboración propia].

Por otro lado, el punto de inicio del camino debe estar exento de la optimización, ya que es igual al punto de origen ideal. Y así mismo, dado que los móviles pueden quedarse esperando en puntos del mapa para evitar una colisión, se debe eximir también a dichos puntos de la optimización y agrégales un par de circunferencias de mínimo radio de giro a cada lado, para respetar la cinemática del móvil, ver Figura 4.3.

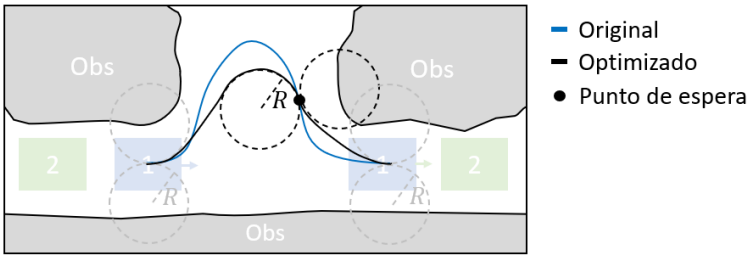


Figura 4.3: Adición de circunferencias en punto de espera [Elaboración propia].

4.1.2 Cálculo de Función de Costo

Dado que se desea minimizar la longitud camino, se plantea minimizar una función de costo f_l que describe la sumatoria de todas las distancias entre dos puntos consecutivos del camino. Donde cada punto es de denotado con el subíndice i y donde f_{l_i} es la distancia entre el punto i y el $i-1$.

$$X_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (136)$$

$$\Delta X_i = X_i - X_{i-1} \quad (137)$$

$$f_{l_i} = |\Delta X_i|^2 \quad (138)$$

$$f_l = \sum_{i=1}^{N} f_{l_i} = \sum_{i=1}^{N} |\Delta X_i|^2 \quad (139)$$

Por otro lado, para que el espaciamiento entre puntos sea uniforme, se propone minimizar la función de costo f_s que describe la diferencia entre dos distancias ΔX consecutivas en el camino.

$$f_{s_i} = |\Delta X_{i+1} - \Delta X_i|^2 \quad (140)$$

$$f_s = \sum_{i=1}^{N-1} f_{s_i} = \sum_{i=1}^{N-1} |\Delta X_{i+1} - \Delta X_i|^2 \quad (141)$$

Mientras que para evitar que el vehículo colisione con los obstáculos estáticos se plantea minimizar la función de costo f_o que describe cuanto se acerca la distancia entre el móvil y el obstáculo a la distancia mínima d_{min} que deben mantener entre ambos. Donde esta función actúa solo cuando los puntos X_i se encuentran dentro de la circunferencia de radio d_{min} centrada en O_i.

$$f_{o_i} = \left(\frac{1}{|X_i - O_i|} - \frac{1}{d_{min}}\right)^2 * (|X_i - O_i| < d_{min}) \quad (142)$$

$$f_o = \sum_{i=1}^{N} f_{o_i} \quad (143)$$

Así mismo, para respetar la restricción física del giro del móvil y aproximar el camino a las curvas de Dubins, se plantea minimizar la función de costo f_c que describe cuanto se acercan los puntos i del camino a dos circunferencias con radio r_{min} igual al radio mínimo de giro del móvil, las cuales se encuentran ubicadas al lado derecho Cr_j e izquierdo Cl_j de los puntos j inicial, final y de espera del móvil.

$$Cr_j = X_j + R * \begin{bmatrix} \cos(\theta_j - 0.5 * \pi) \\ \sin(\theta_j - 0.5 * \pi) \end{bmatrix} \quad (144)$$

$$Cl_j = X_j + R * \begin{bmatrix} \cos(\theta_j + 0.5 * \pi) \\ \sin(\theta_j + 0.5 * \pi) \end{bmatrix} \quad (145)$$

$$fcr_{i_j} = \left(\frac{1}{|X_i - Cr_j|} - \frac{1}{r_{min}} \right)^2 * (|X_i - Cr_j| < r_{min}) \quad (146)$$

$$fcl_{i_j} = \left(\frac{1}{|X_i - Cl_j|} - \frac{1}{r_{min}} \right)^2 * (|X_i - Cl_j| < r_{min}) \quad (147)$$

$$f_c = \sum_{j=1}^{M} \left(\sum_{i=1}^{N} fcr_{i_j} + \sum_{i=1}^{N} fcl_{i_j} \right) \quad (148)$$

Cabe precisar que para un móvil tipo bicicleta, el radio mínimo de giro r_{min} queda definido por su desplazamiento d y su ángulo máximo de giro del timón δ_{max} de la siguiente forma:

$$r_{min} = \frac{d}{\tan(\delta_{max})} \quad (149)$$

Al igual que la función de costo anterior, esta actúa solo cuando los puntos X_i se encuentran dentro de alguna de las circunferencias de Dubins, siendo su valor igual cero cuando se encuentra fuera de las circunferencias.

Mientras que para corregir la posición del punto final de manera gradual hasta ser igual al punto destino, se minimiza la función de costo f_e que describe la distancia entre la posición del punto final X_{end} y la posición del punto final deseado X_{goal}.

$$f_e = |X_{end} - X_{goal}|^2 \quad (150)$$

Mientras que, para evitar la colisión con todos los V móviles en todo instante t, se minimiza la función de costo f_d que describe cuanto se acerca durante todo el tiempo T que dura recorrer el camino, la distancia entre el móvil evaluado y el resto de V móviles a la distancia mínima d_{car} que debe mantener el móvil evaluado con cada uno de los móviles v.

$$f_{d_{t_v}} = \left(\frac{1}{|X_t - Xv_t|} - \frac{1}{d_{car}} \right)^2 * (|X_t - Xv_t| < d_{car}) \quad (151)$$

$$f_{d_t} = \sum_{v=1}^{V} f_{d_{t_v}} \quad (152)$$

$$f_d = \sum_{t=1}^{T} f_{d_t} \quad (153)$$

Notar que para esta función de costo se pueda aplicar, las trayectorias de todos los móviles deben tener definidas las posiciones en los mismos instantes de tiempo, lo cual se puede lograr por interpolación.

Luego, la función de costo total f para el móvil evaluado queda definida de la siguiente manera.

$$f = w_l * f_l + w_s * f_s + w_o * f_o + w_c * f_c + w_e * f_e + w_d * f_d \quad (154)$$

Donde w_l, w_s, w_o,w_c, w_e y w_d son pesos asociados a cada término de la función de costo total.

4.1.3 Cálculo de Gradientes y Hessianas

Para el término asociado a la reducción de longitud, las derivadas con respecto a cada punto del camino quedan definidos de la siguiente forma:

$$\frac{d(|\Delta X_i|)}{d\Delta X_i} = \frac{\Delta \mathrm{X_i}^T}{|\Delta X_i|} \quad (155)$$

$$\frac{df_{l_i}}{dX_i} = w_l * 2 * |\Delta X_i| * \frac{\Delta \mathrm{X_i}^T}{|\Delta X_i|} * \frac{d(\Delta \mathrm{X_i})}{dX_i} = w_l * 2 * \Delta X_i^T * I = 2 * w_l * \Delta X_i^T \quad (156)$$

$$\frac{df_{l_i}}{dX_{i-1}} = w_l * 2 * |\Delta X_i| * \frac{\Delta \mathrm{X_i}^T}{|\Delta X_i|} * \frac{d(\Delta \mathrm{X_i})}{dX_{i-1}} = w_l * 2 * \Delta X_i^T * -I = -2 * w_l * \Delta X_i^T \quad (157)$$

$$\frac{df_{l_i}}{dX_{i-k}} = 0, \qquad k > 1 \quad (158)$$

$$\frac{df_{l_i}}{dX_{i+k}} = 0, \qquad k > 0 \quad (159)$$

De la misma manera, para el termino de espaciamiento, se tiene lo siguiente:

$$\frac{df_{s_i}}{dX_{i-1}} = 2 * w_s * (\Delta X_{i+1} - \Delta X_i)^T \quad (160)$$

$$\frac{df_{s_i}}{dX_i} = -4 * w_s * (\Delta X_{i+1} - \Delta X_i)^T \quad (161)$$

$$\frac{df_{s_i}}{dX_{i+1}} = 2 * w_s * (\Delta X_{i+1} - \Delta X_i)^T \quad (162)$$

$$\frac{df_{s_i}}{dX_{i-k}} = 0, \qquad k > 1 \quad (163)$$

$$\frac{df_{s_i}}{dX_{i+k}} = 0, \qquad k > 1 \quad (164)$$

Mientras que la derivada del término asociado a la corrección de punto final es la siguiente:

$$\frac{df_e}{dX_{end}} = 2 * w_e * \left(X_{end} - X_{goal}\right)^{\mathrm{T}} \quad (165)$$

$$\frac{df_e}{dX_i} = 0, \qquad i \neq end \quad (166)$$

Así mismo, las derivadas asociadas al término de evasión de obstáculos son de la siguiente forma:

$$\frac{df_{o_i}}{dX_i} = w_o * -2 * \left(\frac{1}{|X_i - O_i|} - \frac{1}{d_{min}}\right) * \frac{(X_i - O_i)^T}{|X_i - O_i|^3} .* (|X_i - O_i| < d_{min}) \quad (167)$$

$$\frac{df_{o_i}}{dX_{i+k}} = 0, \qquad k \neq 0 \quad (168)$$

De la misma manera, la derivada asociada a la distancia entre dos móviles se define de la siguiente manera:

$$\frac{df_{dv_t}}{dX_t} = w_d * -2 * \left(\frac{1}{|X_t - Xv_t|} - \frac{1}{d_{car}}\right) * \frac{(X_t - Xv_t)^T}{|X_t - Xv_t|^3} .* (|X_t - Xv_t| < d_{car}) \quad (169)$$

$$\frac{df_{dv_t}}{dX_{t+k}} = 0, \qquad k \neq 0 \quad (170)$$

Mientras que, las derivadas del término asociado a las restricciones físicas de los móviles son las siguientes:

$$\frac{dfcr_{i_j}}{dX_i} = w_c * -2 * \left(\frac{1}{|X_i - Cr_j|} - \frac{1}{r_{min}}\right) * \frac{(X_i - Cr_j)^T}{|X_i - Cr_j|^3} .* (|X_i - Cr_j| < r_{min}) \quad (171)$$

$$\frac{dfcl_{i_j}}{dX_i} = w_c * -2 * \left(\frac{1}{|X_i - Cl_j|} - \frac{1}{r_{min}}\right) * \frac{(X_i - Cl_j)^T}{|X_i - Cl_j|^3} .* (|X_i - Cl_j| < r_{min}) \quad (172)$$

$$\frac{dfc_{i_j}}{dX_i} = \frac{dfcr_{i_j}}{dX_i} + \frac{dfcl_{i_j}}{dX_i} \quad (173)$$

$$\frac{dfc_{i_j}}{dX_{i+k}} = 0, \qquad k \neq 0 \quad (174)$$

Luego, si cada punto i es un punto muestreado con tiempos uniformes, tal que $i = t$, se tiene que, para cada punto, las derivadas se definen de la siguiente manera:

$$\frac{df_i}{dX_{i-1}} = \frac{df_{l_i}}{dX_{i-1}} + \frac{df_{s_i}}{dX_{i-1}} \quad (175)$$

$$\frac{df_i}{dX_i} = \frac{df_{l_i}}{dX_i} + \frac{df_{s_i}}{dX_i} + \frac{df_{o_i}}{dX_i} + \frac{df_{d_i}}{dX_i} + \sum_{j=1}^{M} \frac{dfc_{i_j}}{dX_i} \quad (176)$$

$$\frac{df_i}{dX_{i+1}} = \frac{df_{s_i}}{dX_{i+1}} \quad (177)$$

$$\frac{df_{end}}{dX_{end}} = \frac{df_e}{dX_{end}} \quad (178)$$

$$\frac{df_i}{dX_{i-k}} = 0, \qquad k > 1, \qquad i - k \neq end \quad (179)$$

$$\frac{df_i}{dX_{i+k}} = 0, \qquad k > 1, \qquad i + k \neq end \quad (180)$$

Luego, la matriz Hessiana de f_{l_i} para dos puntos consecutivos es:

$$H_{l_i} = \begin{bmatrix} \frac{d^2 f_{l_i}}{d^2 X_{i-1}} & \frac{d^2 f_{l_i}}{dX_{i-1} dX_i} \\ \frac{d^2 f_{l_i}}{dX_i dX_{i-1}} & \frac{d^2 f_{l_i}}{d^2 X_i} \end{bmatrix} = \begin{bmatrix} 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \end{bmatrix} \quad (181)$$

$$det(H_{l_i}) = 0 \quad (182)$$

Con eigenvalores iguales a 0,0,4 y 4, por ende, la matriz Hessiana es positiva semi definida y la función de costo f_{l_i} es convexa y en consecuencia f_l también. Por otro lado, como el determinante es igual a cero, la matriz hessiana es no invertible.

Mientras que, la Hessiana de f_{s_i} para tres puntos consecutivos es:

$$H_{s_i} = \begin{bmatrix} \frac{d^2 f_{s_i}}{d^2 X_{i-1}} & \frac{d^2 f_{s_i}}{dX_{i-1} dX_i} & \frac{d^2 f_{s_i}}{dX_{i-1} dX_{i+1}} \\ \frac{d^2 f_{s_i}}{dX_i dX_{i-1}} & \frac{d^2 f_{s_i}}{d^2 X_i} & \frac{d^2 f_{s_i}}{dX_i dX_{i+1}} \\ \frac{d^2 f_{s_i}}{dX_{i+1} dX_{i-1}} & \frac{d^2 f_{s_i}}{dX_{i+1} dX_i} & \frac{d^2 f_{s_i}}{d^2 X_{i+1}} \end{bmatrix} = \begin{bmatrix} 2 & 0 & -4 & 0 & 2 & 0 \\ 0 & 2 & 0 & -4 & 0 & 2 \\ -4 & 0 & 8 & 0 & -4 & 0 \\ 0 & -4 & 0 & 8 & 0 & -4 \\ 2 & 0 & -4 & 0 & 2 & 0 \\ 0 & 2 & 0 & -4 & 0 & 2 \end{bmatrix} \quad (183)$$

$$det(H_{s_i}) = 0 \quad (184)$$

Con eigenvalores iguales a 0,0,0,0,12 y 12, por ende, la matriz Hessiana es positiva semi definida y la función de costo f_{s_i} es convexa y en consecuencia f_s también. Por otro lado, como el determinante es igual a cero, la matriz hessiana es no invertible.

Mientras que, para un punto $X_i = [x_i, y_i]$ la Hessiana H_{o_i} del término f_{o_i}, que describe al distanciamiento con un obstáculo, es de la siguiente forma:

$$\nabla_{o_i} = \left(\frac{df_{o_i}}{dX_i}\right)^T = w_o * -2 * \left(\frac{1}{|X_i - O_i|} - \frac{1}{d_{min}}\right) * \frac{X_i - O_i}{|X_i - O_i|^3} \quad (185)$$

$$H_{o_i} = J(\nabla_{o_i}) = w_o * -2 * \left(\left(\left(\frac{1}{|X_i - O_i|} - \frac{1}{d_{min}}\right) * \frac{1}{|X_i - O_i|^3}\right) * (X_i - O_i)\right)' \quad (186)$$

$$H_{o_i} = w_o * -2 * \left((X_i - O_i) * \left(-\frac{(X_i - O_i)^T}{|X_i - O_i|^6} - \left(\frac{1}{|X_i - O_i|} - \frac{1}{d_{min}}\right) * \frac{3 * (X_i - O_i)^T}{|X_i - O_i|^5}\right) + \left(\frac{1}{|X_i - O_i|} - \frac{1}{d_{min}}\right) * \frac{1}{|X_i - O_i|^3}\right) \quad (187)$$

Así mismo, para un punto $X_i = [x_i, y_i]$ la Hessiana $H_{c_{i_j}}$ del término fc_{i_j}, que describe a la restricción cinemática con circunferencias de mínimo radio de giro, es de la siguiente forma:

$$H_{c_{ij}} = w_c * -2 * \left((X_i - C_j) * \left(-\frac{(X_i - C_j)^T}{|X_i - C_j|^6} - \left(\frac{1}{|X_i - C_j|} - \frac{1}{r_{min}} \right) * \frac{3 * (X_i - C_j)^T}{|X_i - C_j|^5} \right) + \left(\frac{1}{|X_i - C_j|} - \frac{1}{r_{min}} \right) * \frac{1}{|X_i - C_j|^3} \right) \quad (188)$$

Y para un punto $X_t = [x_t, y_t]$, la Hessiana $H_{d_{t_v}}$, asociada al término $f_{d_{t_v}}$, que describe la distancia con un móvil v en el instante t es de la siguiente forma:

$$H_{d_{t_v}} = w_v * -2 * \left((X_t - Xv_t) * \left(-\frac{(X_t - Xv_t)^T}{|X_t - Xv_t|^6} - \left(\frac{1}{|X_t - Xv_t|} - \frac{1}{d_{car}} \right) * \frac{3 * (X_t - Xv_t)^T}{|X_t - Xv_t|^5} \right) + \left(\frac{1}{|X_t - Xv_t|} - \frac{1}{d_{car}} \right) * \frac{1}{|X_t - Xv_t|^3} \right) \quad (189)$$

De estas tres últimas, no se puede inferir si f_{o_i}, f_{c_i} y f_{d_t} son convexas, mas sí se puede notar que sus Hessianas son discontinuas cuando $X_i = O_i$, cuando $X_i = C_j$ y cuando $X_t = Xv_t$ respectivamente. Por otro lado, las funciones de costo original f_{o_i}, f_{c_i} y f_{d_t} son discontinuas, pues incluyen una multiplicación elemento a elemento que fija todos los valores a cero cuando no se infringe una distancia mínima.

4.1.4 Algoritmo de Optimización

Debido a que las determinantes de las matrices Hessianas H_{l_i} y H_{s_i} son nulas y por ende no invertibles, no se puede aplicar un método de optimización que invierta a la matriz Hessiana, como el método de Raphson-Newton. Por ello, se opta por utilizar el algoritmo de descenso de gradiente aplicado a todos los puntos del camino en simultáneo.

Así, para una matriz $\mathbb{X}$ que contiene a los N puntos del camino de la siguiente forma:

$$\mathbb{X} = \begin{bmatrix} x_0 & x_1 & \dots & x_N \\ y_0 & y_1 & \dots & y_N \end{bmatrix} \quad (190)$$

Se forma la matriz de gradientes, con las derivadas calculadas, tal que el punto inicial y los puntos de espera tengan un gradiente nulo para ser exentos de la optimización.

$$\nabla_{\mathbb{X}_{i-1}} = \begin{bmatrix} \bar{0} & \left(\frac{df_1}{dX_0}\right)^T & \left(\frac{df_2}{dX_1}\right)^T & \dots & \left(\frac{df_{N-1}}{dX_{N-1}}\right)^T & \bar{0} \end{bmatrix} \quad (191)$$

$$\nabla_{\mathbb{X}_i} = \begin{bmatrix} \bar{0} & \left(\frac{df_1}{dX_1}\right)^T & \left(\frac{df_2}{dX_2}\right)^T & \dots & \left(\frac{df_{N-1}}{dX_{N-1}}\right)^T & \left(\frac{df_{end}}{dX_{end}}\right)^T \end{bmatrix} \quad (192)$$

$$\nabla_{\mathbb{X}_{i+1}} = \begin{bmatrix} \bar{0} & \left(\frac{df_1}{dX_2}\right)^T & \left(\frac{df_2}{dX_3}\right)^T & \dots & \left(\frac{df_{N-1}}{dX_N}\right)^T & \bar{0} \end{bmatrix} \quad (193)$$

$$\nabla_{\mathbb{X}} = \nabla_{\mathbb{X}_{i-1}} + \nabla_{\mathbb{X}_i} + \nabla_{\mathbb{X}_{i+1}} \quad (194)$$

Luego, la actualización de los puntos del camino, para cada iteración del algoritmo queda definida de la siguiente manera:

$$\mathbb{X}_{k+1} = \mathbb{X}_k - \eta * \nabla_{\mathbb{X}} \qquad (195)$$

Donde η es una constante que escala al gradiente.

Así mismo, para poder, aplicar la función de costo f_d asociada a la distancia entre móviles para mismos instantes de tiempo, se debe corregir la diferencia de tiempo en las trayectorias, originadas por el planeamiento SIPP, tal que estas compartan los mismos tiempos. Para ello, se debe interpolar las trayectorias de todos los móviles en el tiempo antes de iniciar la optimización y luego, corregir la interpolación de las posiciones interpoladas entre al punto de espera y el siguiente punto de la trayectoria, tal que estos coincidan con la posición del punto de espera, ver Figura 4.4.

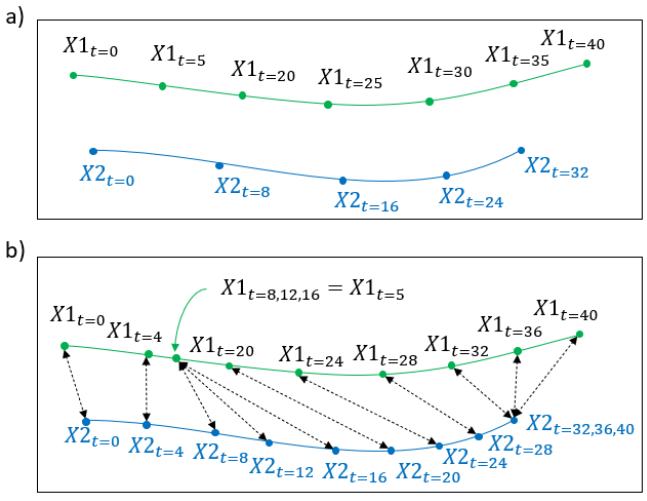


Figura 4.4: a) Trayectorias con tiempos distintos, b) Trayectorias interpoladas en el tiempo con corrección en puntos de espera [Elaboración propia].

Luego, bastará con optimizar todas las trayectorias de manera semi simultánea, compartiendo la información de sus posiciones para la evasión de colisiones, mas no la de sus gradientes. Resultando en v problemas de optimización desacoplados los cuales actualizan y cesan de manera independiente.

La optimización de trayectorias resultantes del planeamiento de v vehículos sería tal como se muestra en Figura 4.5. En esta, se define el criterio de convergencia que cesa la optimización, los pesos de la función de costo, la tasa de aprendizaje y máximo número de iteraciones y luego se llama a la función de optimización de trayectorias.

```
inicio = {configuracionInicial1,configuracionInicial2,configuracionInicial3}
final  = {configuracionFinal1,configuracionInicial2,configuracionInicial3}
trayectorias = planeadorGlobal(inicio,final,mapa)
wl = 1e1
ws = 1e0
wo = 1e5
wc = 1e5
we = 1e1
wd = 1e5
pesos = {wl,ws,wo,wc,we,wd}
lr  = 1e-2
critConv = 1e-5
maxIteraciones = 2000
trayectoriasOptim = optimTrayectorias(trayectorias,pesos,maxIteraciones,critConv,lr)
```

Figura 4.5: Optimización de trayectorias resultantes del planeamiento [Elaboración propia].

Mientras que el pseudocódigo del algoritmo de optimización de trayectorias usando descenso de gradiente de forma desacoplada es tal como se muestra en Figura 4.6.

```
funcion optimTrayectorias(trayOr,pesos,maxIteraciones,critConv,lr):
    periodo   = obtenPeriodo(trayOr)       % calcula tiempo de muestreo
    trayInt   = interpola(trayOr,periodo)  % interpola la data
    trayInt   = corrigeInterpolacion(trayOr,trayInt) % corrige interpolación
    nv        = numeroDetrayectorias(trayInt) % número de móviles
    actualiza = ceros([1 nv])              % indicador de actualización
    i = 0
    para v = 1 a nv:
        X{v} = trayInt(v).X;

    mientras i < maxIteraciones:           % optimización desacoplada
        i = i+1;
        si suma(actualiza) == 0:           % si todos convergieron
            salir del bucle                % cesa el algoritmo
        para v = 1 a nv:                   % para cada movil v
            si actualiza(v) == falso:      % si convergió anteriormente
                continua                   % pasar al siguiente móvil
            f{v} = calculaCosto(v,X,mapa,pesos)
            si abs(f{v}-fAnterior{v})/f{v} < critConv:   % si converge
                actualiza(v) = falso       % cesar actualización
                continua                   % pasar al siguiente móvil
            G{v} = calculaGradientes(v,X,mapa,pesos);
        para v = 1 a nv:
            si actualiza(v) == true   % actualiza solo si no ha convergido
                X{v} = X{v} - lr*G{v}
                fAnterior{v} = f{v};

    para v = 1 a nv:
        trayectOptim(v).X = X{v};         % guarda trayectoria optimizada

    retorna trayectOptim;
```

Figura 4.6: Pseudocódigo de algoritmo de optimización de trayectorias [Elaboración propia].

En dicho pseudocódigo, se interpola las trayectorias para un periodo de tiempo y se corrige las posiciones entre el punto de espera y el siguiente. Enseguida, se procede con la optimización por descenso de gradiente de forma desacoplada, calculando para cada iteración i los gradientes de la función de costo asociada a la trayectoria de cada móvil “v”, y luego, de calcular todos los gradientes para cada móvil “v”, se actualiza la posición X de todos los móviles. Finalmente, al cumplirse el criterio de convergencia o exceder el máximo número de iteraciones se para la optimización y se almacenan las trayectorias optimizadas.

4.2 Control

Dado que los móviles deben aparcarse en puntos de espera para evitar posibles colisiones, se requiere de un controlador, que permita variar la velocidad del móvil. Así mismo, el planeamiento realizado asume que los móviles se mueven a bajas velocidades sin deslizamiento alguno (pues de otra manera no seguirían las curvas de Dubins); por ello, se hará uso solo del modelo cinemático del móvil. Y como el deslizamiento es nulo, y solo las ruedas frontales pueden girar, se considera los valores de $\delta_f = \delta$, $\delta_r = 0$ y $\beta = 0$. Luego, reemplazando estos valores en las ecuaciones cinemáticas del móvil, se llega a lo siguiente:

$$\dot{V} = a \quad (196)$$

$$\dot{X} = V * cos(\theta) \quad (197)$$

$$\dot{Y} = V * sin(\theta) \quad (198)$$

$$\dot{\theta} = V * \frac{\tan(\delta)}{l_f + l_r} \quad (199)$$

A partir de estas ecuaciones, se diseñará un controlador LQR, para ello, se necesita primero tener el modelo linealizado del sistema.

Como se evidencia en [49], [50] y [51], el modelo cinemático expuesto es un sistema flatness diferencial, el cual es linealizable por extensión dinámica. A continuación, se provee la definición de sistema flatness diferencial:

4.2.1 Sistema Flatness Diferencial

En [48] se define a un sistema flatness diferencial, como un sistema en el cual se puede describir todos los elementos del vector de estado del sistema y los valores de las entradas del sistema, usando solo un conjunto de salidas flat del mismo.
Formalmente, esto es: Un sistema $\dot{x} = f(x,u)$ con un vector de estado $x \in R^n$ y una entrada $u \in R^n$, es flatness diferencial si existe un vector de salida $F \in R^n$ de la forma:

$$F = h(x, u, \dot{u}, \dots, u^{(r)}) \quad (200)$$

Tal que:

$$x = \phi(F, \dot{F}, \dots, F^{(q)}) \quad (201)$$

$$u = \alpha(F, \dot{F}, \dots, F^{(q)}) \quad (202)$$

De esta notación se puede identificar que el valor de la entrada de control u que llevaría al sistema a un estado x (es decir la ley de control), puede ser determinada si se conoce la salida F, obteniéndose así una ley de control no lineal, a partir de la

definición de sistema flatness diferencial. Esta relación de entrada u a salida F conlleva a sí mismo a la linealización por realimentación dinámica del sistema, la cual es llamada linealización por extensión dinámica, ya que se extiende el sistema con las derivadas de la entrada de control u.

4.2.2 Modelo Cinemático Flatness Diferencial

Se puede demostrar que el modelo cinemático es flatness diferencial tomando las salidas flat como:

$$F = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix} \quad (203)$$

Tanto la posición, la velocidad, como la orientación del móvil, pueden ser descritas usando solo las salidas flat del sistema:

$$X = F_1 \quad (204)$$

$$Y = F_2 \quad (205)$$

$$V = \sqrt{\dot{F_1}^2 + \dot{F_2}^2} \quad (206)$$

$$\theta = \operatorname{atan}\left(\frac{\dot{F_2}}{\dot{F_1}}\right) \quad (207)$$

Y así mismo la dinámica del sistema de la siguiente manera:

$$\dot{\theta} = \frac{\dot{F_2} * F_1 - \dot{F_1} * F_2}{\dot{F_1}^2 + \dot{F_2}^2} \quad (208)$$

$$\dot{V} = \frac{\dot{F_1} * \ddot{F_1} + \dot{F_2} * \ddot{F_2}}{\sqrt{\dot{F_1}^2 + \dot{F_2}^2}} \quad (209)$$

$$\dot{X} = \dot{F_1} = \sqrt{\dot{F_1}^2 + \dot{F_2}^2} * cos\left(\operatorname{atan}\left(\frac{\dot{F_2}}{\dot{F_1}}\right)\right) \quad (210)$$

$$\dot{Y} = \dot{F_2} = \sqrt{\dot{F_1}^2 + \dot{F_2}^2} * sin\left(\operatorname{atan}\left(\frac{\dot{F_2}}{\dot{F_1}}\right)\right) \quad (211)$$

Es decir, el vector de estado $x = [X, Y, \theta, V]$, tiene la forma $x = \phi(F, \dot{F}, \dots, F^{(q)})$ y así mismo, definiendo a y δ_f como las entradas del sistema, se puede describir a estas también en función de las salidas Flat:

$$u_1 = \dot{V} = \frac{\dot{F_1} * \ddot{F_1} + \dot{F_2} * \ddot{F_2}}{\sqrt{\dot{F_1}^2 + \dot{F_2}^2}} \quad (212)$$

$$\dot{\theta} = V * \frac{\tan(u_2)}{l_f + l_r} \quad (213)$$

$$u_2 = \operatorname{atan}\left(\left(l_f + l_r\right) * \frac{\ddot{F}_2 * \dot{F}_1 - \ddot{F}_1 * \dot{F}_2}{\left(\dot{F_1}^2 + \dot{F_2}^2\right)^{\frac{3}{2}}} \right) \quad (214)$$

Teniéndose así un vector de entrada de la forma $u = \alpha(F, \dot{F}, \ldots, F^{(q)})$. Probándose así que la dinámica del sistema y todas las variables usadas en esta pueden ser descritas usando solo las salidas flat y sus derivadas, por ende, el sistema es de tipo flatness diferencial.

4.2.3 Linealización por Extensión Dinámica

Siguiendo la linealización realizada en [52] para linealizar el sistema por extensión dinámica, se deriva las salidas flat con respecto al tiempo, hasta encontrar una relación entrada salida con las entradas extendidas. De la segunda derivada de las salidas flat, se tiene lo siguiente:

$$\ddot{F}_1 = \dot{V} * \ cos(\theta) - \frac{\tan(\delta_f)}{l_f + l_r} * V^2 * sin(\theta) \quad (215)$$

$$\ddot{F}_2 = \dot{V} * \ sin(\theta) + \frac{\tan(\delta_f)}{l_f + l_r} * V^2 * cos(\theta) \quad (216)$$

Definiendo $\eta_\delta = \tan(\delta_f)/(l_f + l_r)$ y a como la entrada extendida, se puede expresar las ecuaciones de la siguiente forma matricial:

$$\begin{bmatrix} \ddot{F}_1 \\ \ddot{F}_2 \end{bmatrix} = \begin{bmatrix} cos(\theta) & -V^2 * sin(\theta) \\ sin(\theta) & V^2 * cos(\theta) \end{bmatrix} * \begin{bmatrix} a \\ \eta_\delta \end{bmatrix} \quad (217)$$

Resultando en una relación entrada salida con las entradas a y η_δ. Despejando las ecuaciones de la matriz se puede obtener también la relación de salida a entrada:

$$\begin{bmatrix} a \\ \eta_\delta \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta)/V^2 & cos(\theta)/V^2 \end{bmatrix} * \begin{bmatrix} \ddot{F}_1 \\ \ddot{F}_2 \end{bmatrix} \quad (218)$$

Así, de conocerse $\ddot{F}_1$ y $\ddot{F}_2$ se puede determinar a y η_δ, y de este último δ de la siguiente manera:

$$\delta = atan\left(\left(l_f + l_r\right) * \eta_\delta\right) \quad (219)$$

Nótese que, existe una singularidad para el cálculo de η_δ cuando la velocidad es igual a cero. Por ende, el valor δ tenderá a más o menos infinito cuando la velocidad del móvil se aproxime a cero.

Así mismo, el sistema puede ser representado de manera lineal de la siguiente manera:

$$\begin{bmatrix}\dot{F}_1\\ \dot{F}_2\\ \ddot{F}_1\\ \ddot{F}_2\end{bmatrix}=\begin{bmatrix}0&0&1&0\\0&0&0&1\\0&0&0&0\\0&0&0&0\end{bmatrix}*\begin{bmatrix}F_1\\F_2\\\dot{F}_1\\\dot{F}_2\end{bmatrix}+\begin{bmatrix}0&0\\0&0\\1&0\\0&1\end{bmatrix}*\begin{bmatrix}\ddot{F}_1\\\ddot{F}_2\end{bmatrix} \quad (220)$$

$$\dot{X}_F = A_F * X_F + B_F * U_F \quad (221)$$

Luego, bastará con controlar este sistema, para conocer los valores de $\ddot{F}_1$ y $\ddot{F}_2$ y por medio de la relación de entrada salida con a y η_δ se hallará las entradas a y δ al modelo cinemático.

4.2.4 Seguimiento de Trayectoria

Teniendo el sistema linealizado y una trayectoria de referencia, es posible aplicar cualquier método de control de seguimiento de trayectoria para sistemas lineales. Definiéndose la ley de control como:

$$U_F - U_F^d = -K(X_F - X_F^d) \quad (222)$$

$$U_F = U_F^d - K(X_F - X_F^d) \quad (223)$$

Se halla el valor de K por control óptimo:

$$K = R^{-1} * B_F^T * P \quad (224)$$

Donde la matriz P es obtenida de la solución de la ecuación de Riccati:

$$A_F^T * P + P * A_F - P * B_F * R^{-1} * B_F^T + Q = 0 \quad (225)$$

Para el problema de control óptimo, los pesos de Q de la función de costo son variados tal que la minimización de los errores e_1 y e_2 primen en la optimización; asociándoseles valores altos a dichos elementos, luego, la matriz Q queda de la siguiente forma:

$$Q = \begin{bmatrix}Q_{big}&0&0&0\\0&Q_{big}&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} \quad (226)$$

Mientras que R se define como:

$$R = \begin{bmatrix}1&0\\0&1\end{bmatrix} \quad (227)$$

De la ley de control, se puede obtener los valores de $\ddot{F} = \left[\ddot{F}_1, \ddot{F}_2\right]^T$, de la siguiente forma:

$$U_F = U_F^d - K(X_F - X_F^d) \quad (228)$$

$$\ddot{F} = \ddot{F}^d - K * (X_F - X_F^d) \quad (229)$$

El diagrama de bloques del controlador sería tal como se muestra en Figura 4.7.

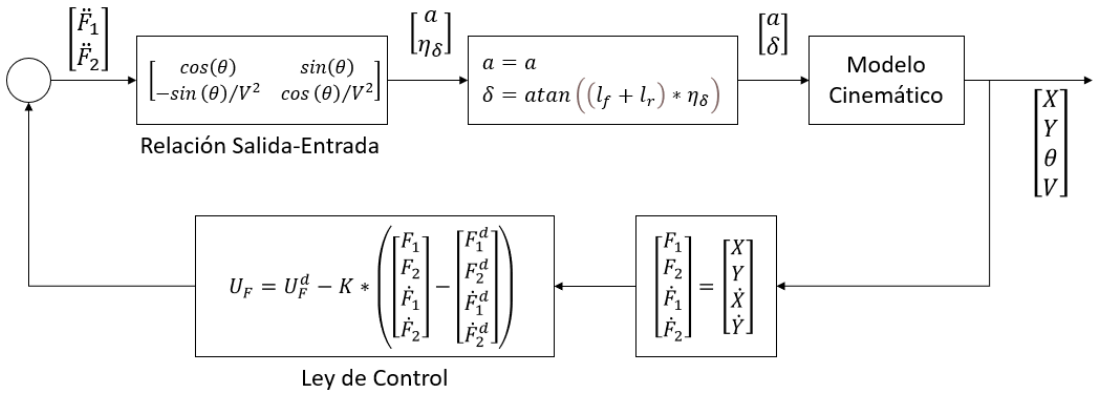


Figura 4.7: Diagrama de bloques de controlador [Elaboración propia].

4.3 Integración del Sistema

Finalmente, se une el planeamiento, la generación de trayectoria y el control tal como se muestra en Figura 4.8.

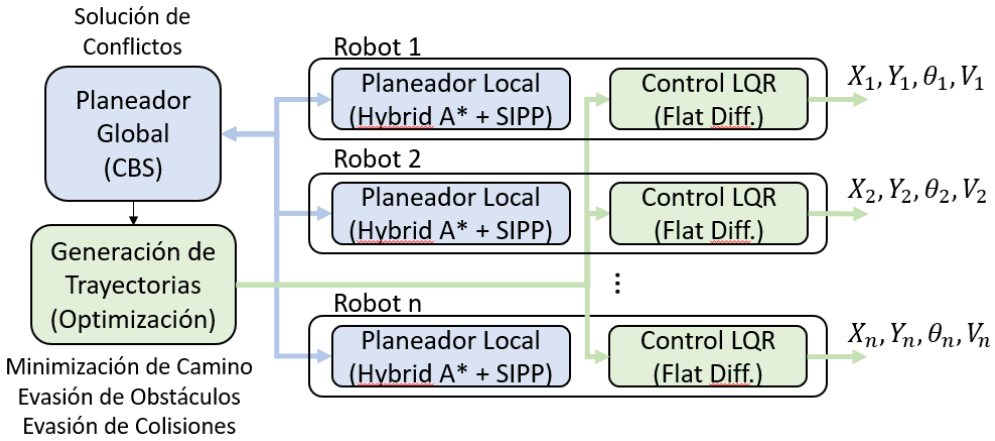


Figura 4.8: Diagrama de integración del planeamiento, generación de trayectoria y control de seguimiento de trayectoria [Elaboración propia].

Donde cada robot planea de manera independiente su camino en el tiempo a seguir con su respectivo planeador local. Luego, el planeador global compara los caminos de los robots, y de encontrar colisiones, impone restricciones a los planeadores locales de los robots envueltos en la colisión hasta que no se detecten más colisiones. Luego, los caminos en el tiempo de todos los robots son enviados a un generador de trayectorias, o propiamente dicho un optimizador de trayectorias, que minimiza la longitud de los caminos a seguir respetando las restricciones de colisión con obstáculos estáticos del entorno (evasión de obstáculos) y la colisión con otros robots (evasión de colisiones). Finalmente, la trayectoria optimizada resultante de cada robot es utilizada como entrada de referencia por el controlador óptimo de seguimiento de trayectoria del respectivo robot. Obteniéndose la simulación del ploteo de los resultados de cada uno de los robots.

Capítulo 5: Simulación

Con el fin de simplificar la implementación de las múltiples partes del sistema se omite el uso de velocidades negativas durante el planeamiento, pues esto implicaría extender el cálculo de la distancia no holonómica de 6 a 48 tipos curvas y agregar múltiples exclusiones y puntos de espera en la optimización.

5.1 Simulación del Planeamiento

5.1.1 Visualización del Planeamiento

En la simulación del planeamiento, mostrada en Figura 5.1, se muestra la configuración origen del móvil (la orientación y posición del punto de partida) en líneas azules, la configuración de destino ideal del móvil (orientación y posición de destino) en líneas azules punteadas, las configuraciones discretas exploradas para ir del origen al destino en flechas azules, el camino optimo encontrado en líneas azules y las configuraciones continuas que da el móvil por cada paso del camino óptimo en puntos negros.

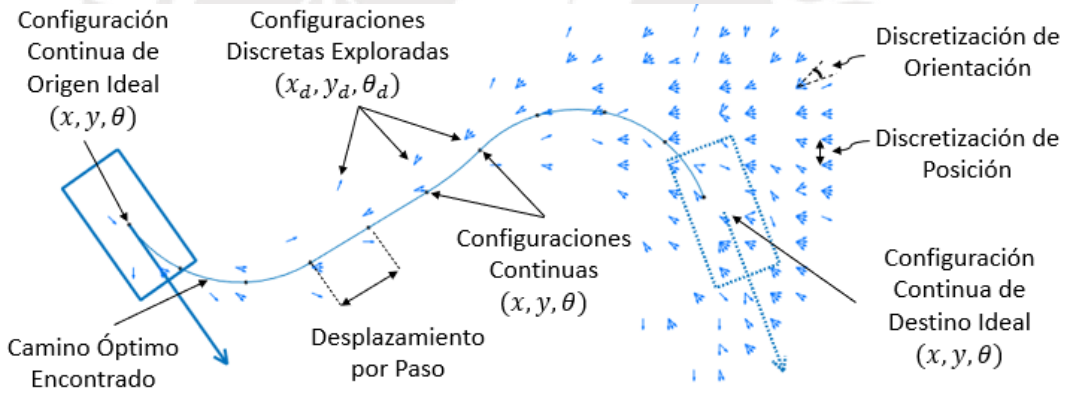


Figura 5.1: Elementos de Simulación de planeamiento [Elaboración propia].

Donde las configuraciones son discretizadas en base a una distancia y orientación mínima. Para las pruebas realizadas, la discretización de posición es seteado en 3 y la discretización de orientación en 2*pi/32. Esto quiere decir que, si el mapa es de 100x100, habrá 33x33 posiciones discretizadas en el mapa y por cada posición discretizada del mapa habrá 32 posibles orientaciones en el que el móvil puede estar sobre dicho punto, sumando un total de 32x33x33 posibles configuraciones a explorar. Por otro lado, los movimientos del móvil son discretizados, fijándose en 5 el número de giros discretos a realizar por el timón y fijándose el desplazamiento del móvil igual al ancho del mismo. Así, de tener el timón un giro máximo de pi/4, se crearían 5 siguientes configuraciones, correspondientes a desplazar el móvil una distancia igual a su ancho con ángulos de steering de -pi/4, -pi/8, 0 pi/8 y pi/4.

5.1.2 Pruebas del Planeador Local

a) Por Tipo de Mapa

A continuación, se prueba el planeador local ante diferentes tipos de mapa, los cuales se muestran para el planeamiento de un móvil en Figura 5.2. Donde todos los mapas poseen dimensiones de 450x750 pixeles.

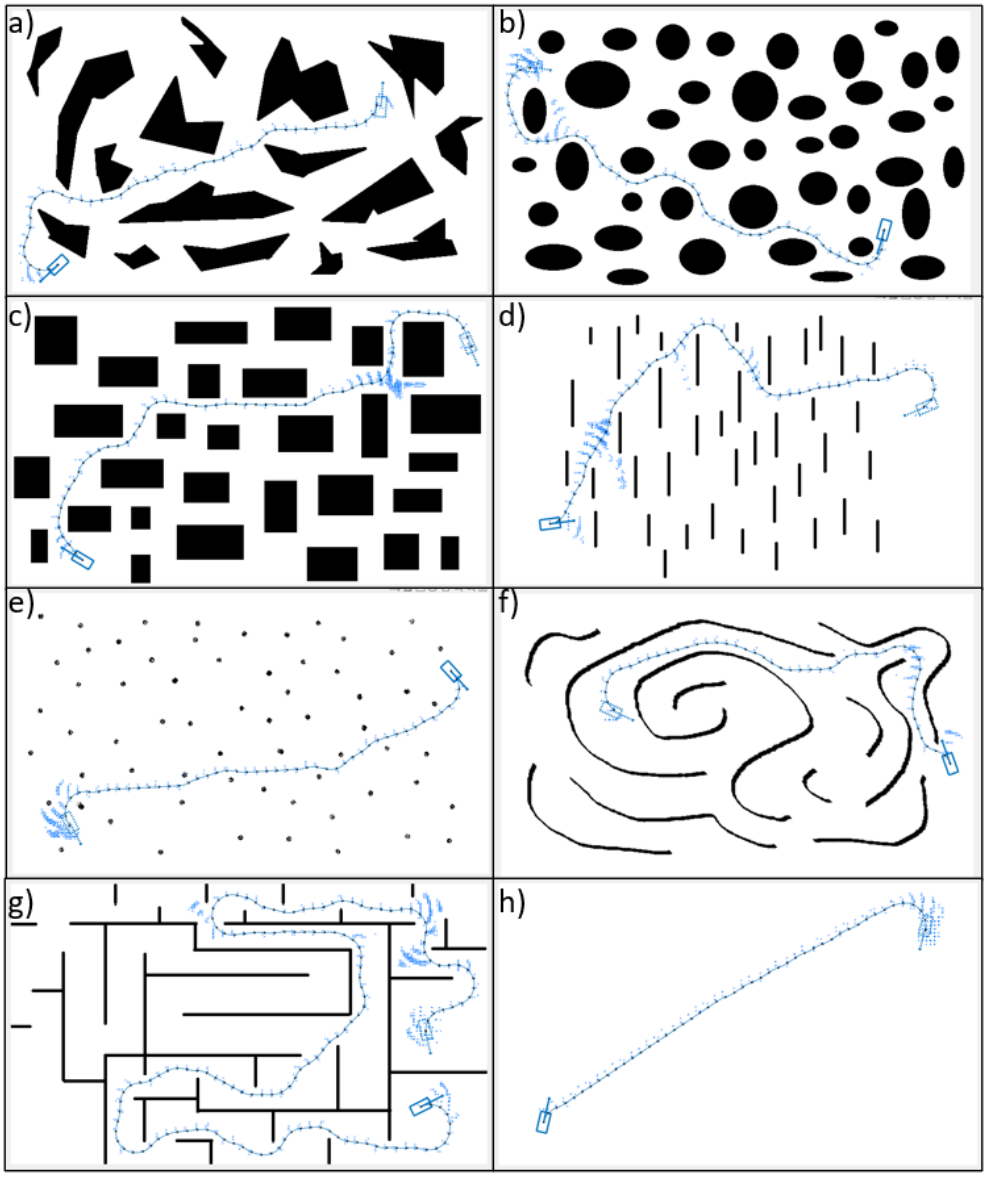


Figura 5.2: Planeamiento de un móvil para mapa a) de obstáculos no convexos, b) de obstáculos redondos, c) de obstáculos rectangulares, d) de obstáculos de barras, e) de obstáculos de puntos dispersos, f) de estructuras curvas, g), de estructuras rectas y h) sin obstáculos [Elaboración propia].

Para testear al planeador, se realiza 100 simulaciones aleatorias por cada mapa de Figura 5.2, en las cuales se lleva a un vehículo de dimensiones de 32x16 pixeles, equivalente a 3.2x1.6 metros, desde una configuración origen aleatoria a una configuración destino aleatoria.

Los resultados de las pruebas se muestran en la Tabla 3, donde las abreviaturas Despl., Cfg. Expl. y Std significan desplazamiento, configuraciones exploradas y desviación estándar respectivamente.

Tabla 3: Resultados del planeamiento en diferentes tipos de mapa, luego de 100 ejecuciones aleatorias por mapa [Elaboración propia].

Tipo de Mapa	Parámetros	Mediana	Media	Std	Min	Max
Obstáculos No Convexos (450x750 px)	Tiempo (s)	1.63	4.44	5.84	0.81	29.00
	Cfg. Expl.	2311	9909.00	15451	84	73474
	Pasos	36	37.52	13.1	6	77
	Despl. (px)	584	600.32	209.61	96	1232
Obstáculos Redondos (450x750 px)	Tiempo (s)	1.04	2.65	4.85	0.83	42.79
	Cfg. Expl.	582	4496.01	11190	11	94354
	Pasos	30	30.83	11.06	7	66
	Despl. (px)	480	493.28	176.93	32	1056
Obstáculos Rectangulares (450x750 px)	Tiempo (s)	1.12	2.53	3.28	0.84	22.96
	Cfg. Expl.	886	1037.10	7103	69	45340
	Pasos	35	36.21	11.80	6	67
	Despl. (px)	56	579.36	188.88	96	1072
Obstáculos de Barras (450x750 px)	Tiempo (s)	1.09	1.81	1.76	0.85	12.76
	Cfg. Expl.	902	2921.30	4368	52	25807
	Pasos	28	31.02	12.53	6	57
	Despl. (px)	448	499.04	200.53	160	912
Obstáculos de Puntos Dispersos (450x750 px)	Tiempo (s)	0.91	1.21	1.04	0.84	9.74
	Cfg. Expl.	226	1398.11	3294	33	25105
	Pasos	23	23.58	9.88	5	55
	Despl. (px)	376	377.28	158.07	80	880
Obstáculos de Estructuras Curvas (450x750 px)	Tiempo (s)	1.43	5.21	6.65	0.82	31.08
	Cfg. Expl.	1765	12000.10	17712	20	80662
	Pasos	44	43.75	17.99	7	92
	Despl. (px)	704	700.00	387.83	32	1472
Obstáculos de Estructuras Rectas (450x750 px)	Tiempo (s)	2.09	10.51	13.27	0.84	44.06
	Cfg. Expl.	5217	22679.31	30164	24	101714
	Pasos	106	99.18	48.12	5	245
	Despl. (px)	1696	1586.32	769	80	3920
Sin Obstáculos (450x750 px)	Tiempo (s)	1.02	1.08	0.23	0.93	2.61
	Cfg. Expl.	232	809.64	1517	18	9869
	Pasos	23	24.50	9.99	8	52
	Despl. (px)	368	392.21	159.78	32	832

De los resultados se puede observar que, para mapas con obstáculos, en el mejor de los casos, al planeador le toma entre 0.81 a 0.84 segundos encontrar el camino de un móvil; mientras que en el peor de los casos le toma de 12.76 a 44.06 segundos. Así mismo se puede notar que dependiendo del tipo de mapa, el tiempo

promedio para encontrar un camino puede variar de 1.08 a 10.51 segundos. Por otro lado, si se analiza la mediana (con el fin de evitar sesgos de valores muy altos en el promedio), se puede notar que al planeador le suele tomar entre 1.02 a 2.09 segundos encontrar el camino y explorar entre 232 a 5217 configuraciones para encontrar un camino formado de entre 23 a 106 pasos (configuraciones).

Así mismo se puede notar que para mapas de obstáculos, el mejor rendimiento se da en el mapa de obstáculos de puntos dispersos (con 1.21 segundos y 1398 configuraciones exploradas) y el de peor rendimiento se da en el de obstáculos de estructuras rectas (10.51 segundos y 22679 configuraciones exploradas). Cabe precisar que se espera que el rendimiento sea menor en los obstáculos de estructuras, dado que forman caminos cerrados de mayor longitud. En orden del mayor a menor rendimiento se tiene: mapa sin obstáculos, obstáculos de puntos dispersos, obstáculos de barras, luego el de obstáculos redondos, enseguida el de obstáculos rectangulares, obstáculos no convexos, estructuras curvas y estructuras rectas.

b) Por Tamaño de Mapa

Por otro lado, se analiza la efectividad del planeador para diferentes tamaños de mapa. Para ello, se utilizará el mapa de barras trasversales que se muestra para el planeamiento de un móvil en Figura 5.3, se variará el tamaño del mismo entre 250x250 pixeles a 2500x2500 pixeles y se planeará 10 veces por cada mapa.

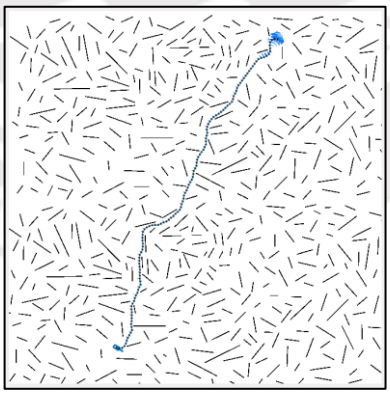


Figura 5.3: Planeamiento de móvil en mapa de barras trasversales de 2000x2000 [Elaboración propia].

Nuevamente, las pruebas son realizadas para un móvil de dimensiones de 32x16 y la discretización del mapa (posición y ángulo de orientación) y del móvil (ángulos discretos de giro) son iguales a los especificados en las pruebas anteriores. Los resultados del planeamiento ante diferentes tamaños del mapa, se muestran en la Tabla 4.

Tabla 4: Resultados de planeamiento ante variación de tamaño del mapa, luego de 10 ejecuciones por mapa [Elaboración propia].

Tamaño de Mapa	Parámetros	Mediana	Media	Std	Min	Max
2500x2500	Tiempo (s)	19.67	20.19	1.98	18.39	25.68
	Cfg. Expl.	949.5	3641	6506	107	22295
2250x2250	Tiempo (s)	15.30	15.55	1.11	14.12	17.49
	Cfg. Expl.	1036	2007	1772	312	4931
2000x2000	Tiempo (s)	11.55	13.64	4.22	11.09	24.85
	Cfg. Expl.	807	3176	5062	220	17446
1750x1750	Tiempo (s)	9.25	10.15	2.44	8.78	17.23
	Cfg. Expl.	1168	3891	6447	311	21970
1500x1500	Tiempo (s)	6.30	6.96	1.96	5.96	12.77
	Cfg. Expl.	773.5	3236	6903	87	23817
1250x1250	Tiempo (s)	4.58	5.44	2.40	4.21	12.50
	Cfg. Expl.	1444	4216	6617	328	23135
1000x1000	Tiempo (s)	3.00	4.99	4.07	2.71	16.24
	Cfg. Expl.	1160	6176	10503	96	35181
750x750	Tiempo (s)	2.05	4.10	4.663	1.52	17.59
	Cfg. Expl.	1679	6639	10745	162	37032
500x500	Tiempo (s)	0.91	3.86	6.07	0.66	20.77
	Cfg. Expl.	812	8213	14167	68	46576
250x250	Tiempo (s)	0.28	1.97	3.21	0.16	10.76
	Cfg. Expl.	380	4475	7163	36	22927

De la Tabla 4 se puede notar que el tiempo incrementa de manera exponencial o conforme se aumenta la dimensión del mapa, ver Figura 5.4. Esto se debe a que antes de iniciar el planeamiento, se realiza el cálculo de la distancia holonómica de cada punto del mapa usando el algoritmo de Dijkstra, por ello, al aumentar las dimensiones del mapa, el tiempo de solución aumenta. Por otro lado, de evaluar las configuraciones exploradas, no se encuentra algún tipo de patrón. Sin embargo, es de inferirse que, de incrementar el tamaño del mapa, los caminos a hallar son más largos y por lo tanto incrementan el tiempo de solución.

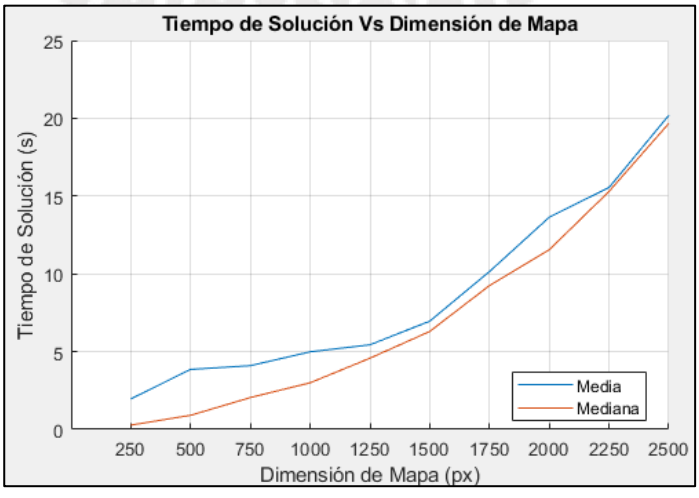


Figura 5.4: Tiempo de solución Vs Dimensión de mapa [Elaboración propia].

5.1.3 Pruebas del Planeador Global

a) Espacios Abiertos

Para probar al planeador global, se planea el movimiento de múltiples móviles que intersecan sus caminos en espacios abiertos. Para ello, se considera dos tipos de movimientos. El primero se denominó movimiento en dirección directa, en el cual la orientación de la configuración de origen y destino son iguales y al segundo se le denomino movimiento en dirección opuesta, en el cual la orientación del destino es opuesta a la orientación de la configuración de origen, ver Figura 5.5.

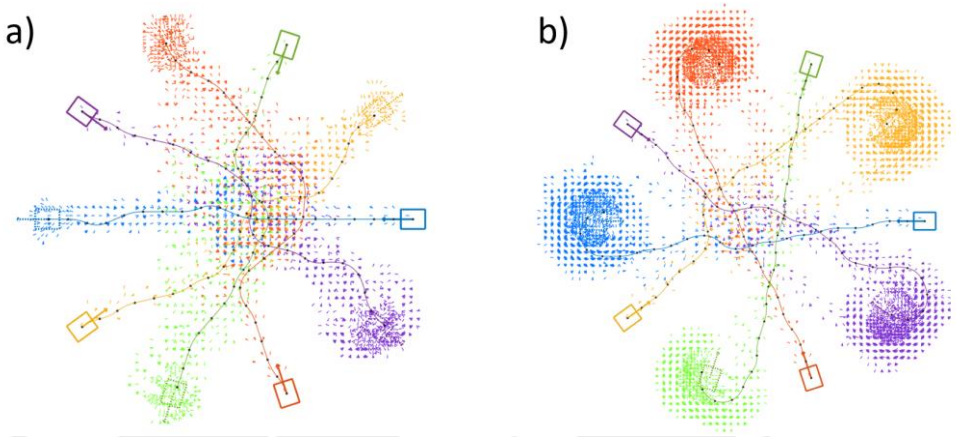


Figura 5.5: Movimiento de móviles en a) dirección directa y b) dirección opuesta [Elaboración propia].

El planeamiento de los móviles, fue realizado en un mapa vacío de dimensiones de 450x750 pixeles y los móviles utilizados fueron de dimensiones de 20x16 pixeles. Los resultados de la prueba se muestran en la Tabla 5.

Tabla 5: Resultados del planeamiento ante incremento de agentes [Elaboración propia].

Número de Móviles	Tiempo de solución (s)	Número de Restricciones	Número de Nodos CBS	Dirección
2	1.22	2	6	Directa
2	1.10	0	1	Opuesta
3	1.60	4	14	Directa
3	1.35	0	1	Opuesta
5	2.91	21	48	Directa
5	2.48	8	24	Opuesta
7	6.91	47	144	Directa
7	5.26	25	82	Opuesta
9	10.54	93	196	Directa
9	14.78	125	254	Opuesta
11	20.62	149	302	Directa
11	19.00	130	290	Opuesta
13	21.39	146	302	Directa
13	29.55	189	382	Opuesta

De esta se puede notar que el tiempo de solución del planeamiento incrementa de manera exponencial, conforme se aumenta el número de móviles, ver Figura 5.5.

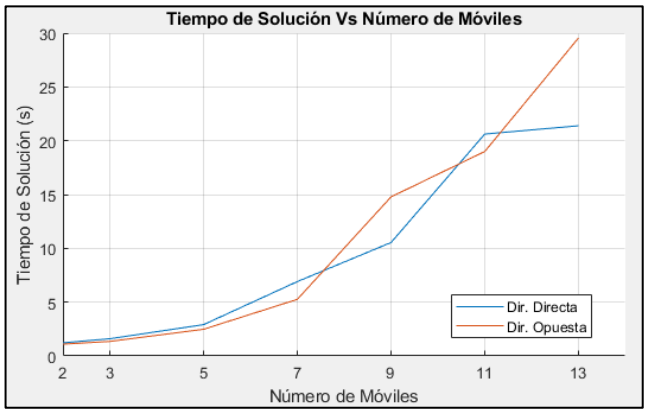


Figura 5.6: Tiempo de Solución Vs Número de móviles [Elaboración propia].

Y de la misma manera, incrementa también el número de restricciones de manera exponencial, ver Figura 5.7.

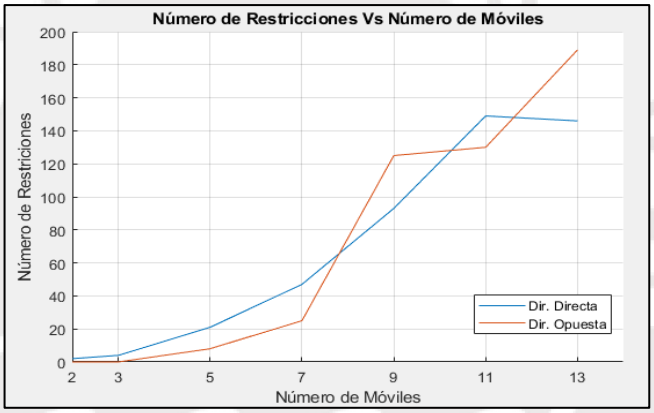


Figura 5.7: Número de Restricciones Vs Número de móviles [Elaboración propia].

Y, por ende, también el número de nodos CBS, según se aumenta el número de móviles, ver Figura 5.8.

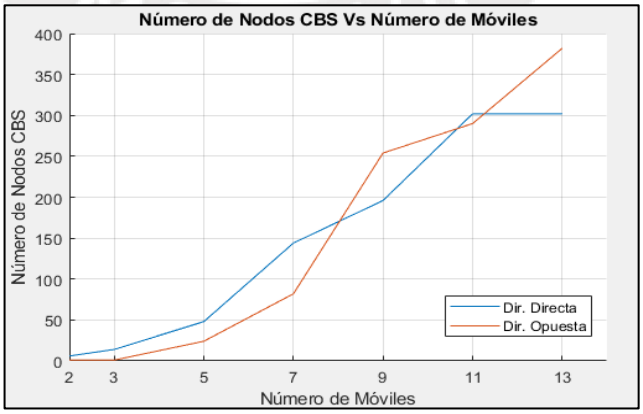


Figura 5.8: Número de nodos CBS Vs Número de móviles [Elaboración propia].

b) Pares en Entornos Restringidos

Así mismo se prueba la capacidad de solución del planeador para pares de móviles ante entornos que tienden al atasco, estos son entornos tipo T, tipo I y tipo H. Las pruebas fueron realizadas para móviles de dimensiones de 32x16 pixeles. Y los resultados de las mismas, se muestran en la Tabla 6.

Tabla 6: Resultados de planeamiento en entornos restringidos [Elaboración propia].

Mapa	Test	Tiempo de Sol. (s)	Id de Móvil	Restric-ciones	Cfg. Expl.	N° Nodos SIPP	N° Nodos CBS
Tipo T 100x450	Inversión de Orden	2.07	0	0	446	1485	20
			1	5	2729	7781	
Tipo T 100x450	Intercambio de Posición	0.33	0	2	170	304	6
			1	0	251	343	
Tipo I 150x450	Cruce de Caminos	0.50	0	0	501	744	4
			1	1	254	326	
Tipo H 250x750	Pasillo Ocupado	16.12	0	0	31525	207404	52
			1	13	22982	87228	

Donde “N° Nodos CBS” es el total de nodos CBS creados, “N° Nodos SIPP” es la suma de todos los nodos SIPP que se crean en todos los nodos CBS y “Cfg. Expl.” es el total de configuraciones exploradas.

Así mismo, los planeamientos realizados se muestran en Figura 5.9.

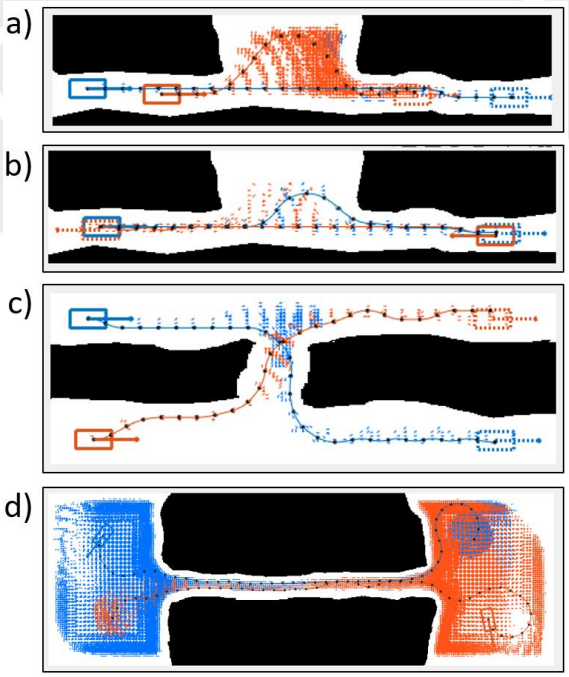


Figura 5.9: a) Inversión de orden en mapa tipo T, b) Intercambio de posición en mapa tipo T, c) Cruce de caminos en mapa tipo I y d) Pasillo ocupado en mapa tipo H [Elaboración propia].

De los resultados, se puede notar que las situaciones que demandan esperar a otro móvil por un tiempo prolongado son también las que exploran más configuraciones el mapa y por ende las que exploran más nodos SIPP asociados a dichas configuraciones. Esto debido a que exploran todos los posibles nodos SIPP de menor costo al de esperar en un punto en el mapa, con el fin de encontrar un camino alternativo. Y al no haber otro camino, terminan explorando un elevado número de configuraciones.

c) Múltiples Móviles en Mapas con Obstáculos

Con el fin de verificar que el planeamiento funciona en cualquier tipo de mapa, se planea el movimiento de múltiples móviles en mapas con obstáculos variados. Los resultados de dichas pruebas se muestran en la Tabla 7 y las pruebas realizadas, se muestran en Figura 5.10.

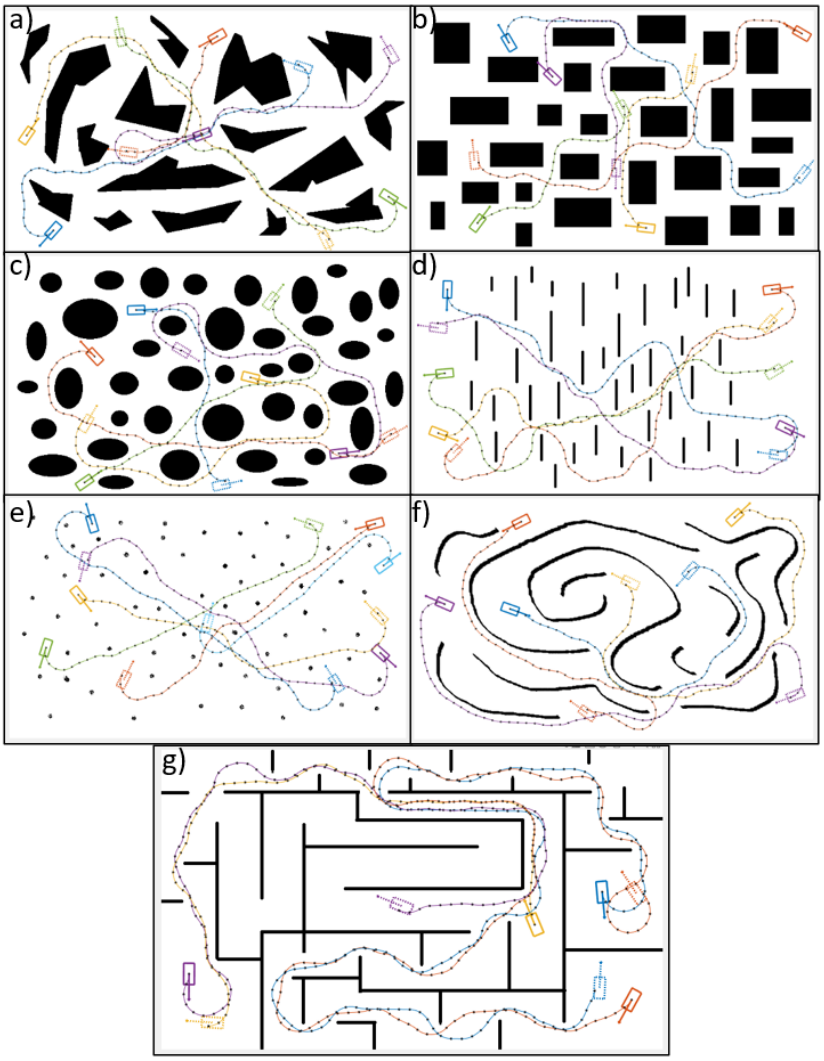


Figura 5.10: Planeamiento en mapa a) de obstáculos no convexos, b) de obstáculos rectangulares, c) de obstáculos redondos, d) de obstáculos de barras, e) de obstáculos de puntos dispersos, f) de estructuras curvas y g) de estructuras rectas [Elaboración propia].

Tabla 7: Resultados de planeamiento en mapas con obstáculos variados [Elaboración propia].

Mapa	N° de móviles	Tiempo de Sol. (s)	Id de Móvil	Restric-ciones	Cfg. Expl.	N° Nodos SIPP	N° Nodos CBS
No Convexo (750x450)	5 (32x16)	6.17	0	0	2445	8042	30
			1	2	189	434	
			2	2	1370	6937	
			3	0	3982	12599	
			4	6	920	2951	
Redondo (750x450)	5 (32x16)	15.01	0	0	713	2923	30
			1	0	1869	18397	
			2	3	2957	20048	
			3	0	3524	3524	
			4	6	18390	89305	
Rectangular (750x450)	5 (32x16)	7.57	0	0	540	3598	38
			1	7	9242	15589	
			2	2	372	1654	
			3	4	2711	20181	
			4	0	142	257	
Barras (750x450)	5 (32x16)	5.71	0	7	3588	9879	30
			1	3	3784	5113	
			2	0	3880	15349	
			3	1	1216	2335	
			4	0	597	1789	
Puntos Dispersos (750x450)	5 (32x16)	7.1	0	1	1155	2865	28
			1	0	7145	7617	
			2	0	7594	11312	
			3	2	1001	2426	
			4	4	4423	12901	
Estructuras Curvas (750x450)	4 (32x16)	40.11	0	0	12852	12852	20
			1	8	50735	449252	
			2	0	1042	6463	
			3	0	10323	10600	
Estructuras Rectas (750x450)	4 (32x16)	7.76	0	2	2320	9255	28
			1	0	8445	30063	
			2	0	2573	6150	
			3	5	3148	16103	

De la tabla se puede notar, que el planeamiento tomó mayor tiempo en el mapa de estructura curva y en el de obstáculos redondos, mientras que para el resto tomó entre 5.71 a 7.76 segundos. En el mapa de estructuras curvas, la demora se debe a que el móvil 1, colisiona múltiples veces y trata de buscar caminos alternativos que eviten dichas colisiones explorando nuevas regiones del mapa, lo cual genera un elevado número de configuraciones, en vez de esperar en una misma región a que

el resto de móviles pasen (esto evitaría que se generen tantas configuraciones). Luego, al haber más configuraciones a explorar, se generan más nodos SIPP y por ende el planeador local demora un mayor tiempo en encontrar la secuencia de nodos SIPP que describen el camino solución. Esto se evidencia al compararlo con el móvil 0 del mapa de barras, el cual tiene un número similar de restricciones, pero un menor número de configuraciones exploradas y también un menor tiempo de solución. Las razones de la demora para el mapa de obstáculos redondos son las mismas, con la diferencia que el problema ocurre en el planeamiento de móvil 4.

5.2 Simulación del Generador de Trayectoria

5.2.1 Visualización de Optimización de Trayectoria

En la simulación del generador de trayectoria, mostrada en Figura 5.11, se muestra en línea azul la trayectoria original del móvil obtenida por el planeador y en línea negra la trayectoria optimizada. Así mismo, se muestra también en líneas punteadas, las circunferencias de radio mínimo de giro del móvil en los puntos de inicio y final. De Figura 5.11, se observa también el efecto de la optimización, esto es la evasión de obstáculos, la minimización de la longitud del camino y uniformidad del mismo, la no superposición de la trayectoria sobre la circunferencia de radio mínimo de giro y la corrección de la posición final del móvil.

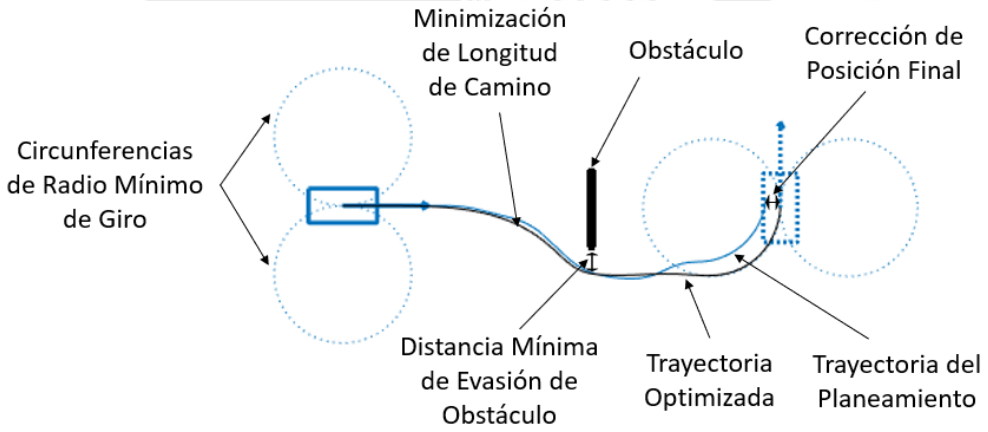


Figura 5.11: Visualización de optimización de trayectoria de un móvil [Elaboración propia].

Mientras que en Figura 5.12, se muestra el efecto de la optimización con múltiples móviles; esto es la evasión de colisiones con otros móviles, corrigiendo la posición de los mismos, tal que las posiciones de los móviles para un mismo instante Z respeten la distancia mínima de evasión de colisión.

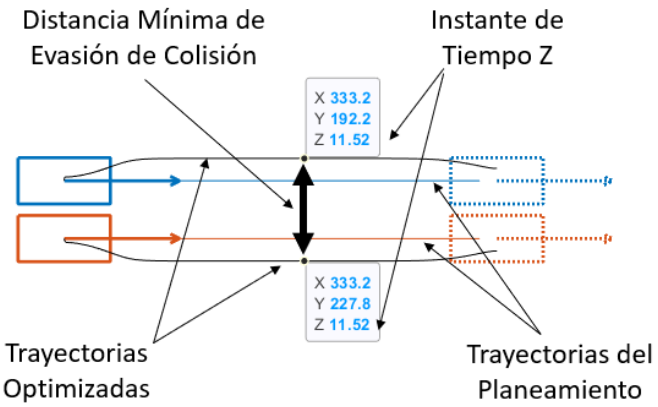


Figura 5.12: Visualización de optimización de trayectoria de múltiples móviles [Elaboración propia].

Cabe precisar que, para obtener la distancia mínima de colisión de un punto con un obstáculo del mapa, se calcula, de manera offline, la distancia euclidiana de cada punto del mapa al obstáculo más cercano y se almacenan dichas distancias en una matriz. Esta es obtenida usando el comando “bwdist” de MATLAB.

5.2.2 Pruebas del Generador de Trayectoria

a) Pruebas en Mapas Variados

El móvil de Figura 5.13 parte de la configuración [240, 257, -2.4] (esto es posición XY y orientación en radianes) y pretende llegar a la configuración [569, 125, -3.2]. Por medio del planeador, el móvil es capaz de llegar hasta el punto [571.8, 114.9] a través del camino mostrado en línea azul. Luego de la optimización, se puede observar que el móvil alcanza el punto [569, 125] a través del camino mostrado en línea negra.

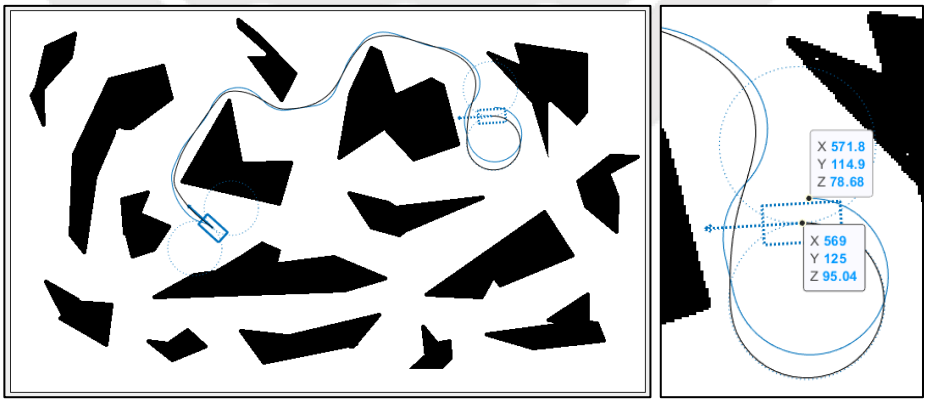


Figura 5.13: Efecto de optimización en corrección de posición final y restricción de giro en puntos de espera [Elaboración propia].

De la misma figura se puede observar, que la línea negra no se superpone a las circunferencias de Dubins que tiene el móvil a los lados, por lo que se asegura que en los puntos de inicio y final se respete la restricción física de giro del móvil.

Los resultados del planeamiento de Figura 5.13, así como los tiempos del planeamiento y optimización se muestran en Figura 5.14.

```
AgentId:0 - Total Steps: 51 - Total Cfgs Created: 8572 - Total SIPP Created:8572 - N° Rest:0
Total Restrictions: 0 - Makespan: 71.5313
Plann: Elapsed time is 2.6637 seconds
Optim: Elapsed time is 1.2389 seconds
>>
```

Figura 5.14: Resultados del planeamiento y tiempos transcurridos [Elaboración propia].

En Figura 5.15, se muestra en azul el camino obtenido por el planeador para llevar a un móvil a su destino. Durante el planeamiento el móvil tiende a acercarse a los obstáculos para encontrar el camino más corto que lo lleve a su destino. Esto en una implementación física, puede ocasionar que el móvil colisione con los obstáculos. Por ello, por medio de la optimización, se incrementa la distancia mínima de evasión de obstáculos, el resultado de la optimización se muestra en línea negra.

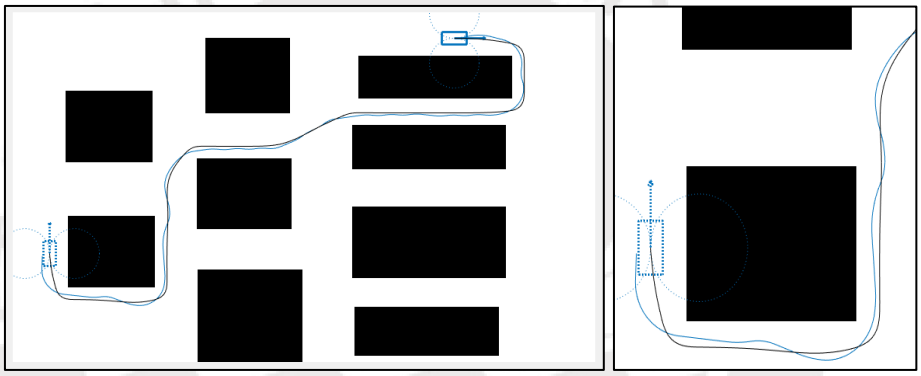


Figura 5.15: Efecto de optimización en evasión de obstáculos del entorno [Elaboración propia].

Mientras que en Figura 5.16, se muestra el efecto de la optimización (en línea negra), al minimizar longitud del camino obtenido por planeamiento (en línea azul) y al uniformizar espaciamiento entre puntos del camino.

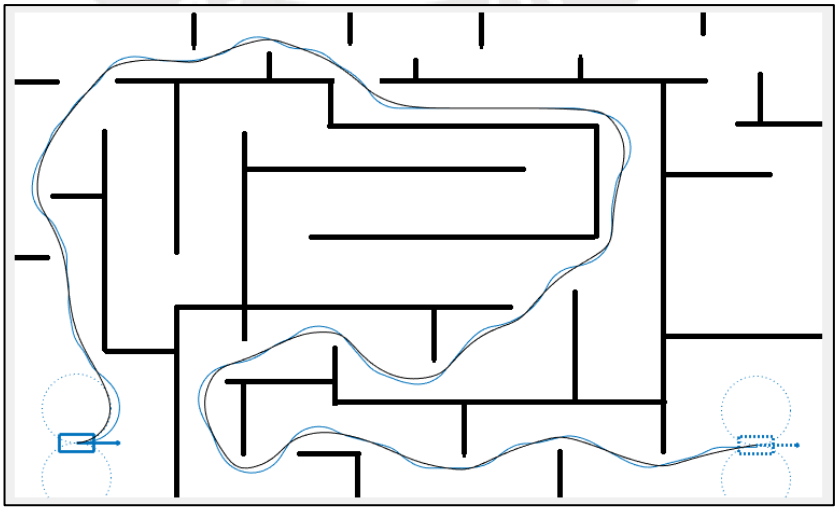


Figura 5.16: Efecto de optimización en minimización de longitud de camino [Elaboración propia].

Por otro lado, en Figura 5.17, se muestra el efecto de incrementar la distancia mínima de evasión de colisiones con otros móviles a tres veces su valor habitual (esto es tres veces la diagonal del móvil rectangular). En Lineas azules y rojas se muestran los caminos obtenidos por el planeador, mientras que en lineas negras se muestra el resultado de la optimización

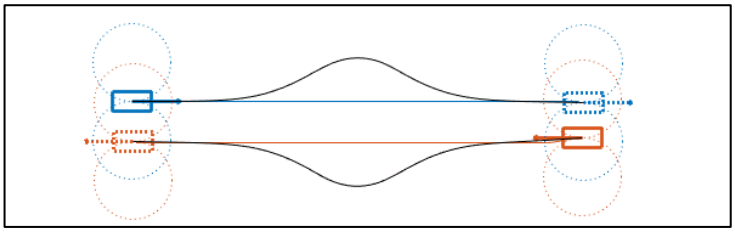


Figura 5.17: Efecto de optimización en la evasión de colisiones con otros móviles [Elaboración propia].

Vistos los ejemplos anteriores, se confirma que la optimizacón de la trayectoria se comporta de la manera esperada. Sin embargo, nótese que la optimización no incluye restricciones del movimiento cinemático o circular del móvil, por lo que en ocasiones puede ocurrir que secciones del camino sean no realizables por el móvil debido a cambios abruptos en la curvatura del mismo.

b) Tiempo de Optimización

Se prueba el generador de trayectoria con el fin de verificar su funcionamiento y el tiempo de optimización. Para ello, se utiliza el mapa con obstáculos rectangulares, se realiza el planeamiento, seguido de la optimización, ver Figura 5.18.

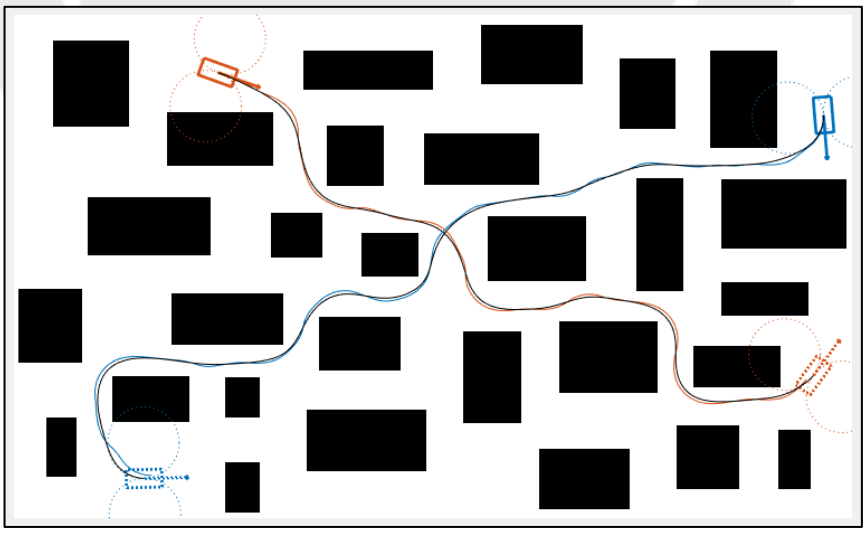


Figura 5.18: Optimización de trayectoria de móviles en mapa de obstáculos rectangulares [Elaboración propia].

En Figura 5.19, se muestra los resultados del planeamiento, los tiempos de planeamiento y optimización de Figura 5.18, así como también, el valor de la función de costo de cada uno de los móviles por cada iteración del algoritmo.

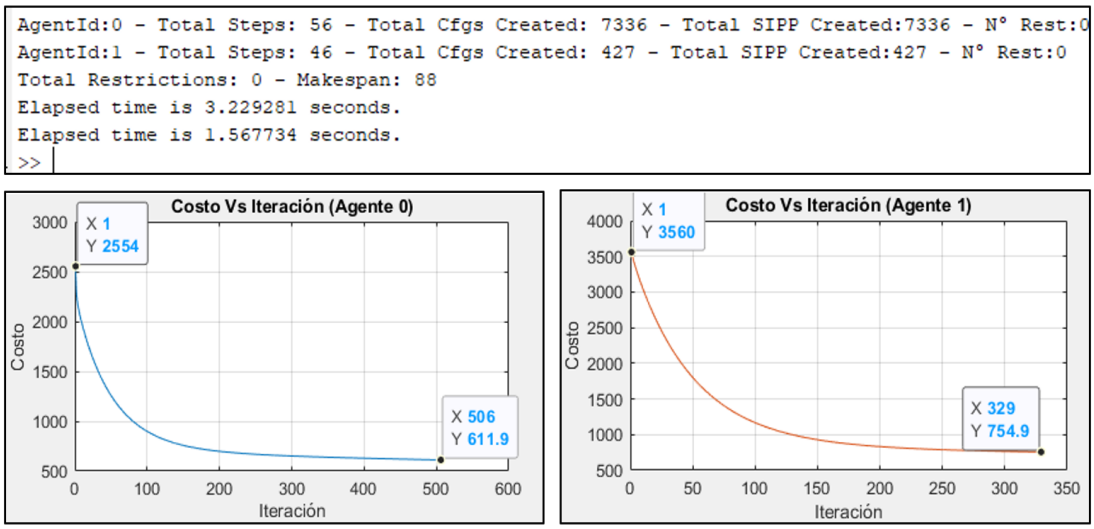


Figura 5.19: Resultados del planeamiento y optimización de trayectoria [Elaboración propia].

De esta, se observa que la optimización tomó la mitad del tiempo que tomó el planeamiento (1.57 segundos respecto a 3.22 segundos) y así mismo se observa que para ambos móviles el valor de la función de costo se reduce con respecto a su valor inicial y llega a converger en el tiempo.

Cabe precisar que la optimización utiliza el algoritmo de descenso de gradiente de manera independiente para cada móvil, aunque compartiendo la posición de cada uno durante sus optimizaciones, para la evasión de colisiones. Es decir, se tiene una función de costo por móvil, la cual cesa de manera independiente de las otras al cumplirse el criterio de convergencia o alcanzar un valor mínimo. Por ejemplo, para Figura 5.19, la optimización de la trayectoria del agente 0 cesó debido al criterio de convergencia en la iteración 506 mientras que para el agente 1 cesó en la iteración 329.

Para las pruebas realizadas, el máximo número de iteraciones de la optimización fue fijado en 2000, mientras que la taza de aprendizaje en 10^-2 y el criterio de convergencia en 10^-5. Los resultados de las pruebas realizadas se muestran en la Tabla 8.

Tabla 8: Resultados de pruebas de optimización de trayectoria [Elaboración propia].

N° móviles	Tiempo Plan. (s)	Tiempo Optim. (s)	Límite de Iteraciones	Id de Móvil	Costo Inicial	Costo Final	Iteraciones Efectuadas
1	0.77	0.71	2000	0	2041	770.1	467
2	3.23	1.57	2000	0	2554	611.9	506
				1	3560	754.9	329

3	3.97	2.8	2000	0	2710	645.7	396
				1	4951	602	736
				2	10720	795.2	389
4	1.39	2.85	2000	0	1771	563.8	466
				1	4376	790.9	285
				2	3207	716.7	442
				3	1433	628.1	393
5	11.26	4.38	2000	0	1140	658.3	355
				1	3312	714.8	634
				2	2481	524.4	576
				3	8836	699.1	547
				4	9683	5943	327
6	52.18	5.65	2000	0	9644	6311	245
				1	1202	576.2	576
				2	2807	1012	766
				3	5631	555.4	534
				4	2197	516.5	578
				5	10090	7317	178

De la Tabla 8, se puede notar que, en todos los casos, las optimizaciones cesaron antes de alcanzar el máximo número de iteraciones, así mismo, se puede notar que el valor de la función de costo al finalizar la optimización no es cercano a cero. Esto se debe a que la optimización minimiza la longitud de la trayectoria y dado que la longitud de esta nunca llegará a cero, la función de costo se mantendrá elevada.

Así mismo, se puede notar que la optimización de las trayectorias incrementa de manera lineal, conforme se aumenta el número de móviles, ver Figura 5.20.

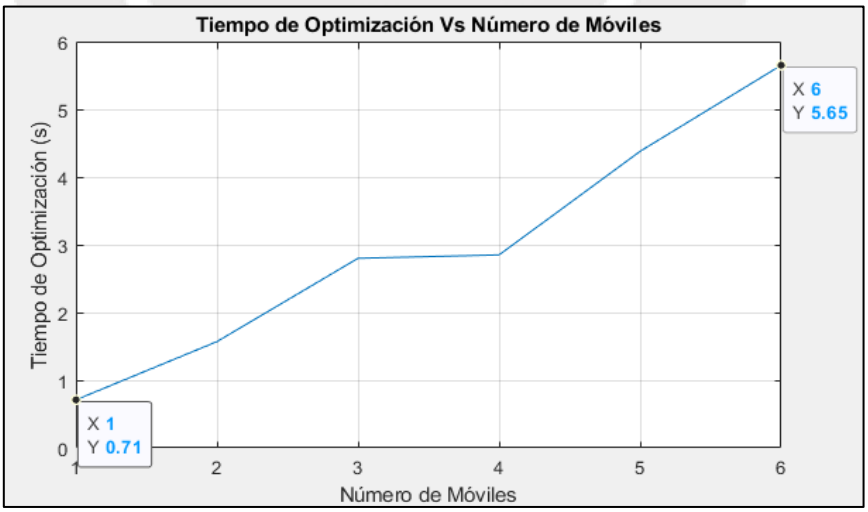


Figura 5.20: Tiempo de optimización vs número de vehículos [Elaboración propia].

Teniendo como mínimo un tiempo de 0.71 segundos para la optimización de la trayectoria de un móvil.

5.3 Simulación del Controlador

5.3.1 Visualización de Seguimiento de Trayectoria

En la simulación del control de seguimiento de trayectoria mostrada en Figura 5.21, se muestra en línea negra la trayectoria de referencia a seguir y en línea verde la trayectoria del móvil luego de aplicar el controlador LQR.

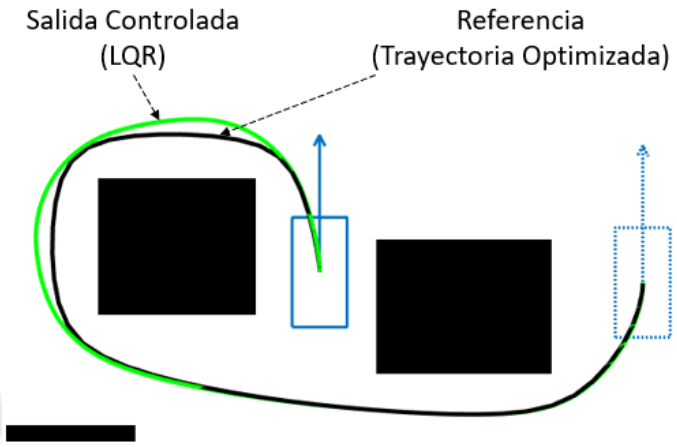


Figura 5.21: Visualización de simulación del control [Elaboración propia].

Mientras que en Figura 5.22, se muestra la simulación del seguimiento de la trayectoria de referencia considerando las dimensiones del móvil.

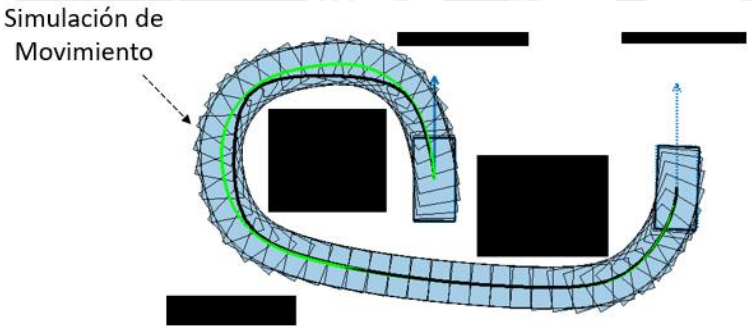


Figura 5.22: Visualización de simulación del movimiento del móvil [Elaboración propia].

5.3.2 Pruebas del Controlador

Las pruebas del controlador fueron realizadas con móviles de dimensiones de 3.2 x1.6m esto en pixeles es 32x16 pixeles y el tiempo de muestreo fue fijado en 0.01 segundos. Los móviles fueron limitados a un rango de velocidad de -2 m/s a 2m/s, un rango de aceleración de -2m^2 a 2 m/s^2 y un rango de giro del timón que va de -45 a 45 grados.

a) Control de un Móvil en Mapa con Obstáculos

Para realizar esta prueba, se utiliza el mapa de obstáculos rectangulares, ver Figura 5.23. Se pretende llevar al móvil de la configuración origen [130, 395, pi/4] en el instante 0 a la configuración destino [713, 91.09, -pi/2] en el instante 102.4.

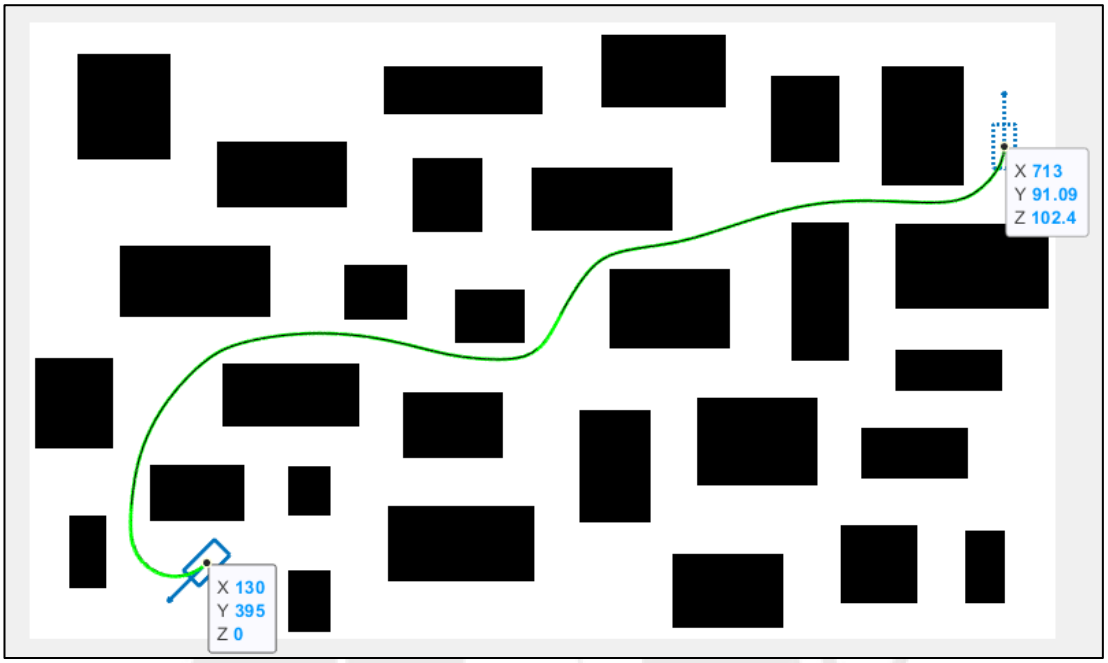


Figura 5.23: Control de seguimiento de trayectoria de móvil en mapa de obstáculos rectangulares [Elaboración propia].

Como se puede observar en Figura 5.23, la trayectoria controlada (en línea verde) se superpone a la trayectoria de referencia en (línea negra), esto mismo, se puede observar en el gráfico de posición X vs tiempo, en Figura 5.24 y en el gráfico de posición Y vs Tiempo, en Figura 5.25.

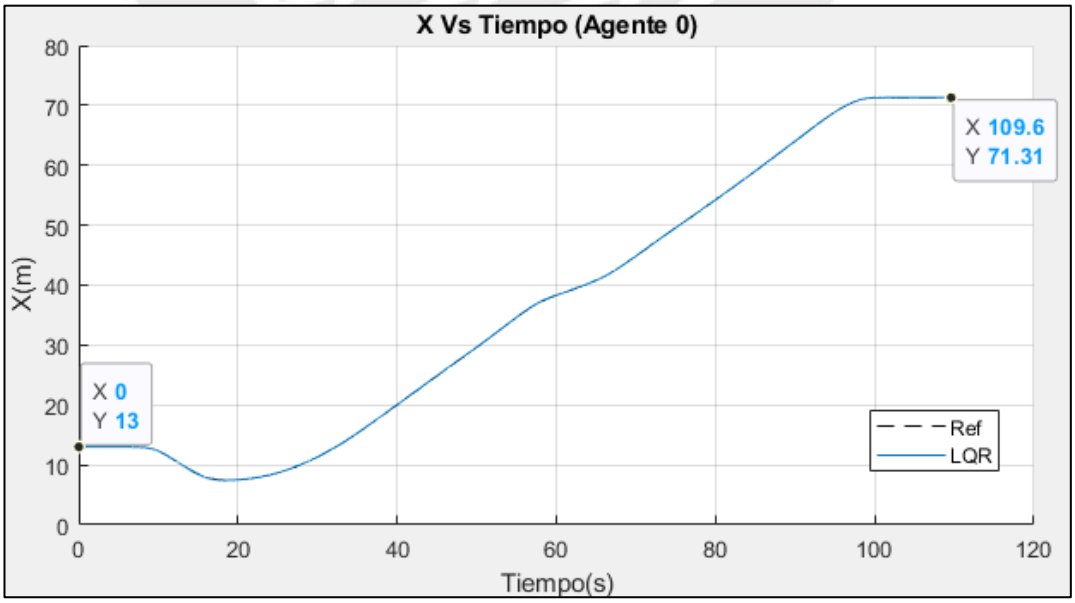


Figura 5.24: Gráfico de posición X vs tiempo [Elaboración propia].

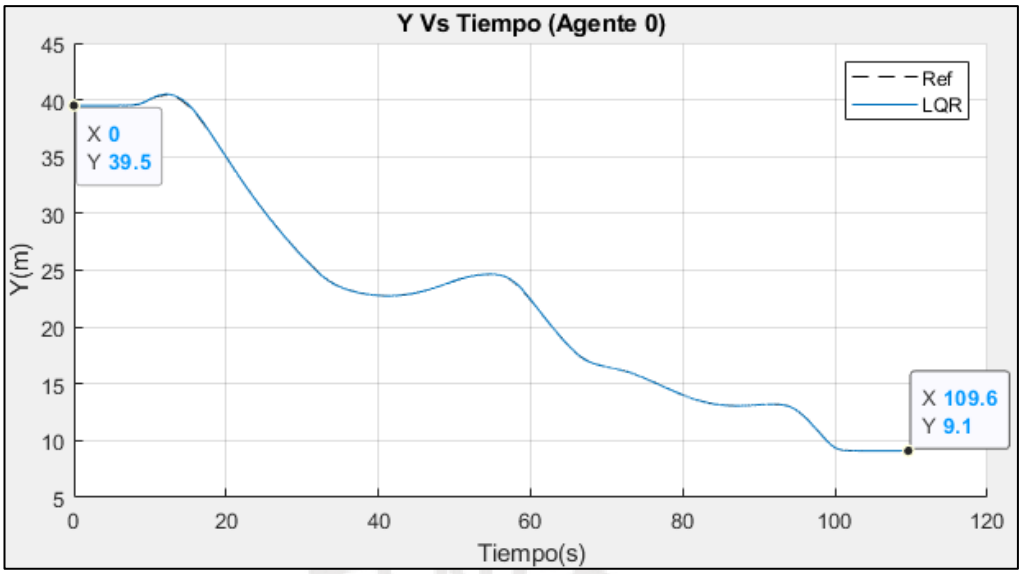


Figura 5.25: Gráfico de posición Y vs tiempo [Elaboración propia].

Como se puede observar en Figura 5.24 y Figura 5.25, la posición del móvil controlada por el controlador LQR se superpone a la trayectoria de referencia, por lo que se puede afirmar que el controlador sigue la posición del móvil de manera óptima.

Por otro lado, en Figura 5.26, se muestra la orientación del móvil, obtenida de manera aproximada de la trayectoria de referencia, y la orientación del mismo luego de que el controlador LQR siga la trayectoria deseada

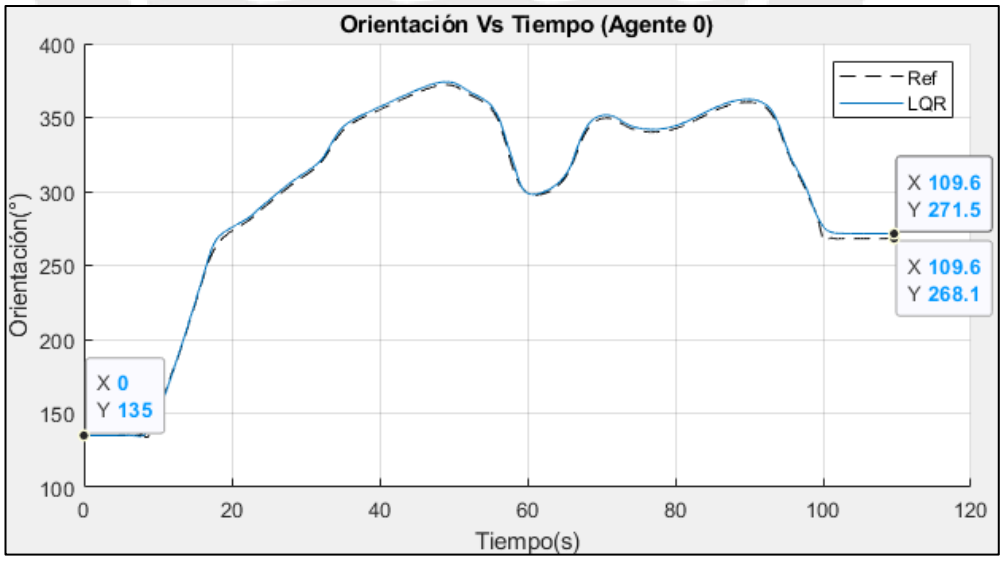


Figura 5.26: Gráfico de orientación vs tiempo [Elaboración propia].

Como se puede observar, la orientación es seguida a través del tiempo, presentado un error en estado estable de 3.4 grados (271.5 - 268.1).

Mientras que en Figura 5.27, se muestra la velocidad longitudinal de referencia a seguir y la obtenida por el controlador LQR.

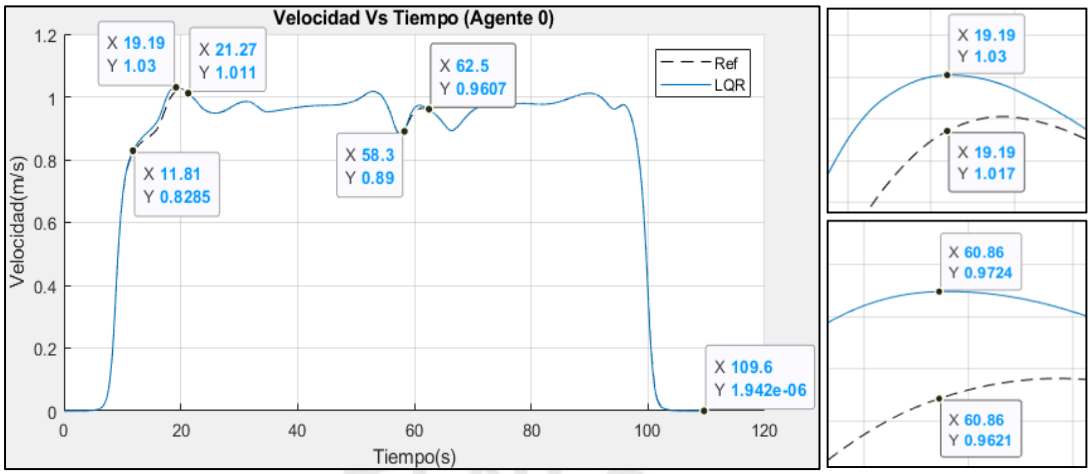


Figura 5.27: Gráfico de velocidad vs tiempo [Elaboración propia].

En esta se puede observar que la velocidad obtenida por el controlador, se superpone a la velocidad de referencia en la mayor reparte del tiempo, excepto entre los instantes 11.81 a 21.27, donde se presenta una desviación de velocidad de alrededor de 0.013 m/s (1.017 - 1.03) y entre los instantes 58.3 a 62.5, donde se presenta una desviación de velocidad de alrededor de 0.01 m/s (0.972-0.962). Estos errores ocurren cuando la optimización modifica al camino original formando curvas con un radio de giro menor al realizable por el móvil. Para evitar dicho error se excluye de la optimización las secciones del camino que presentan curvas pronunciadas, tal como se realiza en [37].

Así mismo, en Figura 5.28, se muestra la aceleración de entrada al móvil en el tiempo.

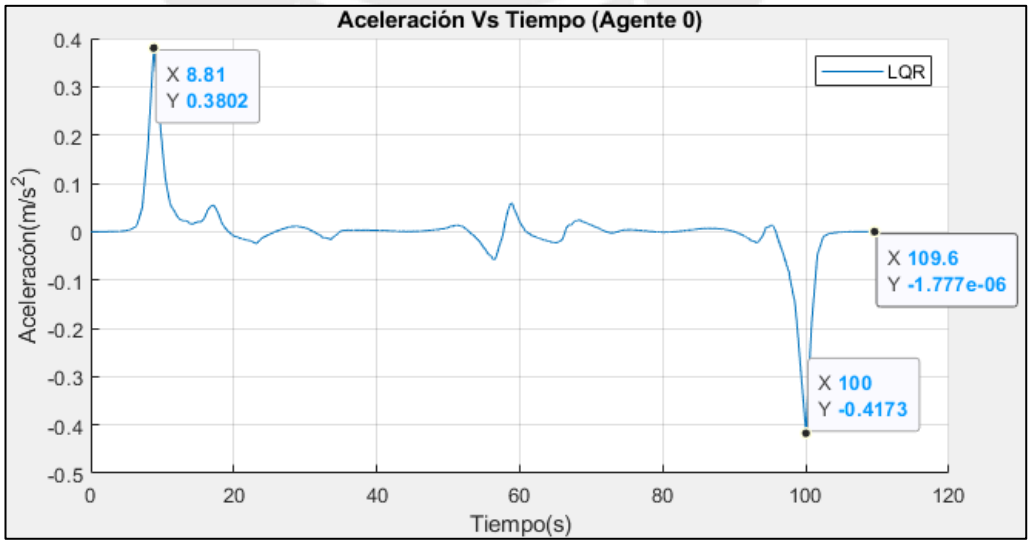


Figura 5.28: Gráfico de aceleración vs tiempo [Elaboración propia].

En esta, se puede notar que la aceleración varía entre 0.38 m/s^2 a -0.42 m/s^2, manteniéndose cercana a cero la mayor parte del tiempo. Los picos de aceleración presentes, corresponden a los instantes de arranque y frenado del móvil. Estos picos pueden ser reducidos incluyendo un perfil de velocidad en el planeamiento (esto a trabajos futuros), para así tener una aceleración y trayectoria más suaves.

Por otro lado, en Figura 5.29, se muestra el ángulo de entrada del giro del timón (ángulo de steering) en el tiempo.

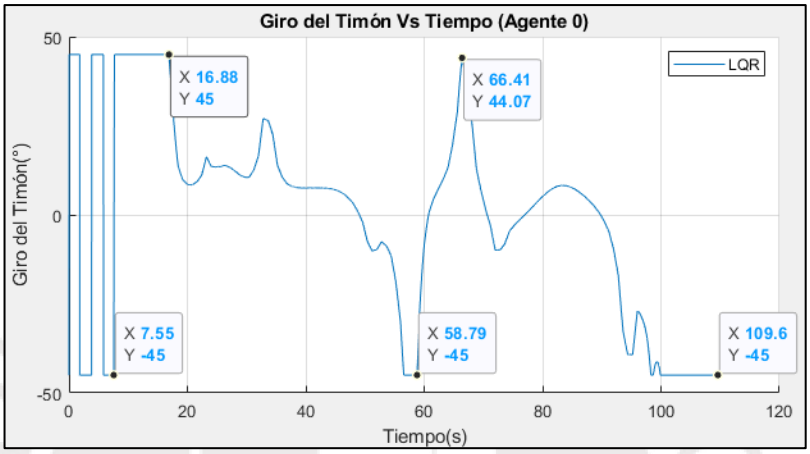


Figura 5.29: Gráfico de giro del timón vs tiempo [Elaboración propia].

De esta se puede notar, que el giro del timón fluctúa entre -45 a 45 grados, presentando cambios abruptos de -45 a 45 entre los instantes 0 a 7.5 (cuando el móvil se encuentra quieto). Dichos cambios abruptos se deben a que el modelo flatness diferencial linealizado presenta una singularidad en el cálculo del ángulo de steering cuando la velocidad del móvil es igual a cero.

La simulación del móvil en movimiento se muestra en Figura 5.30.

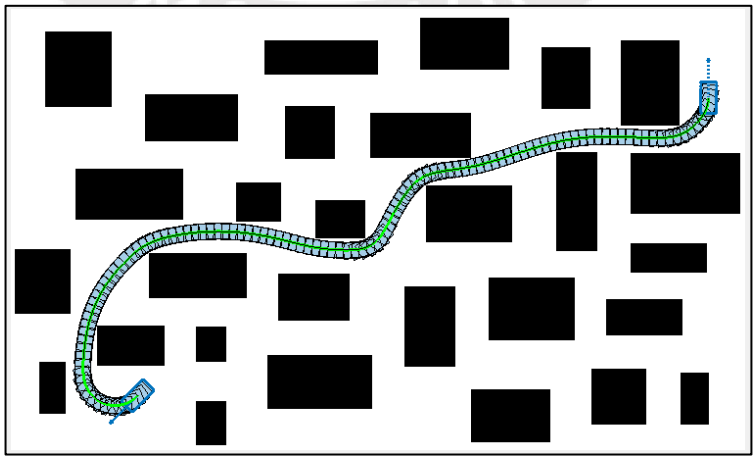


Figura 5.30: Simulación del movimiento del móvil en mapa de obstáculos rectangulares [Elaboración propia].

b) Control de Móviles con puntos de Espera

Para verificar que el controlador permite a los móviles frenar en un punto de espera y reanudar su marcha, se fuerza a un móvil esperar en un punto del mapa usando un mapa tipo H y una situación de pasillo ocupado, ver Figura 5.31.

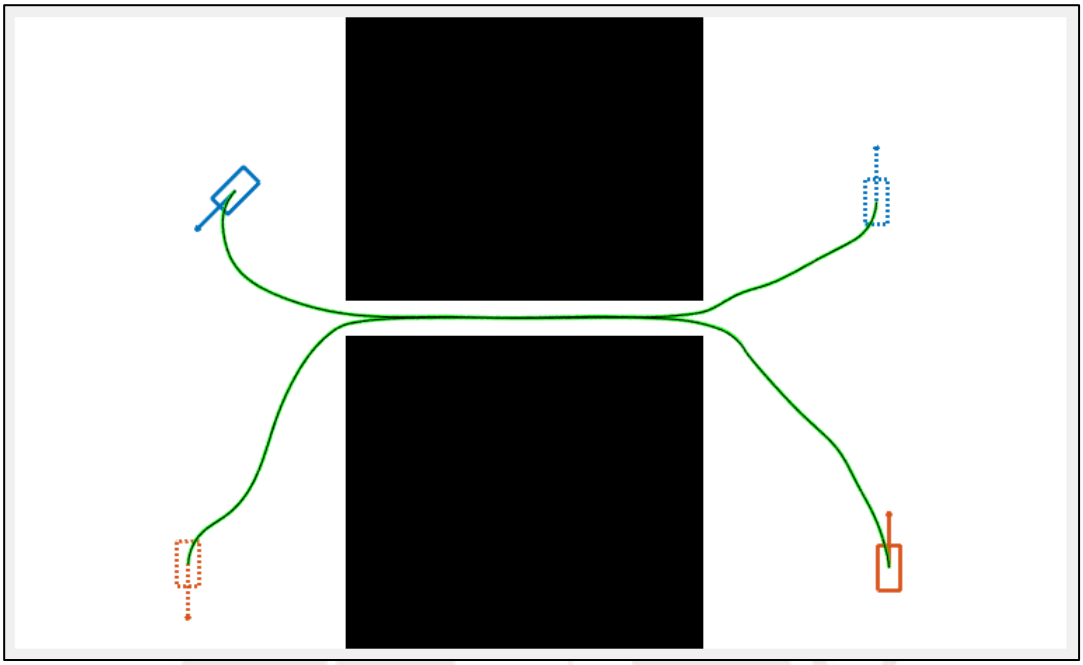


Figura 5.31: Seguimiento de trayectoria en mapa tipo H con situación de pasillo ocupado [Elaboración propia].

Los resultados del planeamiento de los móviles de Figura 5.31 y los tiempos de solución del planeamiento, optimización y control se muestran en Figura 5.32.

```
AgentId:0 - Total Steps: 36 - Total Cfgs Created: 7850 - Total SIPP Created:22375 - N° Rest:0
AgentId:1 - Total Steps: 45 - Total Cfgs Created: 15775 - Total SIPP Created:52729 - N° Rest:9
Total Restrictions: 9 - Makespan: 94.4
Elapsed time is 6.508973 seconds.
Elapsed time is 1.552559 seconds.
Elapsed time is 0.174882 seconds.
>>
```

Figura 5.32: Resultados del planeamiento de móviles en mapa tipo H [Elaboración propia].

De Figura 5.33, se puede notar que el controlador del agente 1 (móvil de color naranja) sigue la trayectoria de referencia de manera óptima, presentando un error de orientación en estado estable de 2.1 grados

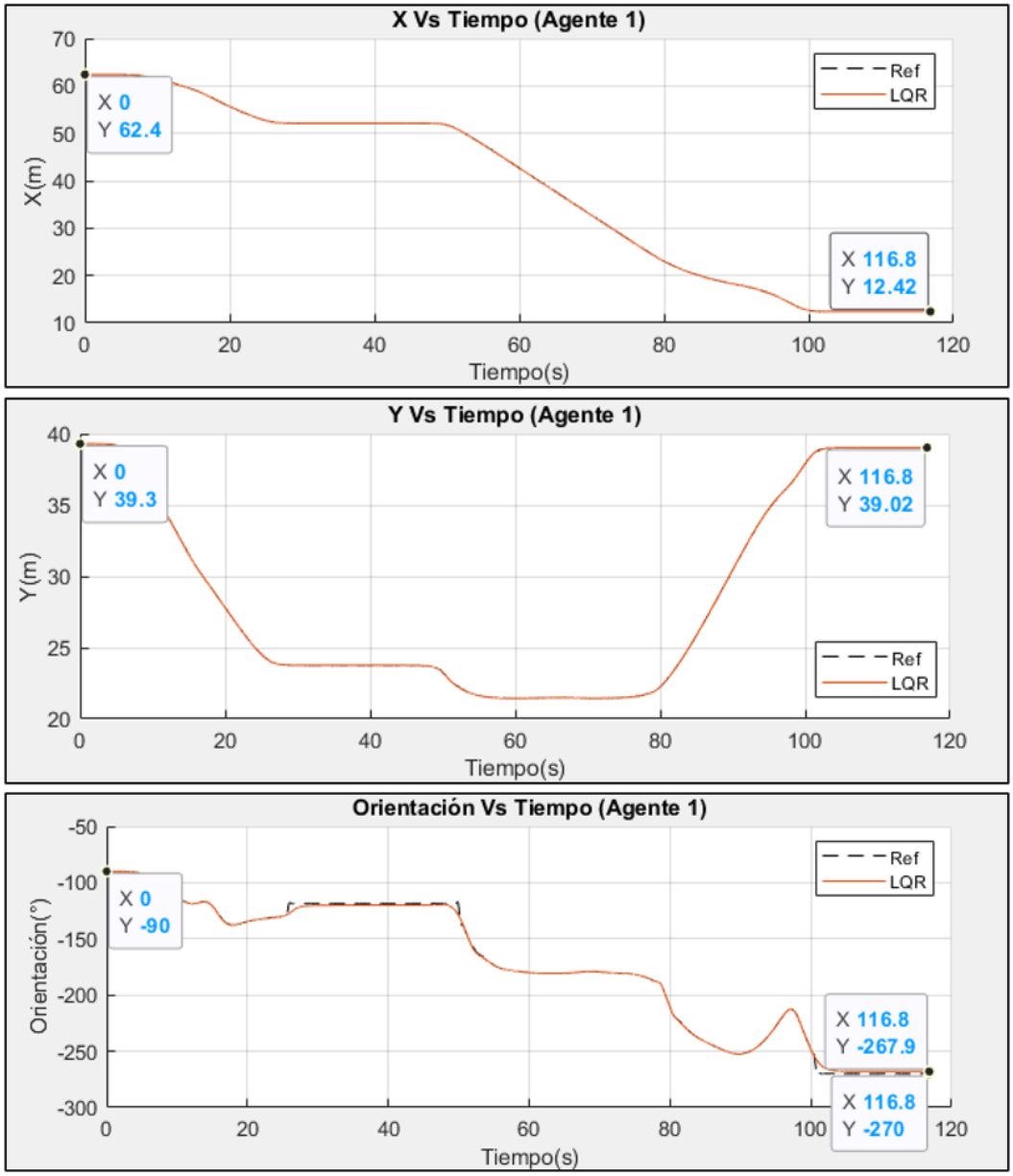


Figura 5.33: Gráficas de posición y orientación vs tiempo del control del agente 1 en mapa tipo H [Elaboración propia].

Así mismo, de la figura se puede observar que el móvil mantiene su posición entre los instantes 25 a 50; es decir se queda esperando en un punto de espera hasta que el otro móvil deje libre el pasillo del mapa tipo H.

Mientras que, en Figura 5.34, se muestra que la velocidad del móvil se reduce de manera suave a cero al llegar al punto de espera, y luego de la espera, reanuda su marcha suavemente.

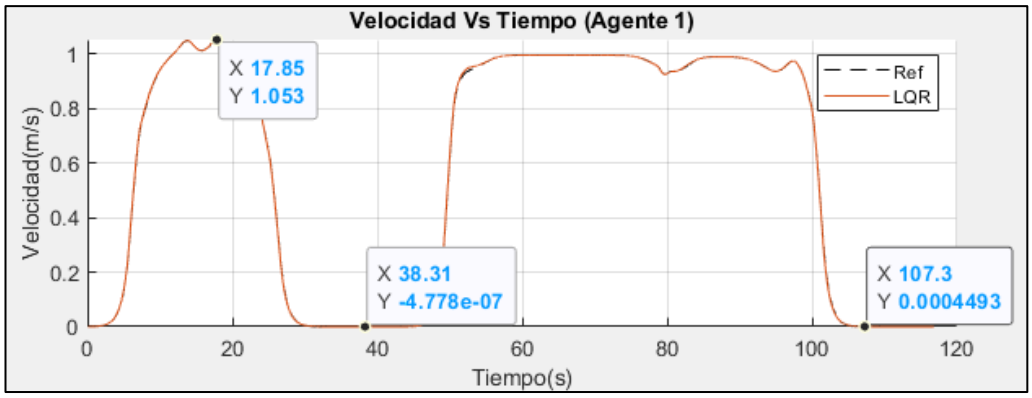


Figura 5.34: Gráfica de velocidad vs tiempo del control del agente 1 en mapa tipo H [Elaboración propia].

Por otro lado, en Figura 5.35, se muestra las entradas de control del móvil, estas son la aceleración y ángulo del timón.

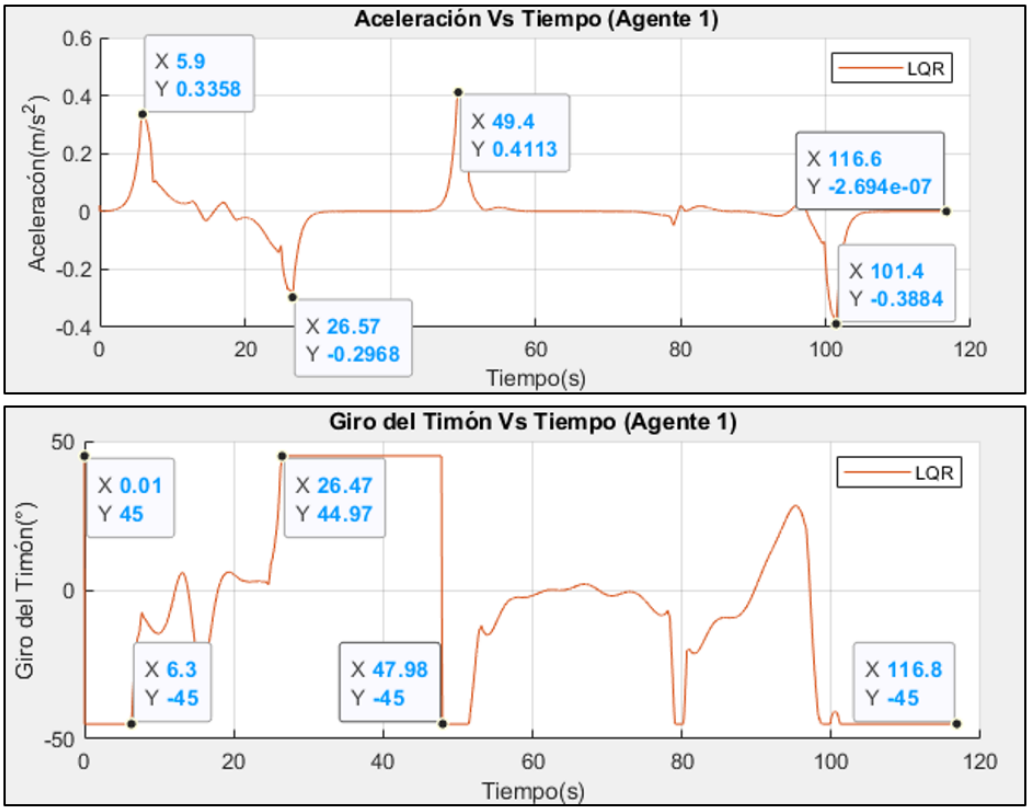


Figura 5.35: Gráficas de aceleración y giro del timón vs tiempo del control del agente 1 en mapa tipo H [Elaboración propia].

De estas se puede observar que la aceleración durante el frenado en el punto de espera, llega a -0.29 m/s^2 y en la reanudación de su marcha llega a 0.4113 m/s^2, manteniéndose cercana a cero durante el movimiento del móvil. Por otro lado, el ángulo de giro del timón cambia de manera abrupta en el instante del frenado y en la reanudación de la marcha, esto se debe a que el controlador LQR del sistema linealizado presenta una singularidad cuando la velocidad es igual a cero.

Los resultados del control del agente 0 (el móvil azul), se muestra en Figura 5.36.

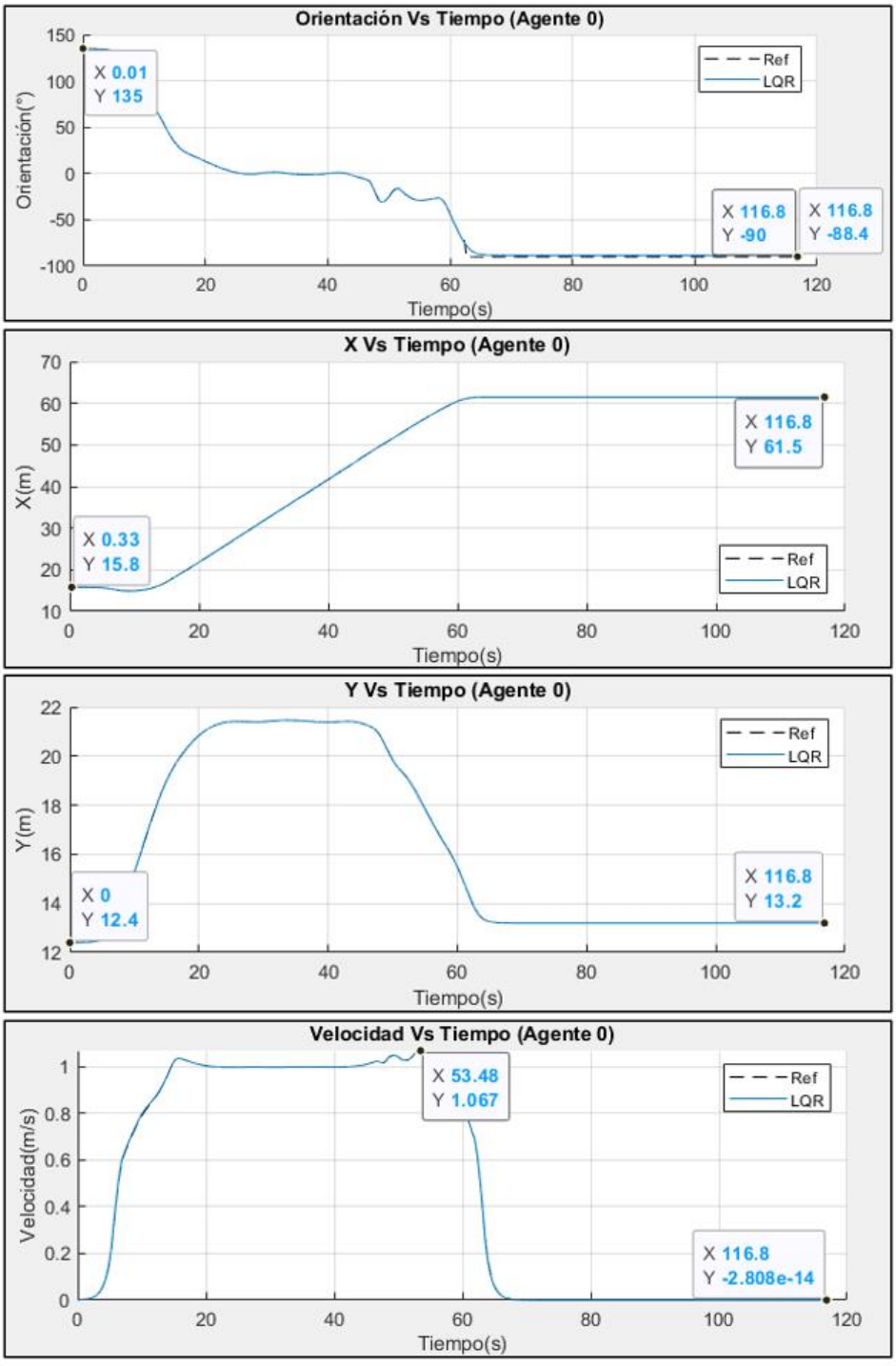


Figura 5.36: Gráficas de posición, orientación y velocidad vs tiempo del control del agente 0 en mapa tipo H [Elaboración propia].

De estas, se pude notar que el móvil mantiene la marcha hasta llegar a su destino, sin parar por algún punto de espera, así mismo se puede notar que el controlador sigue la trayectoria de referencia de manera óptima, presentando un error en estado estable de 1.6 grados en la orientación.

Mientras que las entradas de control del agente 0 se muestran en Figura 5.37.

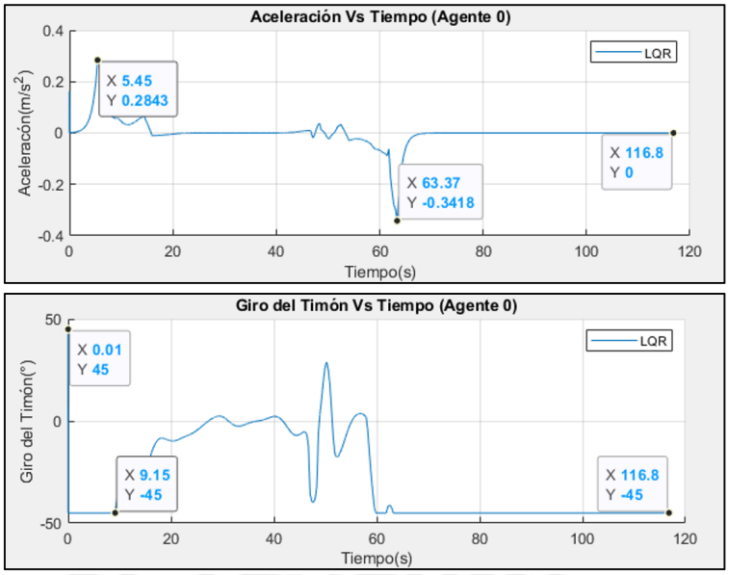


Figura 5.37: Gráficas de aceleración y giro del timón vs tiempo del control del agente 0 en mapa tipo H [Elaboración propia].

De estas, se puede observar que la aceleración llega a un máximo de 0.23 m/s^2 y a un mínimo de -0.34 m/s^2 durante el arranque inicial del móvil y en el frenado al llegar a su destino, manteniendo su valor cercano a cero durante la marcha. Así mismo, se observa un cambio suave del ángulo de giro del móvil durante la marcha y un cambio abrupto en el instante 0.

Finalmente, en Figura 5.38, se muestra la simulación del movimiento de los móviles en cuatro instantes de tiempo distintos

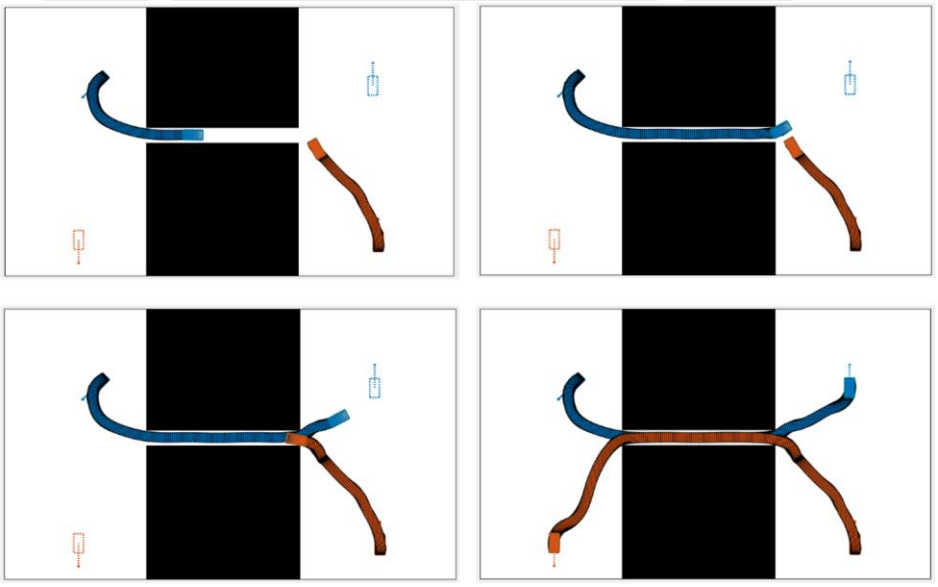


Figura 5.38: Simulación del movimiento de los móviles en mapa tipo H [Elaboración propia].

5.4 Simulación del Sistema Integrado

Para las siguientes pruebas, se utilizó móviles de dimensiones de 20x16 pixeles (equivalente a 2x1.6 metros) y mapas de 350x750 pixeles. Donde los móviles se mueven a una velocidad de 1 m/s con una aceleración máxima de 2m/s^2.

a) Mapas espaciados

En Figura 5.39, se muestra el planeamiento de caminos (en colores variados), la optimización de trayectoria (en negro) y el seguimiento de trayectoria (en verde) de 10 móviles en un espacio vacío. Mientras que en Figura 5.40 se muestra el error de seguimiento de trayectoria de cada uno de los móviles.

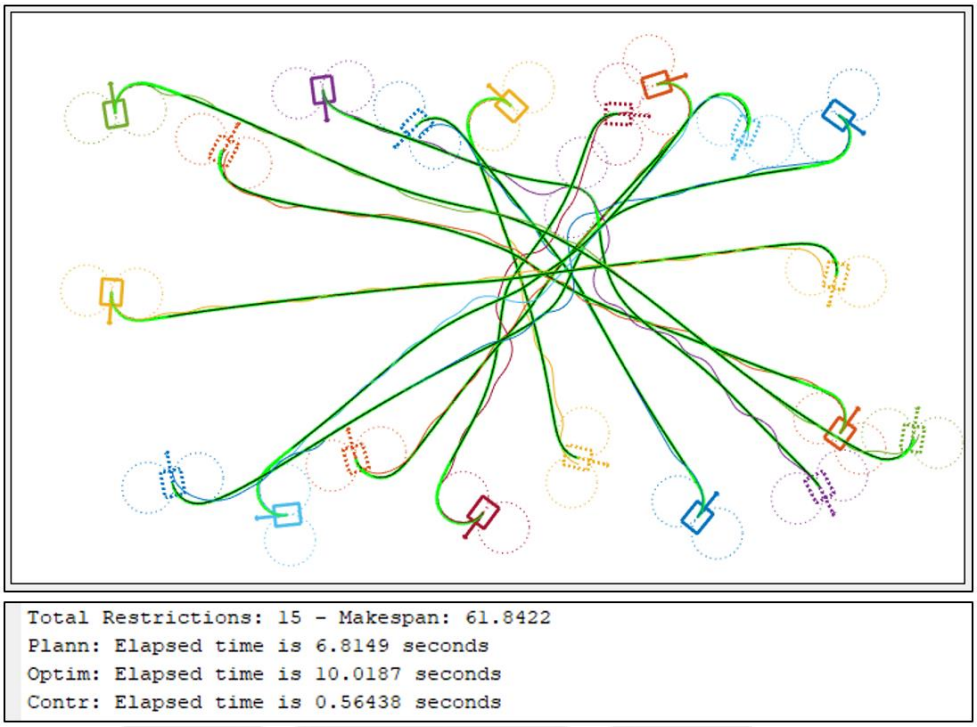


Figura 5.39: Seguimiento de trayectoria de 10 móviles en espacio vacío [Elaboración propia].

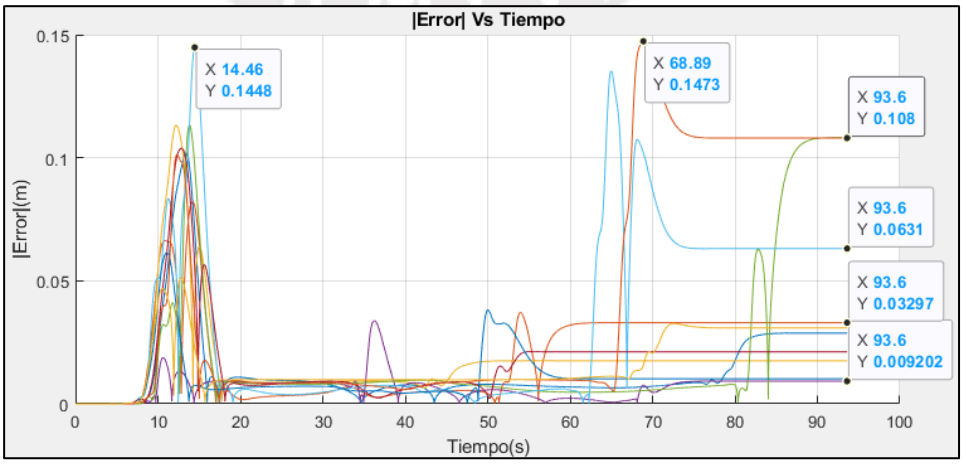
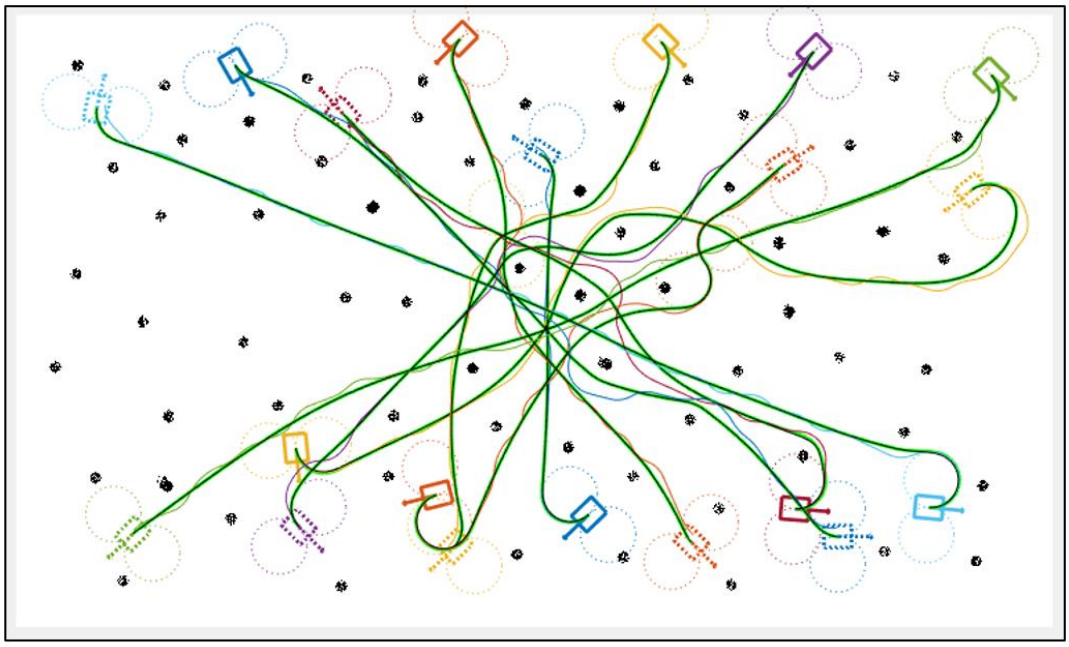


Figura 5.40: Error de seguimiento de trayectoria Vs tiempo [Elaboración propia].

De la primera gráfica se puede observar que la salida de control (línea verde) se superpone a la trayectoria de referencia (línea negra) para todos los móviles; por lo que se puede afirmar que el controlador sigue las trayectorias de todos los móviles. Mientras que en la segunda gráfica se observa que las trayectorias son seguidas con un error máximo de 0.15 m (para un vehículo de 1.6 m de ancho) y con un error en estado estable máximo de 0.11 m, siendo este menor a 0.32 m para 7 de los 10 móviles. Así mismo, de la primera gráfica se puede observar que el planeamiento de los 10 móviles tomó 6.81 segundos, mientras que la optimización de trayectoria tomó 10.02 segundos y el control de todos los móviles tan solo 0.56 segundos. Es decir, tomó un total de 17.39 segundos

De igual manera, se realiza el planeamiento, la optimización de trayectoria y el control de seguimiento de trayectoria para 10 móviles en el mapa con obstáculos de puntos dispersos, tal como se muestra en Figura 5.41. En esta, se puede observar, que se sigue la trayectoria de todos los móviles, pues la línea verde se superpone a la línea negra en todos los casos.



```
Total Restrictions: 35 - Makespan: 65.3368
Plann: Elapsed time is 12.3355 seconds
Optim: Elapsed time is 11.9693 seconds
Contr: Elapsed time is 0.46349 seconds
```

Figura 5.41: Seguimiento de trayectoria de 10 móviles en mapa de puntos dispersos [Elaboración propia].

Así mismo, se realiza el control de seguimiento de trayectoria para 10 móviles en el mapa con obstáculos de barras, tal como se muestra en Figura 5.42.

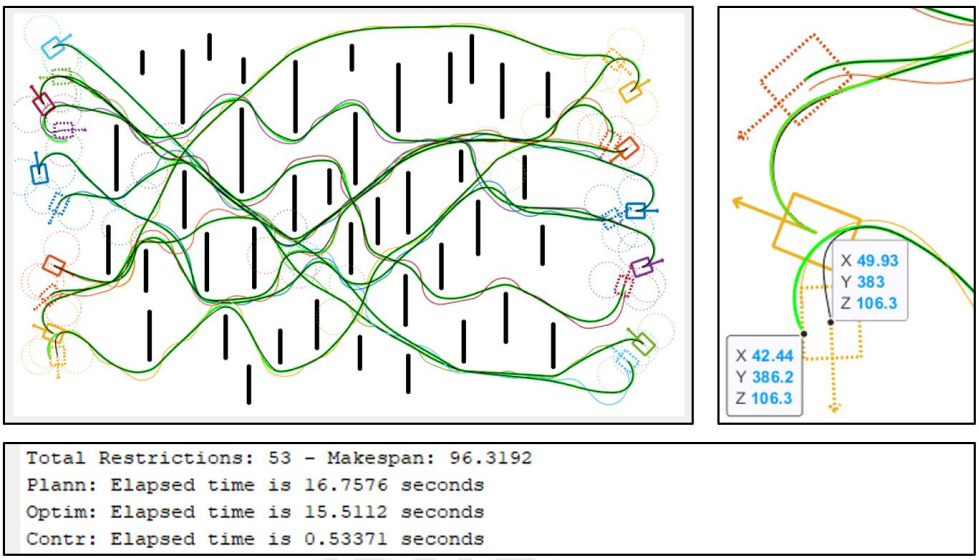


Figura 5.42: Seguimiento de trayectoria de 15 móviles en mapa de barras [Elaboración propia].

En esta, se puede notar que se sigue la trayectoria de referencia en todos casos. Sin embargo, en el punto destino del móvil amarillo de la esquina inferior izquierda, se presenta un error de seguimiento de 0.815 metros (8.15 unidades del mapa), el cual equivale a la mitad del ancho del móvil (1.6 metros). Donde el error es causado por la pronunciada curvatura de la trayectoria de referencia.

b) Mapas Estrechos

El problema del error en el seguimiento de trayectoria debido a la curvatura es recurrente en mapas con poco espacio para la navegación. En Figura 5.43, se muestra el planeamiento, optimización y el seguimiento de trayectoria de 10 móviles en el mapa de obstáculos redondos.

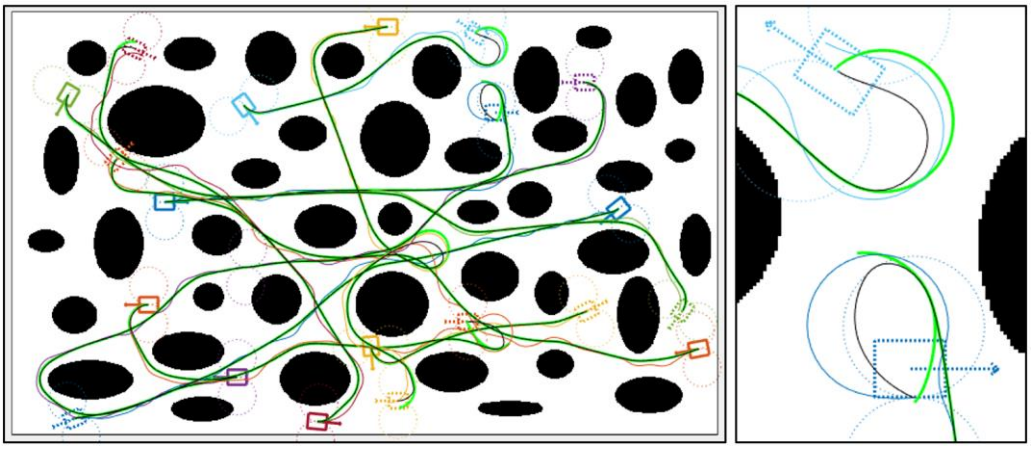


Figura 5.43: Error de seguimiento de trayectoria en mapa de obstáculos redondos [Elaboración propia].

Como se puede observar en la figura, la trayectoria optimizada (en línea negra) presenta curvaturas muy pronunciadas, que distan mucho del camino obtenido por

el planeamiento (líneas azul y celeste para los dos móviles de la figura). Estas curvaturas ocasionan que el controlador pierda la señal de referencia y si bien los móviles terminan en la posición esperada (pues el final de la curva verde es igual al punto destino) no terminan con la orientación correcta y no se garantiza que no colisionen con otros móviles o con los obstáculos del entorno.

Para solucionar este problema, se excluye de la optimización a todos los puntos que poseen una curvatura cercana a la curvatura máxima k_{max} de giro del móvil. Donde la curvatura máxima se calcula como la inversa del radio mínimo.

$$k_{max} = \frac{1}{r_{min}} \quad (230)$$

Usando un factor de 0.8 para representar la proximidad a la curvatura máxima, se tiene lo siguiente:

$$Curvatura(X_i) = \frac{|x'' * y' - x' * y''|}{((x')^2 + (y')^2)^{3/2}} \quad (231)$$

$$\nabla_{\mathbb{X}_i} = \bar{0}, \qquad Curvatura(X_i) > 0.8 * k_{max} \quad (232)$$

Notar que aplicar exclusión por curvatura conlleva a que no siempre se pueda corregir la posición final del móvil obtenida por planeamiento, ya que, si antes de llegar al punto destino se realiza un giro pronunciado, se excluiría de la minimización a todos los puntos cercanos al punto final, impidiendo mover al punto final en sí.

En Figura 5.44, se muestra el resultado de aplicar la exclusión de puntos con curvas pronunciada de la optimización.

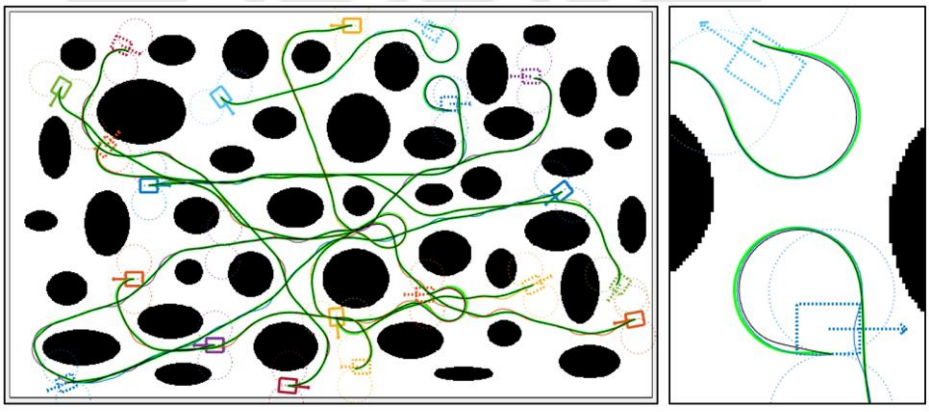


Figura 5.44: Seguimiento de trayectoria con exclusión de puntos de curvatura máxima en mapa de obstáculos redondos [Elaboración propia].

Como se puede observar, se corrige completamente el seguimiento de trayectoria, llevando a los móviles hasta el punto final hallado por el planeador.

Como se mencionó anteriormente, solo aquellos móviles que no posean curvaturas pronunciadas cerca al punto final, pueden ser llevadas hasta el punto destino ideal, ver Figura 5.45.

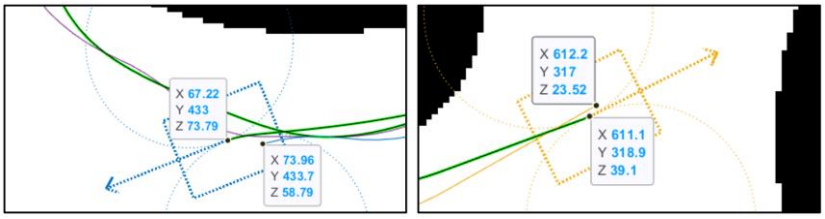


Figura 5.45: Corrección de posición del punto final en móviles sin curvaturas pronunciadas cerca al punto final [Elaboración propia].

Luego, se aplica esto mismo para el planeamiento del resto de mapas.

En Figura 5.46, se muestra el seguimiento de trayectoria de 10 móviles en el mapa de obstáculos no convexos y los tiempos que tomaron el planeamiento, optimización y control de los móviles.

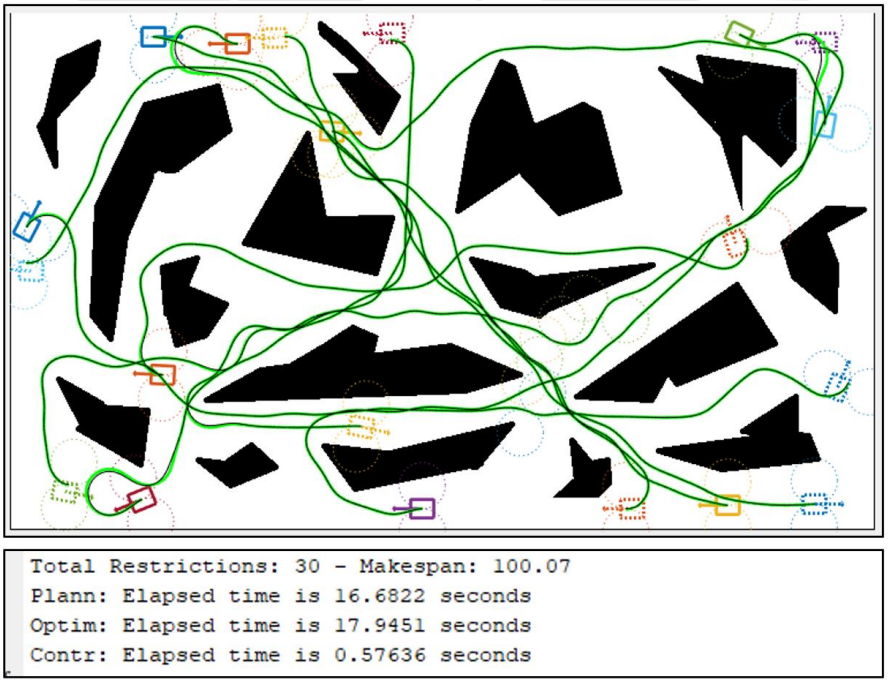
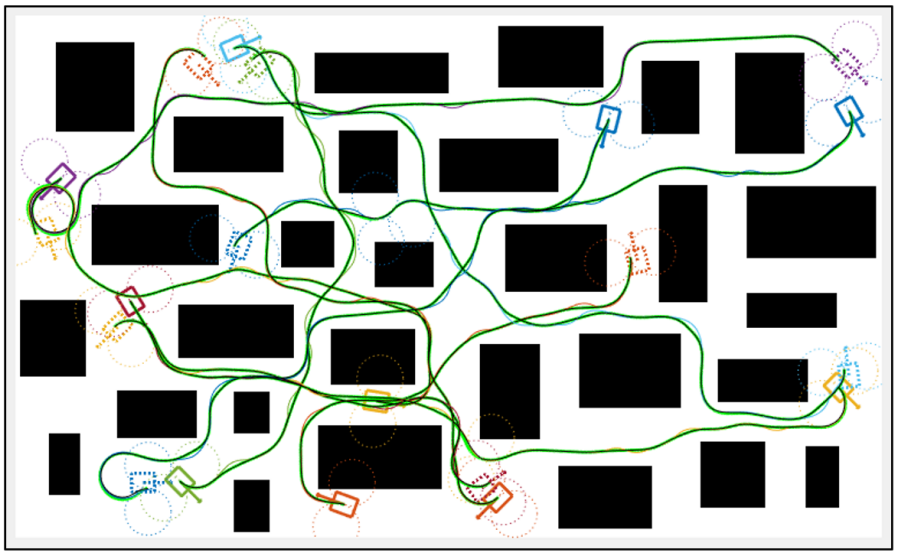


Figura 5.46: Seguimiento de trayectoria de 10 móviles en mapa de obstáculos no convexos [Elaboración propia].

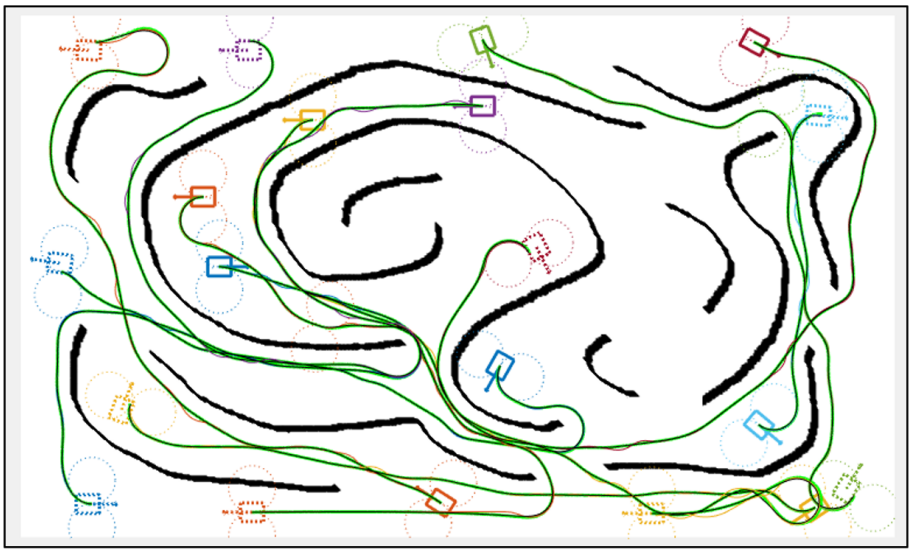
De igual manera, en Figura 5.47, se muestra el seguimiento de trayectoria de 10 móviles en el mapa de obstáculos rectangulares.



```
Total Restrictions: 23 - Makespan: 82.6049
Plann: Elapsed time is 39.8196 seconds
Optim: Elapsed time is 14.1424 seconds
Contr: Elapsed time is 0.44844 seconds
```

Figura 5.47: Seguimiento de trayectoria de 10 móviles en mapa de obstáculos rectangulares [Elaboración propia].

Mientras que, en Figura 5.48, se muestra el seguimiento de trayectoria de 10 móviles en el mapa de estructuras curvas.



```
Total Restrictions: 8 - Makespan: 93.5912
Plann: Elapsed time is 22.4442 seconds
Optim: Elapsed time is 17.3656 seconds
Contr: Elapsed time is 0.54408 seconds
```

Figura 5.48: Seguimiento de trayectoria de 10 móviles en mapa de estructuras curvas [Elaboración propia].

Y, en Figura 5.49, se muestra el seguimiento de trayectoria de 8 móviles en el mapa de estructuras rectas.

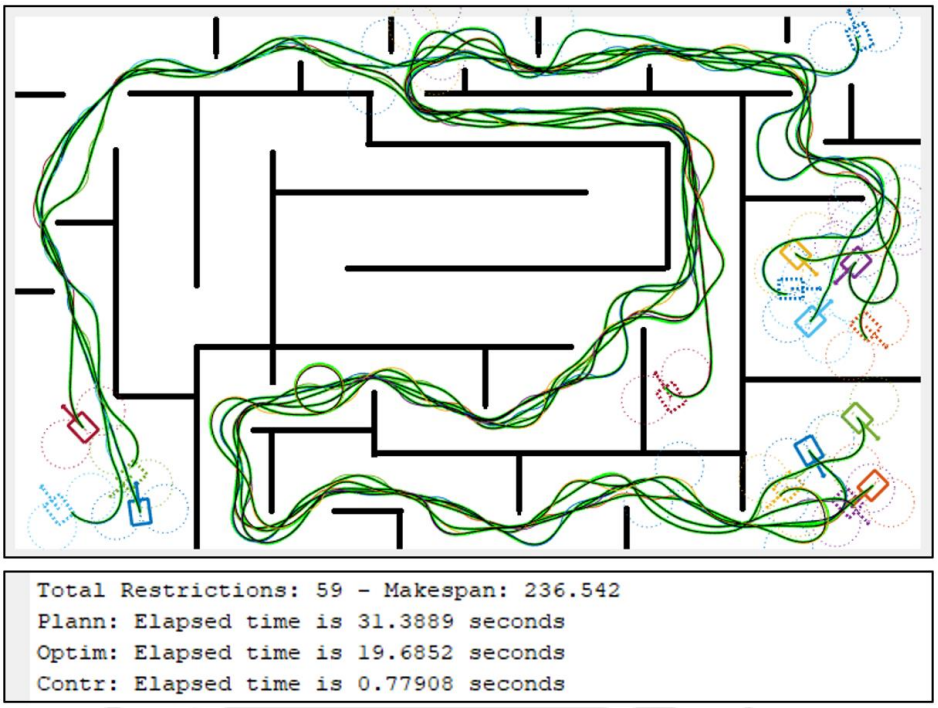


Figura 5.49: Seguimiento de trayectoria de 8 móviles en mapa de estructuras rectas [Elaboración propia].

La Tabla 9 muestra el resumen de las pruebas realizadas para el sistema integrado.

Tabla 9: Resumen de pruebas realizadas del sistema integrado [Elaboración propia].

N° de móviles	Tipo de Mapa	N° de Restr.	Tiempo de Plan. (s)	Tiempo de Optim. (s)	Tiempo de Contr. (s)	Exclusión por Curvatura
10	Vacío (450x750)	15	6.81	10.02	0.56	No
10	Puntos (450x750)	35	12.34	11.97	0.46	No
10	Barras (450x750)	53	16.76	15.51	0.53	No
10	Redondo (450x750)	10	25.40	13.91	0.62	Sí
10	Rectangular (450x750)	23	39.81	14.15	0.45	Sí
10	No Convexo (450x750)	30	16.68	17.95	0.58	Sí
10	Estructuras Curvas (450x750)	8	22.44	17.36	0.54	Sí
8	Estructuras Rectas (450x750)	59	31.39	19.69	0.78	Sí

De la tabla se puede observar que el tiempo que tomo el control de seguimiento de trayectoria es mucho menor que el de planeamiento y optimización en todos los casos. Así mismo, se observa que, entre los mapas con obstáculos, el mapa de puntos fue el más rápido en resolver, mientras que el de estructuras rectas, fue el de mayor tiempo de solución. En orden de menor a mayor tiempo de solución, se tiene: mapa vacío, obstáculos de puntos, obstáculos de barras, obstáculos no convexos, estructuras curvas, obstáculos redondos, estructuras rectas y obstáculos rectangulares. Es decir, los mapas que tomaron mayor tiempo de solución, son los mapas estrechos (de 16.68 a 39.81 segundos) en comparación a los mapas espaciados (de 6.81 a 16.76 segundos). La razón de ello se debe a que durante la búsqueda de nodos SIPP, el algoritmo tiende a elegir primero aquellos nodos que implican explorar nuevos lugares del mapa en vez de aquellos nodos que implican esperar en puntos del mismo. Y dado que en los mapas estrechos tiende a haber esperas prolongadas, debido a que los móviles bloquean los caminos, es de esperarse que la solución tome mayor tiempo, ya que el algoritmo buscará primero caminos alternativos. Esto se evidencia, si se compara la solución del mapa de estructuras rectas (mapa estrecho) con el mapa de barras (mapa espaciado), pues si bien ambos tienen un número similar de restricciones (59 y 53 respectivamente), la solución del mapa de estructuras rectas toma casi el doble del de barras. Por otro lado, dado que la demora radica en explorar configuraciones, se puede reducir el tiempo de solución si se reduce las configuraciones a explorar, esto se puede lograr reduciendo el número de giros discretizados que utiliza el móvil en el algoritmo Hybrid A* (lo cual generaría menos nodos de configuración hijos y por ende menos nodos SIPP), o alternativamente disminuyendo la discretización del posición y orientación en el algoritmo Hybrid A* (con el fin de que configuraciones similares sean tomadas como una misma configuración y así se reduzca el número de nodos de configuración y de nodos SIPP).

CONCLUSIONES

Se logró el planeamiento local y global de múltiples móviles no holonómicos en mapas de obstáculos variados usando los algoritmos de A estrella híbrido (Hybrid A*), planeamiento de caminos en intervalos seguros (SIPP) y búsqueda basada en conflictos (CBS).

Se verificó el funcionamiento del planeador local para móviles no holonómicos en mapas con diferentes tipos de obstáculos. Lográndose en promedio encontrar el camino solución en 1.08 segundos en un mapa de 450x750 pixeles. Resultados obtenidos en un computador con un CPU Intel Core i7-7700HQ @ 2.8GHz y 32GB de RAM.

Se verificó que el planeador global es capaz de resolver conflictos para pares de móviles ubicados en entornos restringidos propensos al atasco, como mapas tipo H, tipo I o tipo T, logrando el intercambio de posición y de orden en el mapa tipo T, resolviendo cruce de caminos en el mapa tipo I y logrando el planeamiento en un mapa tipo H con un pasillo que solo puede ser ocupado por un móvil a la vez.

Se logró optimizar las trayectorias obtenidas con el planeador global, minimizando la longitud del camino, corrigiendo la posición final del móvil y respetando la evasión de colisiones con otros móviles y la evasión de obstáculos con el entorno. Logrando la convergencia de la optimización en alrededor de 0.71 segundos por móvil en el mapa de obstáculos rectangulares.

Se logró diseñar un controlador para móviles no holonómicos usando el modelo cinemático de un móvil tipo bicicleta, un controlador LQR y linealización por extensión dinámica para sistemas flatness diferencial, el cual es capaz de seguir trayectorias que incluyen frenados y puestas en marcha en puntos de espera en el mapa.

Se verificó el funcionamiento del sistema en conjunto, integrando el planeamiento, la generación de trayectoria y el control de seguimiento de trayectoria, lográndose el control de hasta 8 móviles no holonómicos en el mapa de estructuras rectas y 10 móviles no holonómicos en los mapas formados por obstáculos no convexos, obstáculos de barras, obstáculos de puntos, obstáculos rectangulares, obstáculos redondos y por estructuras curvas.

RECOMENDACIONES

Con el fin de reducir el tiempo del planeamiento, se recomienda utilizar el algoritmo de brushfire [53] para el cálculo offline de la distancia holonómica de cada punto de la grilla. O alternativamente calcular la distancia holonómica usando diagramas de Voronoi [54] u otros algoritmos de planeamiento para grafos.

Así mismo, dado que el planeamiento global es de tipo desacoplado; es decir, el planeamiento del camino de cada móvil se realiza de manera independiente al resto de móviles, se puede realizar el planeamiento de todos los móviles de manera semi simultánea usando múltiples hilos y así reducir el tiempo del cálculo computacional.

Se recomienda incluir movimientos con velocidad negativa en el sistema, para encontrar caminos de menor longitud y acortar los tiempos de solución. Para ello se deberá agregar en el planeamiento movimientos con desplazamiento negativo, usando las mismas ecuaciones del movimiento circular del móvil descritas en este documento. Estos movimientos deberán ser penalizados agregando un costo adicional a los nodos SIPP asociados a estos, para evitar su uso excesivo. Así mismo, será necesario extender el cálculo de los 6 tipos de curvas de Dubins [46] a los 48 tipos de curvas de Reeds and Shepp [55] y utilizar estas para el cálculo de la distancia no holonómica con la configuración destino. Adicionalmente, se deberá excluir de la optimización todos los movimientos en retroceso haciendo nulo el gradiente de la función de costo en todos los puntos de la trayectoria que representan un movimiento en retroceso.

Se recomienda compensar la singularidad del controlador en el cálculo del giro del timón cuando la velocidad es igual o cercana a cero; para ello, se puede utilizar un controlador adicional durante el arranque y el frenado del móvil y combinar la lógica de ambos controladores usando lógica difusa.

De implementarse el sistema, se recomiendo utilizar un filtro de Kalman para estimar la posición del móvil, la cual es necesaria para lograr el control del mismo; esto se puede lograr usando fusión de sensores, estimando la posición del móvil usando la data de encoders ubicadas en las ruedas del mismo y corrigiendo la estimación con la data de un acelerómetro. De igual manera, se puede calcular la orientación del móvil, estimando de la orientación del mismo con la data de un giroscopio y corrigiendo la estimación de la orientación con un magnetómetro.

BIBLIOGRAFÍA

[1] BUSINESS INSIDER
2019 *These are the robots that help you get your Amazon packages on time.* Consulta: 19 de enero de 2021.

https://techcrunch.com/2019/03/17/these-are-the-robots-that-help-you-get-your-amazon-packages-on-time

[2] TECHNODE
2018 *Alibaba launches China's biggest robotic warehouse ahead of Singles Day.* Consulta: 19 de enero de 2021.

https://technode.com/2018/10/30/alibaba-new-robotic-warehouse/

[3] RIOTINTO
2021 *Automation.* Consulta: 19 de enero de 2021.

https://www.riotinto.com/en/about/innovation/automation

[4] DYSON
2021 *Robot Vacuum.* Consulta: 17 de abril de 2021.

https://www.dyson.com/vacuum-cleaners/robot-vacuum

[5] IROBOT
2021 *Roomba Vacuums.* Consulta: 17 de abril de 2021.

https://www.irobot.co.uk/roomba

[6] INTEL
2021 *Drone Light Shows Powered by Intel.* Consulta: 17 de abril de 2021.

https://www.intel.com/content/www/us/en/technology-innovation/aerial-technology-light-show.html

[7] SKYMAGIC
2021 *Drone Light Shows.* Consulta: 17 de abril de 2021.

https://skymagic.show/

[8] ALONSO-MORA, Javier
2015 Optimal Control and Optimization Methods for Multi-Robot Systems [diapositiva]. Consulta: 23 de abril de 2019.

http://multirobotsystems.org/sites/default/files/slides/2015_RSS_MRS_Alonso-Mora.pdf

[9] LAVALLE, Steven
2016 *Planning algorithms.* Primera Edición. Cambridge University Press. Consulta: 20 de abril de 2019.

http://planning.cs.uiuc.edu/book.pdf

[10] SHARON, Guni
2015 "Conflict-based search for optimal multi-agent pathfinding". *Artificial Intelligence.* Volumen 219, número 2, pp. 40-66. Consulta: 20 de abril de 2019.

https://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/viewFile/5062/5239

[11] LEHMAN, Eric
2010 *Mathematics for Computer Science.* Primera Edición. MIT Press. Consulta: 01 de abril de 2021.
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-fall-2010/readings/MIT6_042JF10_notes.pdf

[12] SHARON, Guni
2012 "Meta-agent Conflict-Based Search For Optimal Multi-Agent Path Finding". Conferencia presentada en Fifth Annual Symposium on Combinatorial Search. Niagara Falls, 19 de julio. Consulta: 30 de abril de 2019.

http://www.bgu.ac.il/~felner/2012/bcbs.pdf

[13] STANDLEY, Trevor
2010 "Finding Optimal Solutions to Cooperative Pathfinding Problems". Conferencia presentada en AAAI'10 Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence. Atlanta, 11 de julio. Consulta: 10 de abril de 2019.

https://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1926/1950

[14] FELNER, Ariel
2012 "Partial-Expansion A* with Selective Node Generation". Conferencia presentada en AAAI'12 Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence. Toronto, 22 de julio. Consulta: 1 de mayo de 2019.

https://www.aaai.org/ocs/index.php/SOCS/SOCS12/paper/viewFile/5400/5200

[15] YOSHIZUMI, Takayuki
2000 "A* with Partial Expansion for large branching factor problems". Conferencia presentada en Proceedings of the Seventeenth National Conference on Artificial Intelligence. Texas, 03 de agosto. Consulta: 1 de mayo de 2019.

https://pdfs.semanticscholar.org/518f/950ad939f5978ed19bdb130a31aa6bb8c678.pdf

[16] GOLDENBERG, Meir
2014 “Enhanced Partial Expansion A*”. *Journal of Artificial Intelligence Research.* Volumen 50, número 5, pp. 141-187. Consulta: 1 de mayo de 2019.

https://www.jair.org/index.php/jair/article/view/10882/25958

[17] SHARON, Guni
2011 “The Increasing Cost Tree Search for Optimal Multi-agent Pathfinding”. Conferencia presentada en IJCAI'11 Proceedings of the Twenty-Second international joint conference on Artificial Intelligence. Barcelona, 16 de julio. Consulta: 1 de mayo de 2019.

https://www.ijcai.org/Proceedings/11/Papers/117.pdf

[18] VAN DEN BERG, Jur
2015 Multi-Robot Motion Planning #2 [diapositiva]. Consulta: 23 de mayo de 2019.

https://slideplayer.com/slide/5006830/

[19] PENG, Jufeng
2005 “Coordinating Multiple Robots with Kinodynamic Constraints along Specified Paths”. *The International Journal of Robotics Research.* Volumen 24, número 4, pp. 295-310. Consulta: 30 de abril de 2019.

https://webpages.uncc.edu/sakella/papers/PengAkellaIJRR05.pdf

[20] PENG, Jufeng
2003 “Coordinating the Motions of Multiple Robots with Kinodynamic Constraints”. Conferencia presentada en 2003 IEEE International Conference on Robotics and Automation. Taipéi, 14 de septiembre. Consulta: 30 de abril de 2019.

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.78.2112&rep=rep1&type=pdf

[21] AKELLA, Srinivas
2002 “Coordinating the Motions of Multiple Robots with Specified Trajectories”. Conferencia presentada en Proceedings 2002 IEEE International Conference on Robotics and Automation. Washington, 07 de agosto. Consulta: 30 de abril de 2019.

https://www.cc.gatech.edu/~seth/ResPages/pdfs/icra02a.pdf

[22] HÖNIG, Wolfgang
2016 “Multi-Agent Path Finding with Kinematic Constraints”. Conferencia presentada en ICAPS'16 Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling. Londres, 12 de junio. Consulta: 4 de junio de 2019.

https://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13183/12711

[23] YU, Jinglin
2016 “Optimal Multi-Robot Path Planning on Graphs: Complete Algorithms and Effective Heuristics”. *IEEE Transactions on Robotics.* Volumen 32, número 5, pp. 1163-1177. Consulta: 5 de junio de 2019.

https://arxiv.org/pdf/1507.03290.pdf

[24] YU, Jinglin
2013 “Planning Optimal Paths for Multiple Robots on Graphs”. Conferencia presentada en 2013 IEEE International Conference on Robotics and Automation. Karlsruhe, 17 de octubre. Consulta: 5 de junio de 2019.

http://msl.cs.uiuc.edu/~lavalle/papers/YuLav13.pdf

[25] YU, Jinglin
2012 “Time Optimal Multi-Agent Path Planning on Graphs”. Conferencia presentada en Twenty-Sixth AAAI Conference on Artificial Intelligence. Toronto, 15 de julio. Consulta: 5 de junio de 2019.

http://people.csail.mit.edu/jingjin/files/YuLav12WOMP.pdf

[26] HÖNIG, Wolfgang
2018 “Trajectory Planning for Quadrotor Swarms”. *IEEE Transactions on Robotics.* Volumen 34, número 4, pp. 856 -869. Consulta: 7 de junio de 2019.

http://act.usc.edu/publications/Hoenig_TRO2018.pdf

[27] PREISS, James
2017 “Downwash-Aware Trajectory Planning for Large Quadrotor Teams”. Conferencia presentada en 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Vancouver, 24 de setiembre. Consulta: 7 de junio de 2019.

http://act.usc.edu/publications/Preiss_IROS2017.pdf

[28] ARONSON, Jay
1989 “A survey of dynamic network flows”. *Annals of Operations Research.* Volumen 20, número 4, pp. 1-66. Consulta: 10 de mayo de 2019.

https://www.researchgate.net/publication/226470270_A_Survey_of_Dynamic_Network_Flows

[29] ROBINSON, Reed
2018 “An Efficient Algorithm for Optimal Trajectory Generation for Heterogeneous Multi-Agent Systems in Non-Convex Environments”. *IEEE Robotics and Automation Letters.* Volumen 3, número 2, pp. 1215-1222. Consulta: 10 de mayo de 2019.

https://ieeexplore.ieee.org/document/8260912

[30] SCHOUWENAARS, Tom
2001 “Mixed integer programming for multi-vehicle path planning”. Conferencia presentada en 2001 European Control Conference (ECC). Porto, 27 de abril. Consulta: 9 de junio de 2019.

http://hohmann.mit.edu/papers/ECC2001.pdf

[31] RICHARDS, Arthur
2002 “Coordination and Control of Multiple UAVs”. Conferencia presentada en AIAA Guidance, Navigation, and Control Conference and Exhibit. Monterrey, 05 de agosto. Consulta: 10 de mayo de 2019.

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.208.1820&rep=rep1&type=pdf

[32] MELLINGER, Daniel
2012 Trajectory Generation and Control for Quadrotors. Tesis de doctorado. Pensilvania: Universidad de Pensilvania, Facultad de Ingeniería Mecánica y Mecánica Aplicada. Consulta: 10 de mayo de 2019.

https://repository.upenn.edu/cgi/viewcontent.cgi?article=1705&context=edissertations

[33] UNIVERSIDAD STANFORD
2018 Sequential Convex Programming. Consulta: 30 de mayo de 2019.

https://web.stanford.edu/class/ee364b/lectures/seq_slides.pdf

[34] CHEN, Yufan
2015 “Decoupled Multiagent Path Planning via Incremental Sequential Convex Programming”. Conferencia presentada en 2015 IEEE International Conference on Robotics and Automation (ICRA). Seattle, 02 de julio. Consulta: 30 de mayo de 2019.

https://www.researchgate.net/publication/282983586_Decoupled_multiagent_path_planning_via_incremental_sequential_convex_programming

[35] AUGUGLIARO, Federico
2012 “Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach”. Conferencia presentada en 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. Vilamoura, 24 de diciembre. Consulta: 30 de mayo de 2019.

http://flyingmachinearena.org/wp-content/publications/2012/AugugliaroIROS2012.pdf

[36] RAVANKAR, Abhijeet
2018 “Path Smoothing Techniques in Robot Navigation: State-of-the-Art, Current and Future Challenges”. *Sensors.* Volumen 18, número 9. Consulta: 10 de junio de 2019.

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6165411/

[37] DOLGOV, Dmitri
2010 “Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments”. *The International Journal on Robotics Research.* Volumen 29, número 5, pp. 485-501. Consulta: 1 de enero de 2021.

https://journals.sagepub.com/doi/abs/10.1177/0278364909359210

[38] DOLGOV, Dmitri
2008 “Practical Search Techniques in Path Planning for Autonomous Driving”. Conferencia presentada en American Association for Artificial Intelli-gence. Consulta: 1 de enero de 2021.

https://ai.stanford.edu/~ddolgov/papers/dolgov_gpp_stair08.pdf

[39] PIVTORAIKO, Mihail
2009 “Differentially Constrained Mobile Robot Motion Planning in State Lattices”. *Journal of Field Robotics.* Volumen 26, número 3, pp. 308-333. Consulta: 1 de febrero de 2021.

https://rpal.cs.cornell.edu/docs/PivEtal_JFR_2009.pdf

[40] MCNAUGHTON, Matthew
2011 “Motion Planning for Autonomous Driving with a Conformal Spatiotemporal Lattice”. Conferencia presentada en 2011 IEEE International Conference on Robotics and Automation. Shanghai, 13 de mayo. Consulta: 1 de febrero de 2021.

https://ieeexplore.ieee.org/document/5980223

[41] SHIN, Heechan
2018 “Kinodynamic Comfort Trajectory Planning for Car-like Robots”. Conferencia presentada en 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Madrid, 1 de octubre. Consulta: 1 de julio de 2020.

https://ieeexplore.ieee.org/document/8593397

[42] RÖSMANN, Christoph
2012 “Trajectory modification considering dynamic constraints of autonomous robots”. Conferencia presentada en ROBOTIK 2012; 7th German Conference on Robotics. Munich, 21 de mayo. Consulta: 1 de julio de 2021.

https://ieeexplore.ieee.org/document/6309484

[43] PHILLIPS, Mike
2011 “SIPP: Safe interval path planning for dynamic environments”. Conferencia presentada en 2011 IEEE International Conference on Robotics and Automation. Shanghai, 13 de mayo. Consulta: 1 de enero de 2021.

https://ieeexplore.ieee.org/document/5980306

[44] KONG, Jason
2015 “Kinematic and dynamic vehicle models for autonomous driving control design”. Conferencia presentada en 2015 IEEE Intelligent Vehicles Symposium (IV). Seoul, 01 de julio. Consulta: 1 de enero de 2021.

https://ieeexplore.ieee.org/document/7225830

[45] RAJAMANI, Rajesh
2012 *Vehicle Dynamics and Control.* Segunda Edición. Springer. Consulta: 1 de enero de 2021.

https://link.springer.com/book/10.1007/978-1-4614-1433-9

[46] HABRADOR
2021 How to make Dubins Paths in Unity with C# code. Consulta: 2 de enero de 2021.

https://www.habrador.com/tutorials/unity-dubins-paths/2-basic-dubins-paths/

[47] ANDREYCHUK, Anton
2019 “Multi-Agent Pathfinding with Continuous Time”. Conferencia presentada en Twenty-Eighth International Joint Conference on Artificial Intelligence. Macao, 16 de agosto. Consulta: 1 de enero de 2021.

https://arxiv.org/pdf/1901.05506.pdf

[48] RIGATOS, Gerasimos
2015 *Nonlinear Control and Filtering Using Differential Flatness Approaches.* Primera Edición. Springer. Consulta: 8 de mayo de 2021.

https://link.springer.com/chapter/10.1007/978-3-319-16420-5_2

[49] SOUILEM, Haifa
2013 “Vehicle Control by Flatness”. Conferencia presentada en International Conference on Control, Engineering & Information Technology. Sousse, 4 de junio. Consulta: 8 de mayo de 2021.

http://ipco-co.com/PET_Journal/presented%20papers/Poster/002.pdf

[50] VESLIN, Elkin
2011 “Motion planning on Mobile Robots using Differential Flatness”. *IEEE Latin America Transactions.* Volumen 9, número 7, pp. 1006-1011. Consulta: 9 de mayo de 2021.

https://ieeexplore.ieee.org/document/6129695

[51] BESSAS, Aicha

2012 “Differential Flatness-based Planning and Trajectory Tracking of Wheeled Mobile Robots Application to car-like robot”. Conferencia presentada en The International Conference on Electromecanical Engineering. Skikda, 20 de noviembre. Consulta: 8 de mayo de 2021.

https://www.researchgate.net/publication/281678740_Differential_Flatness-based_Planning_and_Trajectory_Tracking_of_Wheeled_Mobile_Robots_Application_to_car-like_robot

[52] PEI, Chin

2009 “Differential Flatness-based Kinematic and Dynamic Control of a Differentially Driven Wheeled Mobile Robot”. Conferencia presentada en 2009 IEEE International Conference on Robotics and Biomimetics. Guilin, 19 de diciembre. Consulta: 8 de mayo de 2021.

https://ieeexplore.ieee.org/document/5420388

[53] TSARDOULIAS, Emmanouil

2016 “A Review of Global Path Planning Methods for Occupancy Grid Maps Regardless of Obstacle Density”. *Journal of Intelligent & Robotic Systems.* Volumen 84, número 1, pp. 829-858. Consulta: 9 de junio de 2021.

https://link.springer.com/article/10.1007/s10846-016-0362-z

[54] TAKAHASHI, Osamu

1989 “Motion planning in a plane using generalized Voronoi diagrams”. *IEEE Transactions on Robotics and Automation.* Volumen 5, número 2, pp. 143-150. Consulta: 19 de enero de 2021.

https://ieeexplore.ieee.org/document/88035

[55] REEDS, James

1990 “Optimal paths for a car that goes both forwards and backwards”. *Pacific Journal of Mathematics.* Volumen 145, número 2, pp. 367-393. Consulta: 20 de julio de 2020.

https://msp.org/pjm/1990/145-2/pjm-v145-n2-p06-s.pdf