Estrategias probabilísticas para la exploración cooperativa de robots móviles

Alfredo Toriz Palacios

17 de octubre de 2007

Resumen

La exploración de ambientes desconocidos puede considerarse como un problema fundamental para los robots móviles, dado que involucra todas las capacidades fundamentales de estos sistemas, por ejemplo, la percepción, la planificación, la localización y la navegación.

Una definición ampliamente aceptada sobre la exploración es "el acto de moverse a través de un ambiente desconocido mientras se construye un mapa que pueda utilizarse para subsecuentes navegaciones". El rendimiento de las estrategias de exploración debe ser valorado en base a la calidad del mapa obtenido y del tiempo necesario para construirlo.

El método de exploración implementado se basa en la generación aleatoria de configuraciones en un área segura local detectada por los sensores. Se crea una estructura de datos llamada *árbol aleatorio de exploración usando sensores (SRT)*, el cual representa el roadmap del área explorada asociado a una región segura (RS). Cada nodo del SRT consiste de una configuración libre y su región segura local (RSL) asociada; la RS es simplemente la unión de todas las RSLs pertenecientes al árbol. La RSL es una estimación del espacio libre circunvecino a una configuración dada del robot; en general, su forma dependerá de las características del sensor pero también puede reflejar diferentes posturas de percepción. En un trabajo anterior, se presentó la versión básica de SRT, en esta tesis presentamos una paralelización de esta versión.

El objetivo principal del presente trabajo es presentar estrategias de exploración cooperativa para robots móviles. La idea principal es la generación aleatoria incremental de una colección de estructuras de datos llamadas árbol aleatorio de exploración usando sensores (SRT), cada uno representa un roadmap de un área explorada con una región de seguridad asociada. La cooperación descentralizada y un mecanismo de coordinación son introducidos para mejorar la eficiencia de la exploración y evitar conflictos. Se presenta la simulación de varios ambientes para mostrar el desempeño de las técnicas propuestas.

Índice general

Capítulo 1

Introducción

La geometría y el movimiento son dos fenómenos omnipresentes en el mundo físico. Para operar en este mundo o simularlo necesitamos:

i) un modelo apropiado del ambiente, ii) estructuras de datos compactas para representar los modelos, y iii) algoritmos eficientes para generar los movimientos de los diferentes tipos de objetos presentes en el mundo.

En muchas aplicaciones se requieren robots con la habilidad, de sensar y moverse sin colisionar con obstáculos existentes en un ambiente dado. La planificación de movimientos (PM) de robots, es una disciplina que trata con tales problemas; en un sentido limitado, el problema básico consiste en encontrar movimientos libres de colisión para los robots en un ambiente conocido poblado con obstáculos [23]. La PM tiene su origen en la robótica, donde la planificación de movimientos libres de colisión, con el fin de alcanzar un objetivo específico, es una característica fundamental de los robots autónomos.

Una limitación de los algoritmos de planificación de movimientos es la necesidad de adquirir con anterioridad el modelo completo del ambiente para que el planificador pueda actuar. Una solución a este problema es la planificación de movimientos usando sensores (PMCS). La PMCS permite a los robots trabajar de forma autónoma en ambientes desconocidos. Específicamente, el robot es capaz de moverse a una configuración dada sin un conocimiento anticipado del ambiente. La falta de este conocimiento previo es la diferencia principal entre la planificación de movimientos que usa sensores y la planificación de movimientos que usa modelos. La planificación de

movimientos usando sensores no es, por consiguiente, un proceso fuera de línea. Ya que, el movimiento se genera paso a paso mientras se acumula más y más información acerca del ambiente.

La planificación de movimientos y la navegación son componentes muy importantes en la operación de un robot móvil autónomo. El problema de navegación trata con el cálculo de un camino libre de colisión para un robot desde una posición origen a una posición destino en un ambiente poblado de obstáculos. A los algoritmos de navegación para ambientes desconocidos frecuentemente se les refiere como métodos de planificación de movimientos usando sensores (MPMCS).

Los MPMCS incorporan la información de tales dispositivos en el proceso de planificación, reflejando el estado actual del ambiente, en contraste a la planificación de movimientos clásico. En estos enfoques, principalmente, se utilizan métodos locales, que pueden llamarse estrategias reactivas y están completamente basados en la información del sensor. Por lo tanto, no es un requisito contar con una localización absoluta y se considera solo la interacción relativa entre el robot y su ambiente. En estas circunstancias, es innecesario un modelado estructural del ambiente, pero el robot debe obtener a través de las entradas del sensor un conjunto de mecanismos de estimulo-respuesta. En este esquema, se espera que el robot lleve acabo tareas simples. Pero, ello no garantiza la solución de una tarea debido a la presencia de callejones sin salida.

En exploración de ambientes desconocidos, el uso de sensores es de vital importancia para actualizar la información sobre el ambiente y guiar el movimiento del robot a través de estrategias de retroalimentación en línea.

Exploración es la tarea básica por la cual un robot móvil cubre un área desconocida, típicamente aprendiendo un modelo del ambiente al mismo tiempo. Posibles aplicaciones incluyen vigilancia automatizada, operaciones de búsqueda y rescate en áreas hostiles, construcción de mapas y misiones planetarias.

Dotado de capacidades espaciales y temporales limitadas, un robot puede fracasar en una misión. Una escapatoria consiste en situar otros robots que compartan el mismo espacio de trabajo para conseguir el objetivo deseado.

El uso de un sistema multi-robot trae en general muchas ventajas. En ex-

ploración, tiene como objetivo reducir significativamente el tiempo requerido para completar la tarea de construir un mapa del ambiente, pero la pregunta central es ¿cómo coordinar el comportamiento de estos sistemas de múltiples robots?, especialmente en ambientes cuya estructura es desconocida. Si un mapa es adquirido, la información redundante proporcionada por múltiples robots puede ser utilizada para incrementar la exactitud del mapa final y la calidad de la localización. Para alcanzar estos objetivos, se requiere de alguna clase de descomposición y asignación de tareas.

En la práctica, las estrategias para distribuir convenientemente los robots sobre el ambiente deberían ser ideadas con el fin de reducir la ocurrencia de conflictos espaciales y realmente aprovechar los beneficios de una arquitectura multi-robot. Claramente, la comunicación juega un papel crucial en la realización de un comportamiento cooperativo con funcionamiento mejorado.

En muchas estrategias de exploración, el límite entre territorio conocido y desconocido (la frontera) es aproximado para maximizar la ganancia de información. En previas investigaciones, se ha demostrado que la exploración basada en fronteras puede ser usada para explorar edificios de oficinas y construir rejillas de ocupación que representan la estructura espacial de esos ambientes [46].

Para el caso multi-robot, un trabajo pionero en esta dirección es [47]: los robots mezclan la información adquirida en un mapa de rejilla global del ambiente, del cual la frontera es extraída y usada para planificar los movimientos individuales del robot. Mientras este esquema básico carece de un mecanismo arbitrario de prevención de robots acercándose a la misma región de la frontera, en [5] se propone una solución para negociar objetivos de robot optimizando una función de utilidad la cual toma en cuenta la información obtenida de una región particular, el costo de alcanzarlo y el número de robots que actualmente se dirigen allí. En [20] la utilidad de una región de frontera particular desde el punto de vista de la localización relativa del robot (y por lo tanto, de la exactitud del mapa combinado) es también considerado. En el algoritmo de despliegue incremental de [16], los robots se aproximan a la frontera mientras se mantiene contacto visual con los otros. Una arquitectura interesante multi-robot en la cual los robots son guiados con exploración por un mercado económico es presentada en [50], mientras que [42] propone un enfoque centralizado el cual usa una búsqueda basada en fronteras y un protocolo de oferta asignando objetivos de frontera

a los robots.

Este documento presenta estrategias probabilísticas [1] para la exploración cooperativa que son consecuencia del método SRT diseñado para un solo robot. La herramienta básica propuesta en este trabajo es el árbol aleatorio de exploración usando sensores (SRT), una estructura de datos compacta que representa un roadmap del "área explorada" la cual puede ser considerada como una versión basada en sensor del algoritmo RRT [10].

En particular, cada nodo de un SRT contiene una configuración asumida por el robot y la región segura local (RSL) percibida de esa localización, mientras un arco entre dos nodos representa una ruta libre de colisión entre las dos configuraciones. El SRT es incrementalmente construido por el robot usando un planificador aleatorio local el cual privilegia la frontera de la RSL, es decir, las direcciones que conducen de la RSL a áreas no exploradas. Este mecanismo automáticamente realiza una compensación entre la información obtenida y el costo de navegación cuando se elige la siguiente configuración del robot.

Nuestras estrategias de exploración cooperativa son esencialmente una paralelización del método SRT para un solo robot, con la adición de tres funcionalidades: (i) cooperación, para incrementar la eficiencia (ii) coordinación, para evitar conflictos y (iii) comunicación, como herramienta principal para cooperar y coordinar.

Cada robot del equipo construye un SRT, tomando en cuenta la presencia de otros robots por una apropiada redefinición del concepto de frontera local, y planificando sus movimientos hacia áreas que aparecen como no exploradas por el, así como por el resto del equipo. Además de este mecanismo de cooperación local, hay un algoritmo simple de coordinación que garantiza

[1] En planificación de movimientos con este tipo de algoritmos (PRM o RRT), el planificador no construye una representación exacta del espacio de configuraciones libre de tal manera que nunca se conoce la forma exacta de este espacio, en particular su conectividad. En cualquier momento durante la planificación, muchas hipótesis acerca de la forma del espacio libre son consistentes con la información recolectada hasta ahora. La medida de probabilidad para muestrear el espacio libre deriva de esta incertidumbre. Por lo tanto, el planificador negocia el costo de calcular el espacio libre exactamente contra el costo de tratar con la incertidumbre. Esta opción es beneficiosa solamente si el muestreo probabilistico conduce a un roadmap que sea mucho más pequeño en tamaño que el de una representación exacta del espacio libre y todavía represente el espacio libre de manera adecuada para contestar preguntas de planificación de movimientos correctamente.

movimiento colectivo seguro. Una vez que un robot ha completado su SRT, esta en disposición de apoyar a otros robots en su expansión; esto introduce una forma de cooperación global. Una característica clave de las estrategias propuestas es que es completamente descentralizada y puede ser implementada con un rango de comunicación limitado e ilimitado.

A continuación se describe brevemente el contenido de los capítulos de este trabajo de tesis.

En este trabajo, se trata con el problema de exploración cooperativa de robots móviles de tipo carro. Se utiliza el método SRT, que esta inspirado en en los planificadores RRT (estos manejan correctamente las restricciones diferenciales del robot móvil).

En el capítulo 2, se presenta un análisis general de los métodos RRT, base para nuestro enfoque.

La planificación multi-robot se detalla en el capítulo 3, así como la técnic de coordinación adoptada.

El capitulo 4 presenta con detalles el método SRT. Se hace un estudio comparativo de la estrategia propuesta por los autores originales y la de Judith Espinoza (SRT-Radial Vs SRT-Star). Se presenta además, la paralelización de SRT para resolver el problema dela exploración cooperativa.

Una erie de experimentos para validar el enfoque adoptado en la exploración cooperativa se presenta en el capítulo 5.

Finalmente, la conclusión y discusión de trabajos futuros se presenta en el capítulo 6.

Capítulo 2

Árboles aleatorios de exploración rápida

El árbol aleatorio de exploración rápida (RRT, del inglés, Rapidly-Exploring Random Tree) fue presentado en [24] como un algoritmo de planificación para búsqueda rápida en espacios de altas dimensiones que tienen tanto restricciones algebraicas (provenientes de los obstáculos) como restricciones diferenciales (originadas por la no-holonomía y la dinámica). La idea clave es dirigir la exploración hacia regiones no exploradas del espacio tomando puntos en el espacio de estados e incrementalmente "jalar" el árbol hacia ellos. Por lo menos, se han propuesto otras dos técnicas aleatorias de planificación de caminos (planificación holonómica): el algoritmo de Hilo de Ariadna [32] y los planificadores en [48]. Intuitivamente, estos planificadores intentan "empujar" la búsqueda lejos de los vértices previamente construidos, en contraste, con el RRT el cual usa el espacio circunvecino para "jalar" la búsqueda. Hasta donde llega nuestro conocimiento, no se había propuesto un método aleatorio con árboles de búsqueda para planificación no-holonómica o kinodynamic. Quizá, los métodos más aproximados son [41, 43], en donde, el método roadmap probabilístico se combina con técnicas de conducción no holonómica para planificar caminos con robots móviles rodantes.

2.1. Formulación del problema

El tipo de problemas considerados por el enfoque RRT están formulados en términos de seis componentes:

1. **Espacio de estados:** Un espacio topológico, X.

2. **Valores limite:** $x_{init} \in X$ y $X_{goal} \subset X$

3. **Detector de colisión:** Una función, $D : X \rightarrow \{true, false\}$, que determina sí las restricciones globales son satisfechas desde el estado x. Esta podría ser una función binaria o real.

4. **Entradas:** Un conjunto, U, que especifica el conjunto de controles o acciones que pueden afectar al estado.

5. **Simulador incremental:** Dado el actual estado, $x(t)$, y las entradas aplicadas sobre un intervalo de tiempo, $\{u(t') | t \leq t' \leq t + \Delta t\}$, calcular $x(t + \Delta t)$.

6. **Métrica:** Una función real, $\rho : X \times X \rightarrow [0, \infty)$, la cual especifica la distancia entre pares de puntos en X.

La planificación de caminos generalmente es vista como una búsqueda en el espacio de estados, X, para un camino continuo desde un estado inicial, x_{init} a una región meta $X_{meta} \subset X$ o un estado meta $x_{meta} \in X$. Se asume que se tiene un conjunto complicado de restricciones diferenciales sobre X y cualquier camino solución debe mantener al estado dentro de este conjunto. Un detector de colisión reporta si un estado dado, x, satisface las restricciones globales. Generalmente, se utiliza la notación, X_{free} para referirse al conjunto de estados que satisfacen las restricciones globales. Se asignan restricciones locales y diferenciales a través de un conjunto de entradas (o controles) y de un simulador incremental. Estos dos componentes especifican los posibles cambios en el estado. El simulador incremental puede definirse por integración numérica de una *ecuación de transición de estado*. Finalmente, se define una métrica para indicar la cercanía de pares de puntos en el espacio de estados.

2.1.1. Planificación de caminos básica (holonómica)

La planificación de caminos, generalmente, es vista como una búsqueda en el espacio de configuraciones, C, en el cual cada configuración, $q \in C$, especifica la posición y orientación de uno o mas cuerpos articulados en un mundo 2D o 3D. La tarea de la planificación de caminos es calcular un camino continuo desde una configuración inicial, q_{init} a una configuración meta q_{goal}. De esta manera, $X = C$, $x_{init} = q_{init}$, $x_{goal} = q_{goal}$, y $X_{free} = C_{free}$, este último denota al conjunto de configuraciones en el

cual los cuerpos no están en colisión con los obstáculos estáticos del ambiente. Los obstáculos son completamente modelados en el ambiente, pero no se tiene disponible una representación explicita de X_{free}. Sin embargo, se puede examinar un estado dado a través de un algoritmo de detección de colisiones. Para ser precisos, se utiliza un algoritmo de cálculo de distancia para indicar cuan cercanos están los cuerpos geométricos de violar las restricciones del ambiente. Este, se utiliza para asegurar que las configuraciones intermedias están libres de colisión cuando el simulador incremental realiza saltos discretos. El conjunto de entradas, U, es el conjunto de todas las velocidades $\dot{x}$ tal que $\|\dot{x}\| \leq c$ para una constante positiva c. El simulador incremental produce un nuevo estado por integración, $x_{new} = x + u\Delta t$, para una entrada dada $u \in U$.

2.1.2. Planificación de caminos no-holonómica

La planificación no-holonómica maneja problemas que involucran restricciones no-integrables en el estado de velocidades, además de los componentes que aparecen en los problemas de la planificación de caminos básica. Con frecuencia, estas restricciones surgen en muchos contextos tal como sistemas de robots con ruedas, y manipulación por empuje. Las restricciones diferenciales a menudo aparecen en forma implícita, $h_i(\dot{q}, q) = 0$ para algún $i = 1..k < N$ (N es la dimensión de C). Por el teorema de función implícita, las restricciones se pueden expresar en la forma de control teórico, $\dot{q} = f(q, u)$, u es una entrada elegida de un conjunto de entradas, U. Usando, una notación mas general sería, $\dot{x} = f(x, u)$. A esta forma se le denomina, *ecuación de transición de estado* o *ecuación de movimiento*. El simulador incremental lo podemos construir utilizando esta ecuación de estado por integración numérica, (utilizando, por ejemplo, las técnicas de Runge-Kutta).

2.1.3. Planificación kinodynamic

En la planificación kinodynamic [1] existen tanto restricciones en la velocidad como en la aceleración, produciendo ecuaciones de la forma $h_i(\ddot{q}, \dot{q}, q) = 0$. Un estado, $x \in X$, se define como $x = (q, \dot{q})$ para $q \in C$. Usando la representación del espacio estado, se puede escribir, como un conjunto de m ecuaciones implícitas de la forma $G_i(x, \dot{x}) = 0$, para $i = 1, .., m$ y $m < 2N$. De igual forma se puede obtener la ecuación de transición de estado aplicando el teorema de función implícita.

[1] Palabra técnica tomada del inglés, con la que se refiere a restricciones tanto cinemáticas como dinámicas. No se tiene una traducción directa al español.

2.1.4. Otros problemas de planificación con RRT

Una variedad de problemas se adaptan a la formulación del problema y pueden abordarse usando las técnicas RRT. En general, puede formularse cualquier problema de diseño de trayectorias de lazo abierto, ya que la mayoría de los modelos son propios de la teoría de control. El planificador podría usarse para calcular una estrategia que controle un circuito eléctrico o un sistema en economía. En algunas aplicaciones no se puede conocer la ecuación de transición de estado, pero esto no significaría un problema. Por ejemplo, un simulador en física podría desarrollarse simulando un diseño propuesto para un carro de carreras. El software simplemente aceptaría entradas de control en algunas secciones del muestreo, y produciría nuevos estados. También pueden manejarse problemas que involucren restricciones de cerradura cinemática.

2.2. Árboles aleatorios de exploración rápida

El algoritmo básico de construcción de RRT, se muestra en la figura 2.1. En cada iteración se intenta extender el árbol agregando un nuevo vértice en dirección a un estado seleccionado aleatoriamente. La función EXTENDER, ilustrada en la figura 2.2, selecciona del árbol el vértice más cercano a un estado dado. Este vértice se elige de acuerdo a una métrica, ρ. La función NUEVO_ESTADO hace un movimiento hacia x aplicando una entrada $u \in U$ para algún incremento Δt. Esta entrada puede seleccionarse aleatoriamente o probando todas las posibles entradas eligiendo aquella que produzca un nuevo estado tan próximo como sea posible a x (si U es infinito puede usarse una aproximación o una técnica analítica). NUEVO_ESTADO utiliza implícitamente una función de detección de colisiones para determinar si el nuevo estado (y todos los estados intermedios) satisfacen las restricciones globales. Si NUEVO_ESTADO se cumple, el nuevo estado junto con la entrada se representan por medio de x_{new} y u_{new}, respectivamente. Pueden ocurrir tres situaciones: *Alcanzado*, el nuevo vértice alcanza al estado muestreado x, (para la planificación no-holonómica, tendríamos un umbral, $\|x_{new} - x\| < \epsilon$ para un ϵ pequeño, $\epsilon > 0$); *Avanzado*, un nuevo vértice $x_{new} \neq x$ es agregado al RRT. *Atrapado*, NUEVO_ESTADO falla en producir un nuevo estado que se encuentre en X_{free}.

CONSTRUIR_RRT(x_{init})
1 $\mathcal{T}$.init(x_{init});
2 **para** *k=1* **a** K
3 $x_{rand} \leftarrow$ ESTADO_ALEATORIO();
4 EXTENDER($\mathcal{T}, x_{rand}$);
5 Regresa $\mathcal{T}$

EXTENDER($\mathcal{T}, x$)
1 $x_{near} \leftarrow$ VECINO_MAS_PROXIMO($x, \mathcal{T}$);
2 **si** NUEVO_ESTADO($x, x_{near}, x_{new}, u_{new}$) **entonces**
3 $\mathcal{T}.agrega.vertice(x_{new})$;
4 $\mathcal{T}.agrega.arista(x_{near}, x_{new}, u_{new})$;
5 **si** $x_{new} = x$ **entonces**
6 Regresa *Alcanzado;*
7 **sino**
8 Regresa *Avanzado;*
9 Regresa *Atrapado;*

Figura 2.1: Algoritmo básico de construción del RRT

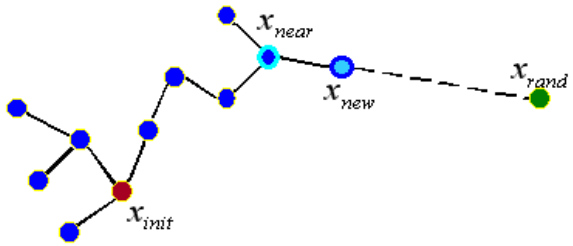


Figura 2.2: Operación de la función EXTENDER

2.3. Diseño de planificadores de caminos

Generalmente consideramos al RRT como un bloque en la construcción de un planificador eficiente. Por ejemplo, podríamos usar un RRT para escapar de un mínimo local en un planificador de caminos aleatorio con campos de potencial. En [44], un RRT fue usado como planificador local para el planificador roadmap probabilístico. Presentamos varias alternativas de planificadores que utilizan un RRT. La elección recomendada depende de varios factores, tal como, sí existieran restricciones diferenciales, el tipo de algoritmo para la detección de colisiones, o la eficiencia en el cómputo del vecino más próximo.

2.3.1. Planificadores RRT simples

En principio, el ***RRT básico*** puede usarse como un planificador de caminos por que sus vértices, eventualmente, cubrirían un componente conectado de X_{free}, llegando arbitrariamente cerca de cualquier x_{goal} especificado. El problema es que sin ningún sesgo hacia el objetivo, la convergencia es lenta. Un planificador mejorado, llamado ***RRT_GoalBias***, se obtiene reemplazando la función ESTADO_ALEATORIO en la figura 2.1 con otra que elija, con cierta probabilidad, entre x_{goal} o un estado tomado de cualquier parte del espacio de estados. Incluso con una probabilidad pequeña (p.ej. 0.05) de elegir x_{goal}, RRT_GoalBias usualmente converge al objetivo mucho más rápido que el RRT básico. Pero, si se designa un sesgo muy grande hacia el objetivo entonces el planificador empieza a comportarse como un planificador aleatorio de campos de potencial, ya que pueden presentarse mínimos locales. Una mejora es, ***RRT_GoalZoom*** el cual reemplaza la función ESTADO_ALEATORIO con una decisión, basada en cierta probabilidad, de optar por un estado aleatorio dentro de una región circunvecina al objetivo o por uno elegido del espacio de estados completo. El tamaño de la región alrededor del objetivo lo controla el vértice del árbol más cercano al objetivo. El efecto es que el foco de muestreo converge al objetivo como el RRT se acerca a él. Este planificador funciona bastante bien en la práctica, sin embargo, su rendimiento aún puede degradarse debido a mínimos locales. En general, parece mejor sustituir la función ESTADO_ALEATORIO con un esquema de muestreo que extraiga los estados desde una función de densidad de probabilidad no-uniforme con un sesgo gradual hacia el objetivo. La figura 2.3 muestra un ejemplo de un RRT construido al muestrear los estados usando una función de densidad de probabilidad que asigna una probabilidad equitativa a anillos concéntricos. Hay muchas investigaciones

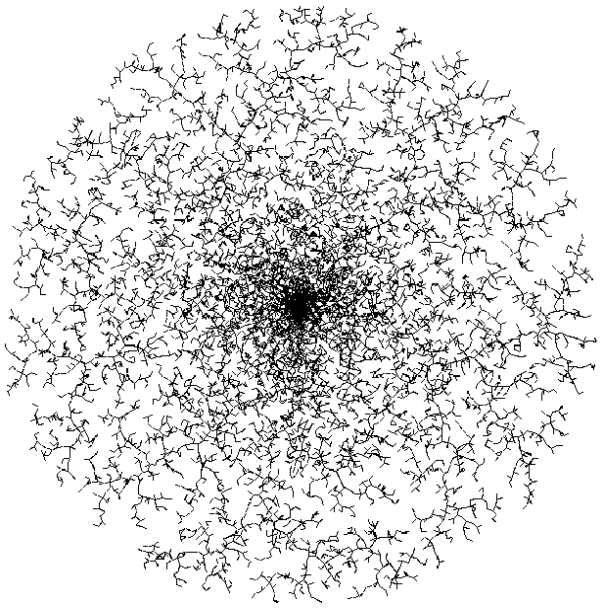


Figura 2.3: Un RRT en 2D construido usando un muestreo sesgado

interesantes con respecto al problema del muestreo.

Un problema más a considerarse, es el tamaño del paso usado en la construcción del RRT. Este podría elegirse dinámicamente durante la ejecución, acorde con la función que calcula la distancia en la detección de colisión. Si los cuerpos están lejos de colisionar, se puede tomar un paso grande. Además de seguir esta idea, ¿qué tan lejos debe estar x_{new} de x_{near}?, ¿Trataríamos de conectar x_{near} con x_{rand}?. En lugar de intentar extender un RRT por un paso incremental, EXTENDER puede ejecutarse iterativamente hasta alcanzar al estado aleatorio o un obstáculo, como se muestra en descripción del algoritmo CONECTAR de la figura 2.4. Al sustituir EXTENDER por CONECTAR el árbol crece muy rápido, si lo permiten las restricciones de detección de colisión y las restricciones diferenciales. Una de las ventajas clave de la función CONECTAR es que construimos un camino tan largo como sea posible con una sola llamada al algoritmo del vecino más próximo. Esta ventaja motiva la elección de un algoritmo voraz. La experiencia ha demostrado que la función CONECTAR produce mejores resultados en problemas holonómicos y la función EXTENDER es sobresaliente en problemas no-holonómicos. Una razón para esta diferencia es que CONECTAR confía

CONECTAR($\mathcal{T}, x$)
1 **repetir**
2 S $\leftarrow$ EXTENDER($\mathcal{T}, x$);
3 **hasta que no(S =** *Avanzado)*
4 Regresa S

Figura 2.4: Función CONECTAR

más en la métrica, y los problemas no-holonómicos requieren del diseño de buenas métricas.

2.3.2. Planificadores bidireccionales

Inspirados en la técnicas clásicas de búsqueda bidireccional, parece razonable esperar un mejor desempeño al crecer dos árboles de exploración, uno desde x_{init} y el otro a partir de x_{goal}; se obtiene una solución cuando los dos árboles se encuentran. Para una búsqueda simple, la implementación es directa, sin embargo, la construcción RRT debe guiarse para asegurar que ambos árboles se encuentren antes de cubrir el espacio entero y permitir una unión eficaz.

La figura 2.5 muestra el algoritmo ***RRT_BI-DIRECCIONAL***, puede compararse con el algoritmo CONSTRUIR_RRT de la figura 2.1. El RRT_BI-DIRECCIONAL divide el tiempo de cómputo entre dos procesos: 1) explorar el espacio de estados; 2) intentar crecer los árboles uno hacia el otro. Siempre existen dos árboles $\mathcal{T}_a$ y $\mathcal{T}_b$, hasta que se enlazan y se encuentra una solución. En cada iteración un árbol crece, y se intenta conectar el nuevo vértice con aquel más cercano en el otro árbol. Entonces, se invierten los roles intercambiando los árboles. El crecimiento de dos RRTs también fue propuesto en [27] para planificación kinodynamic, en ese enfoque, en cada iteración ambos árboles crecían hacia un estado aleatorio. El algoritmo actual intenta que los árboles crezcan uno hacia el otro en la mitad de tiempo, con lo cual se obtiene un buen rendimiento.

Se consideran algunas variaciones del planificador anterior. Puede reemplazarse cualquier ocurrencia de EXTENDER con CONECTAR en el RRT-BIDIRECCIONAL. Cada reemplazo hace a la operación más agresiva. Si sustituimos EXTENDER por CONECTAR en la línea 4, entonces el planificador explora agresivamente el espacio de estados, con la misma compen-

RRT_BIDIRECCIONAL(x_{init}, x_{goal})
1 $\mathcal{T}_a.inic(x_{init})$; $\mathcal{T}_b.inic(x_{goal})$;
2 **para** *k=1* **a** K
3 $x_{rand} \leftarrow$ ESTADO_ALEATORIO();
4 **si** (EXTENDER($\mathcal{T}_a, x_{rand}$) <> *Atrapado*) **entonces**
5 **si** (EXTENDER($\mathcal{T}_b, x_{new}$) = *Alcanzado*) **entonces**
6 Regresa CAMINO($\mathcal{T}_a, \mathcal{T}_b$);
7 INTERCAMBIAR($\mathcal{T}_a, \mathcal{T}_b$);
8 Regresa *Fallo*;

Figura 2.5: Planificador bidireccional usando un RRT

sación que el RRT básico. Si reemplazamos EXTENDER con CONECTAR en la línea 5, el planificador intenta unir los árboles agresivamente en cada iteración. Esta variante sería muy exitosa para resolver problemas de planificación holonómica, por comodidad la denominamos ***RRT_ExtCon*** y al algoritmo original como ***RRT_ExtExt***. Entre las variantes discutidas hasta el momento, encontramos a RRT_ExtCon más eficiente para problemas holonómicos [28] y a RRT_ExtExt ideal para problemas no-holonómicos. El planificador más agresivo que se puede construir es sustituyendo las dos ocurrencias de EXTENDER por CONECTAR, líneas 4 y 5, originando RRT_ConCon.

Concluimos, que el enfoque bidireccional es mucho más eficiente que el RRT básico. Una limitación al utilizar un RRT bidireccional en problemas de planificación no-holonómicos y kinodynamic es que necesitamos conectar varios vértices para unir un RRT al otro. Para un problema de planificación que involucre alcanzar una región meta desde un estado inicial, no se necesita conectar ningún vértice. El espacio entre las dos trayectorias puede cerrarse, en la práctica usando, si es posible, métodos de conducción o métodos clásicos de disparo (shooting), presentes con frecuencia en problemas de valores en la frontera.

2.3.3. Planteamientos adicionales con RRT

Si un enfoque dual ofrece ventajas sobre un árbol simple, entonces es natural preguntar sí el crecimiento de tres o más árboles ofrecería mayores ventajas. Estos árboles adicionales, iniciarían en estados aleatorios. Por supuesto, los problemas de conexión en el caso de la planificación no-

holonómica serían aún más difíciles de resolver. El tiempo de computo debería dividirse entre intentar extender los árboles y tratar de conectarlos unos a otros. Muchas cuestiones quedan por investigar en este y otros planificadores que usen un RRT.

Es interesante considerar el caso límite, en el cual se construye un nuevo RRT para cada estado aleatorio x_{rand}. Una vez generado un nuevo vértice, puede aplicarse la función CONECTAR, de la figura 2.4, a cada RRT. Para mejorar el rendimiento, podemos considerar los vértices que están a una distancia fija de x_{rand}, de acuerdo con la métrica. Si una conexión tiene éxito entonces los dos árboles se unen en un único grafo. El algoritmo resultante, simula el comportamiento del roadmap probabilístico. Por lo que, el roadmap probabilístico puede considerarse como una versión extrema de un algoritmo que usa RRTs, donde se construye y une un número máximo de RRTs independientes.

2.4. Aplicación de los RRTs en robots tipo carro

El diseño de trayectorias para automóviles es un problema importante tanto en robótica como en el desarrollo de prototipos virtuales. En robótica, se necesitan algoritmos que puedan calcular trayectorias para vehículos autónomos en ambientes complicados. En la industria automovilística, los simuladores se utilizan exhaustivamente para evaluar el rendimiento de los vehículos a través de prototipos virtuales. La mayoría de estas simulaciones involucran estrechamente al operador humano con el sistema de simulación. Los planificadores RRTs podrían utilizarse en el diseño de prototipos virtuales para automóviles, considerándolos como un "conductor virtual de prueba" que intenta lograr las condiciones especificadas, como conducir a través de un camino lleno de obstáculos. A continuación se presentan ejemplos de planificadores RRTs aplicados a robots tipo carro.

2.4.1. Modelo cinemático no-lineal para un robot tipo carro

Las restricciones diferenciales presentadas en esta sección son únicamente cinemáticas. Por lo cual, el espacio de estados se reduce al espacio de configuraciones (conjunto de todas las transformaciones que pueden aplicarse al robot). El modelo es tridimensional, ver la figura 2.6, cualquier estado se representa por medio de (x, y, θ) y su ecuación de transición de estado es:

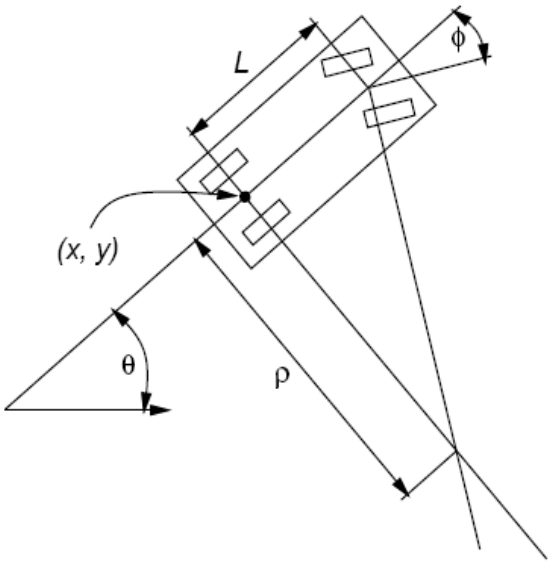


Figura 2.6: Modelo cinemático para un robot tipo carro

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} s\cos\theta \\ s\,\mathrm{sen}\,\theta \\ \frac{s}{L}\tan\phi \end{pmatrix} \tag{2.1}$$

Donde, L denota la distancia entre el eje frontal y el trasero, s es la velocidad y ϕ representa al ángulo de conducción. Este último esta limitado, $|\phi| \leq \phi_{max} < \frac{\pi}{2}$. El vector de entrada es (s, ϕ). Si el carro viaja sólo hacia delante, fijamos a $s = 1$, obteniendo así el carro Dubins [8]. Si el carro se mueve hacia atrás y hacia delante, $s = -1$ o $s = 1$, obtenemos un carro Reeds-Shepp [37]. La figura 2.7 muestra dos RRTs y los caminos encontrados utilizando un planificador RRT bidireccional en un ambiente conocido poblado de obstáculos. Las imágenes en las figuras 2.7.a y 2.7.c muestran una proyección bidimensional del RRT en el plano XY. Nótese que el RRT de la figura 2.7.a contiene cúspides que corresponden a los movimientos de reversa, sin embargo el RRT de 2.7.c no las contiene. La figura 2.8 muestra ejemplos un poco más complicados.

En el modelo de la ecuación 2.1, el ángulo de conducción ϕ es una entrada, esto implica que podemos mover instantáneamente las llantas frontales.

En muchas aplicaciones esta suposición es poco realista. Hay una curvatura discontinua, en el camino trazado por el centro del eje trasero del carro, generada cuando el ángulo de conducción cambia discontinuamente. Podemos obtener un modelo de carro que genere caminos suaves, "modelo smoot", agregando el ángulo de conducción como variable de estado. La entrada es la velocidad angular, ω, del ángulo de conducción.

El resultado es un espacio de estados de cuatro dimensiones, en el cual cada estado se representa por (x, y, ϕ, θ), con su consiguiente ecuación de transición:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} s\cos\theta \\ s\,\mathrm{sen}\,\theta \\ \omega \\ \frac{s}{L}\tan\phi \end{pmatrix} \tag{2.2}$$

La nueva entrada representa un cambio en el ángulo de conducción. La figuras 2.8.a y 2.8.b ilustra como los caminos, calculados a través de este modelo, son más suaves. Las figuras 2.8.c y 2.8.d muestran la aplicación de un RRT-Bidireccional a un carro Dubins que puede moverse solo hacia delante virando a la izquierda. Como se ve en la figura 2.8.d los RRTs pueden utilizarse para construir soluciones en ambientes muy complicados.

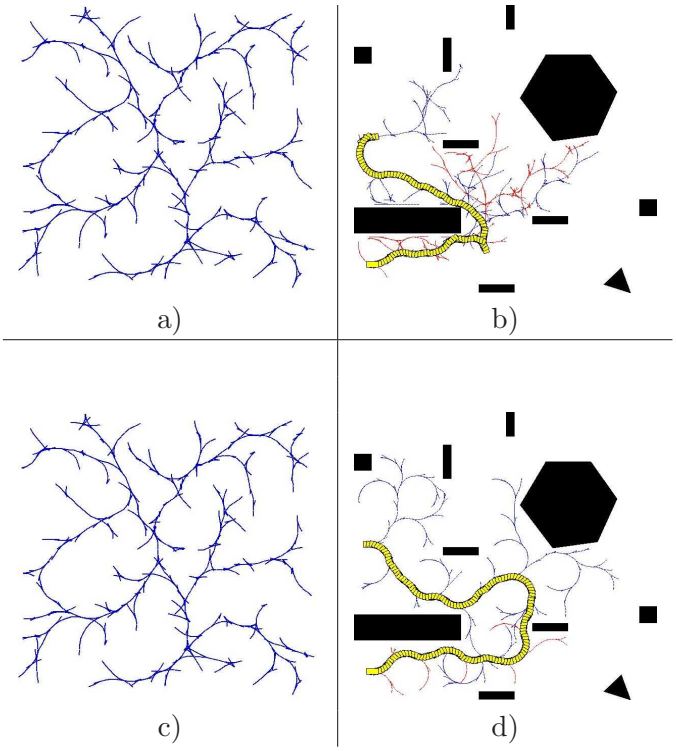


Figura 2.7: a) Árbol de exploración rápida para un carro Reeds-Shepp; b) Camino calculado usando un carro Reeds-Shepp; c) RRT para un carro Dubins; d) Camino calculado para el carro Dubins. Ambos caminos se obtuvieron con el planificador RRT-Bidireccional

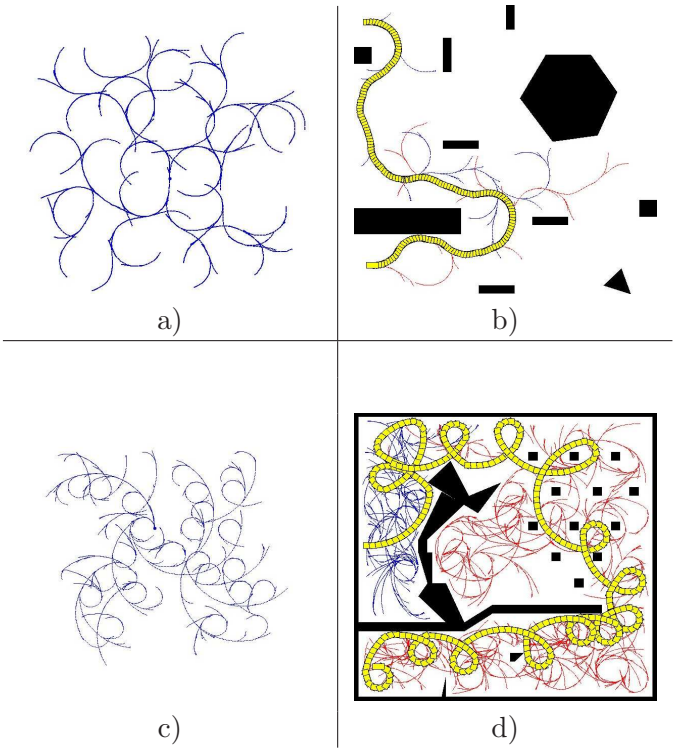


Figura 2.8: a) Árbol de exploración rápida para el modelo "smooth"del carro; b) Camino calculado usando un carro "smooth"; c) RRT para un carro Dubins, avanza sólo virando a la izquierda; d) Camino calculado con el carro de avance delantero y conducción solo a la izquierda. Los caminos se obtuvieron con el planificador RRT-Bidireccional

Capítulo 3

Planificación multi-Robot

3.1. Enfoques principales

Un sistema multi-robot está constituido, como su nombre lo indica, de diversos robots capaces de navegar en un espacio de trabajo común para la realización de una o diversas tareas coordinando sus acciones. El desplazamiento de un robot es el resultado de un proceso de razonamiento, más o menos complejo, y está en función de la arquitectura escogida. La arquitectura del sistema multi-robot define por lo tanto el tipo de posibles interacciones entre los robots, así como la naturaleza de las soluciones adoptadas.

La planificación en sistemas multi-robot representa más que una simple extensión al caso de un solo robot. Dotado de capacidades espaciales y temporales limitadas (no se consideran limitaciones intrínsecas del robot, carga máxima, autonomía energética etc.), un robot puede fracasar en una misión. Una posible solución a esta situación de un solo robot consiste en situar otros robots que compartan el mismo espacio de trabajo para conseguir el objetivo deseado. Por el contrario, los robots pueden evolucionar en un contexto donde las tareas que deben ejecutar son concurrentes a las de los otros robots. La afectación de las tareas, su coordinación, su ejecución y el control de ejecución son problemas más complejos que no se consideran en este trabajo.

El problema de planificación multi-robot es PSPACE-duro [23]. Su resolución induce en el caso general una complejidad temporal prohibitiva. Clásicamente los enfoques de planificación de movimientos multi-robot se reagrupan en dos categorías:

- **Los enfoques centralizados**. Los enfoques centralizados consideran el conjunto de robots como un solo sistema, en el cual su espacio de configuraciones es la combinación de los espacios de configuraciones de todos los robots. Un camino calculado en este espacio permite deducir directamente el camino asociado a cada robot. Centralizar la resolución del problema permite teóricamente garantizar la completez. En la práctica, bajo algunas hipótesis simplificadoras (que generalmente hacen perder la completitud), la complejidad de estos métodos es exponencial en la dimensión del espacio de configuraciones. En este estudio, Schwartz y Sharif [40] propusieron un enfoque basado sobre las curvas críticas, las cuales permiten planificar los movimientos de diversos discos en el plano. Sin embargo, los algoritmos propuestos tienen una complejidad polinomial de acuerdo al número de muros de los obstáculos y una complejidad exponencial de acuerdo al número de discos.

- **Los enfoques desacoplados**. Los enfoques desacoplados planifican un camino para cada robot de manera más o menos independiente y toman en cuenta sus interacciones a posteriori. Kant y Zucker propusieron, uno de los primeros esquemas que descomponen el problema de la planificación de caminos en dos sub problemas. El primero consiste en planificar para cada objeto móvil un camino sin colisión con los obstáculos estáticos ignorando los otros objetos móviles. El segundo consiste en planificar la velocidad de los objetos móviles a lo largo de su camino de tal manera que alcancen su posición final si entrar en colisión [19].

Pero además de estos, existen otros esquemas que se han propuesto con esta misma idea, los cuales se describen a continuación.

- **La planificación con prioridades**. La planificación puede efectuarse según prioridades (decrecientes) asignadas a los robots [9]: Un robot planifica entonces su camino en función de los obstáculos estáticos y de los robots de más alta prioridad, considerados como obstáculos móviles. Un robot está además, más restringido en sus movimientos si su prioridad es baja. El enfoque utiliza la noción del espacio de configuraciones espacio-temporal. La definición de las prioridades asociadas a los robots se suponen conocidas a priori en estos trabajos, y no es trivial de definir.

- **La coordinación de caminos**. Otro enfoque llamado coordinación de caminos fue propuesto en [33] sobre la base de dos robots R_1 y R_2, se calculan los caminos τ_1 y τ_2 independientemente, se construye un diagrama de coordinación en el producto cartesiano de los espacios de las abscisas curvilíneas $s1$ y $s2$ de los robots. Un punto $(s1, s2)$ en el diagrama corresponde respectivamente a las configuraciones $\tau(s1)$ y $\tau(s2)$ de los robots. El punto se considera como prohibido si los robots se interceptan en estas configuraciones. El problema de la coordinación de dos robots se plantea entonces al caso de encontrar un camino en el diagrama de coordinación $s1s2$ uniendo la esquina inferior izquierda a la esquina superior derecha evitando las zonas prohibidas de $s1s2$; si existe, se deduce una coordinación sin colisión en velocidad de los dos robots. La existencia de este camino depende naturalmente de los dos caminos planificados.

- **Las arquitecturas multi-agentes**.La arquitectura de un sistema multi-agente proporciona el estudio necesario para definir el tipo de comportamiento colectivo deseado y determina las capacidades y las limitaciones del sistema. La arquitectura puede ser definida como la parte fija del sistema que no puede ser cambiada más que por la intervención de un operador externo. Así pues, la elección de una arquitectura dada durante la fase de especificación permite definir las modalidades de cooperación y de especificar los parámetros condicionando la naturaleza de las interacciones entre las diferentes entidades del mundo. La primera pregunta a la cual el conceptualizador debe responder tiene relación con la naturaleza lógica del sistema. Es necesario concebir un sistema lógicamente centralizado donde la toma de decisión y el control se hace por un solo y único agente, o es necesario crear un sistema descentralizado donde varios agentes pueden participar en la toma de decisiones en función de sus puntos de vista locales.

3.2. Coordinación muti-robot

3.2.1. Coordinación de dos robots

En este parte del capitulo, retomamos la técnica de coordinación de caminos introducida en [33].

La coordinación de camino requiere dos etapas. La primera necesita planificar un camino para cada robot de manera independiente, y la segunda, que es el objeto de esta sección, concierne a la modelización y la exploración del espacio $s1s2$ de coordinación, producto cartesiano de las abscisas curvilíneas $s1$ y $s2$ de los dos robots R_1 y R_2.

3.2.2. Análisis del problema

Designamos respectivamente por $C_{1_{libre}}$ y $C_{2_{libre}}$ los espacios de configuraciones libres de los robots R_1 y R_2 tomados independientemente. Sean dos caminos τ_1 y τ_2, funciones continuas monótonas, uniendo las configuraciones iniciales y finales q_i y q_f de los robots, que verifican:

$$\tau_1 : [0,1] \rightarrow C_{1_{libre}} : s_1 \mapsto q_1 = \tau_1(s_1) \tag{3.1}$$

$$\tau_2 : [0,1] \rightarrow C_{2_{libre}} : s_2 \mapsto q_2 = \tau_2(s_2) \tag{3.2}$$

Sean las funciones suprayectivas y continuas r_1 y r_2 definidas como sigue:

$$r_1 : [0,1] \rightarrow [0,1] : s_1 \mapsto s1' = r_1(s_1) \tag{3.3}$$

$$r_2 : [0,1] \rightarrow [0,1] : s_2 \mapsto s2' = r_2(s_2) \tag{3.4}$$

con $r_i(0) = 0$ y $r_i(1) = 1$ para $i = 1, 2$.

Las funciones r_1 y r_2 son reparametrizaciones de los caminos τ_1 y τ_2 para los cuales se relaja la restricción habitual de inyectividad para permitir al robot de regresar sobre su camino. Se define un **plan** P, inducido por r_1 y r_2 como sigue:

$$P : [0,1] \rightarrow C_{1_{libre}} \times C_{2_{libre}} : u \mapsto (\tau_1(r_1(u)), \tau_2(r_2(u))) \tag{3.5}$$

P determina una ejecución coordinada de dos caminos τ_1 y τ_2 por los robots. Por ejemplo, las reparametrizaciones siguientes:

$$\begin{cases} r_1(u) = 2u, & u \in \left[0, \frac{1}{2}\right] \\ r_1(u) = 1, & u \in \left[\frac{1}{2}, 1\right] \end{cases} \Rightarrow \begin{cases} r_2(u) = 0, & u \in \left[0, \frac{1}{2}\right] \\ r_2(u) = 2(u - \frac{1}{2}), & u \in \left[\frac{1}{2}, 1\right] \end{cases} \tag{3.6}$$

implican que el robot R_1 se desplaza de inicio solo y ejecuta completamente el camino τ_1 y enseguida R_2 recorre τ_2.

Notamos igualmente la no unicidad de los pares de reparametrización culminando en una misma ejecución coordinada de τ_1 y τ_2.

Para que el plan P inducido por r_1 y r_2 sea libre, la condición siguiente debe considerarse:

$$\forall u \in [0,1] : (r_1(u), r_2(u)) \notin CR \tag{3.7}$$

donde CR, se denomina región prohibida (o región en colisión), y se define como sigue:

$$CR = \{(s_1, s_2) \in [0,1]^2 / R_1(\tau_1(s_1)) \cap R_2(\tau_2(s_2)) \neq \emptyset\} \tag{3.8}$$

donde $R_i(\tau_i(s_i))$ denota el espacio ocupado por el robot R_i cuando este se encuentra en la configuración correspondiente a la abscisa curvilínea s_i del camino τ_i. La región obstáculo comprende, en el caso general, los conjuntos conexos de formas arbitrariamente complicados. Notamos sin embargo que las fronteras de CR correspoden al conjunto de las posiciones para las cuales los robots están en contacto.

3.2.3. El diagrama de coordinación

Un procedimiento clásico de aproximación del espacio $s1s2$ consiste en recortar cada camino $(\tau_i)_{i=1,2}$ en una secuencia de sub-caminos w_i determinados por los intervalos $\delta_{i,k_i} = [S_{i,k_i}, S_{i,k_{i+1}}]$ con $K_i \in Sc_i^{2d} = 0...(w_i - 1), s_{i,0} = 0$ y $s_{i,w_i} = 1$. Generalmente, las δ_{i,k_i} se escogen de tal manera que estas determinan los sub-caminos de longitud idéntica. La subdivisión transforma el espacio continuo $s1s2$ en un cuadro de $w_1 \times w_2$ células rectangulares $\delta_{1,k_1} \times \delta_{2,k_2}$.

Denotamos κ_{k_1,k_2} la célula $\delta_{1,k_1} \times \delta_{2,k_2}$. Una célula es "vacía" si las regiones barridas por R_1 y R_2 no se interceptan cuando (s_1, s_2) varia en $\delta_{1,k_1} \times \delta_{2,k_2}$, por ejemplo:

$$\{R_1(\tau_1(s_1))/s_1 \in \delta_{1,k_1}\} \cap \{R_2(\tau_2(s_2))/s_2 \in \delta_{2,k_2}\} = \emptyset$$

En este caso, aseguramos que los dos robots no colisionarán. En el caso contrario, la célula se etiqueta como "llena" Se trata claramente de un modelo conservativo. El espacio $s1s2$ con las células etiquetadas "vacía" o "llena" se llama diagrama de coordinación. La siguiente figura presenta un ejemplo

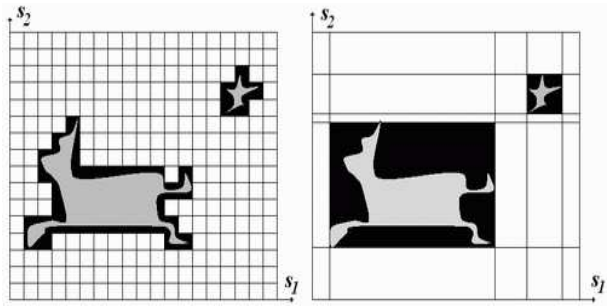


Figura 3.1: Diagramas de coordinación asociados a un muestreo

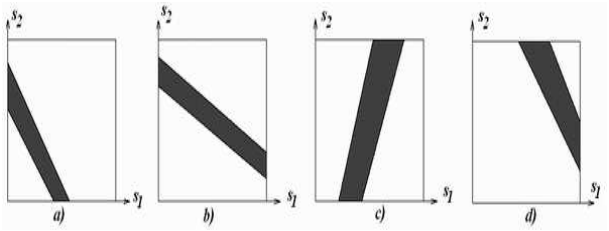


Figura 3.2: Diagramas de coordinación asociados a un muestreo uniforme sobre s_1 y s_2 (a la izquierda) y a las cajas englobantes de las regiones en colision (a la derecha)

Una coordinación existe siempre cuando las posiciones iniciales y finales de los robots estén fuera de las trazas de los otros robots. Es decir, bajo estas hipótesis, la coordinación "trivial" consistente en llevar, uno después del otro, a los robots hasta su posición final es una solución válida.

El método de coordinación fracasa cuando los puntos iniciales $P_I = (0, 0)$ y finales $P_F = (1, 1)$ están en dos componentes conexas diferentes, lo que se traduce por la ausencia de un plan libre en $s1s2$.

La siguiente figura sintetiza los diferentes tipos de fallo que pueden aparecer. En la figura 3.2a ninguno de los robots puede alcanzar su posición final sin entrar en colisión con el otro robot; en 3.2b el robot R_1 si puede pero no el robot R_2 (e inversamente en 3.2c); en 3.2d los dos robots pueden alcanzar la posición final pero de manera concomitante.

Las figuras 3.3 y 3.4 ilustran, bajo la forma de caminos planificados para robots rectangulares, casos concretos para los cuales se produce un fallo.

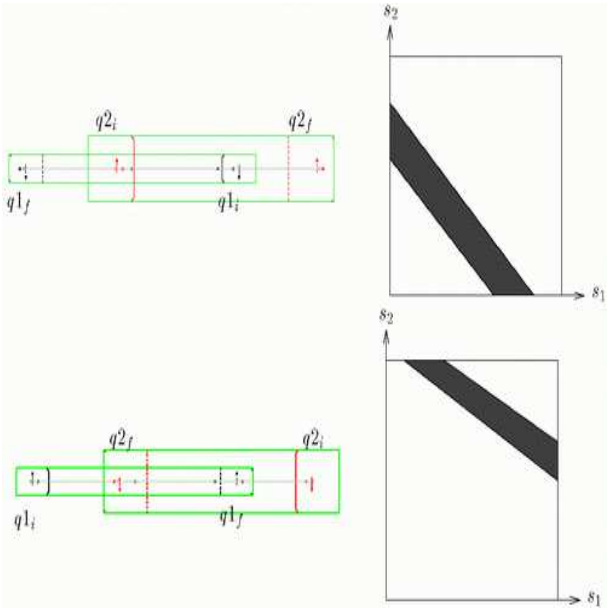


Figura 3.3: Un caso de fallo intrínseco al problema: el cruce sobre una misma vía.

Se trata de inicio de una configuración de cruce ilícito (2 casos posibles) donde los robots llegan en sentido inverso sobre una misma vía (figura 3.3); en seguida de una configuración de rebasamiento ilícito (2 casos posibles) donde los robots están sobre la misma vía en el mismo sentido y donde el que esta en retirada debe pasar ante el otro.

3.2.4. Coordinación de n robots

En esta parte, extendemos la noción de diagrama de coordinación al caso de un número arbitrario de robots ($n > 2$).

Sobre la base de una modelización implícita, introducimos una estructura, denominada "hipercubo", parecida al conjunto de los diagramas de coordinación de dos robots entre n.

3.2.5. Análisis del problema y definiciones

Consideremos un conjunto de n robots $\{R_i\}_{i=1..n}$ que han planificado su camino $(\tau_i)_{i=1..n}$. Sea $S_i = [0, 1]$ el espacio normalizado de las abscisas

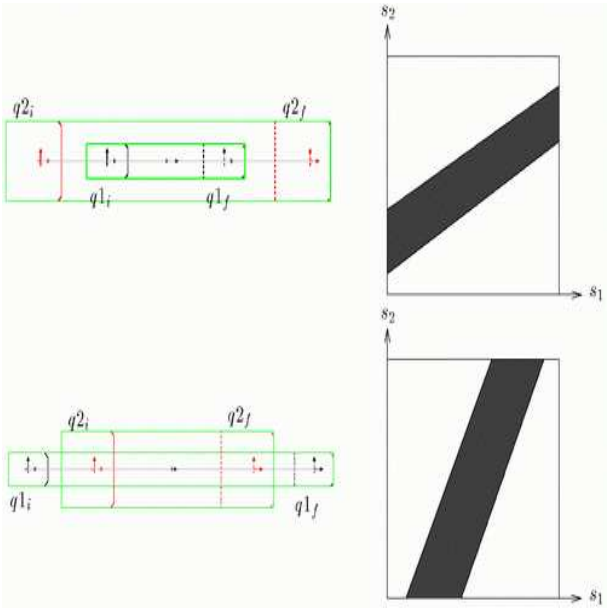


Figura 3.4: Otro caso de fallo intrínseco al problema: el rebasamiento sobre una misma vía.

curvilíneas del robot R_i. Se define el espacio de coordinación de caminos S como el producto cartesiano de las abscisas curvilíneas de los n robots, es decir $S = S_1 \times S_2 \times \ldots \times S_n$.

La región prohibida CR_n está definida como el conjunto de las $(s_1, ..., s_n)$ en S asociadas a las colisiones entres los robots, es decir:

$$CR_n = \{(s_1, ..., s_n) \in S/\ \exists(j,k), 1 \leq j, k \leq n, j \neq k \text{ tales que}$$

$$R_j(\tau_j(s_j)) \cap R_k(\tau_k(s_k)) \neq \emptyset\}$$

La existencia de una solución al problema de coordinación de movimientos de n robots esta caracterizada por la existencia de un camino en S evitando los obstáculos CR_n.

La definición anterior traduce la estructura cilíndrica de la región prohibida CR_n, también consideramos que es suficiente de que una colisión entre dos robots del sistema para la cual una célula dejaría de ser libre independientemente de las posiciones de los otros robots.

En el caso general, la región prohibida, que tiene una dimensión n, esta definida por superficies complejas, difíciles de calcular.

3.2.6. Modelización del espacio de coordinación en hipercubo

Dada la complejidad de CR_n, sería interesante aproximar el espacio de coordinación S por medio de una estructura más simple y manejable.

Así, se define una partición de S. Esta está basada en un reparto de los caminos τ_i en sub-caminos indexados por el conjunto $I_i = \{ 0, 1,. . ., C_{\text{máx}_i} \}$.

Una función $abs\_curve_i : I_i \rightarrow (R^+)^2$ permite recuperar las abscisas curvilíneas de inicio y de final de un sub-camino, a partir de su índice. Por ejemplo, si se subdivide el camino en dos porciones de la misma longitud, se tendrá: $I_i = 0, 1, abs\_curve_i(0) = (0, 1/2)$ y $abs\_curve_i(1) = (1/2, 1)$.

La estructura obtenida que designamos en adelante por el término de hipercubo H, esta formado de "células" o "hiperparalelepipedos" de dimensión n correspondiente al estado de avance del sistema multi-robot, relativamente en el particionamiento fijado sobre cada camino.

Hasta este momento, hacemos la elección de aproximar el espacio de coordinación real S por medio del hipercubo en el cual se considerara en adelante que la célula constituye una unidad elemental de coordinación.

Esto nos lleva a definir de la manera siguiente la región prohibida CR_H:

$$CR_H = \{(c_1, ..., c_n) \in I_1 \times ... \times I_n\}$$
$$\exists (j, k), 1 \leq j, k \leq n, j \neq k, \exists (s_j, s_k) \in abs\_curve(c_j) \times abs\_curve(c_k)$$
$$\text{tales que } R_j(\tau_j(s_j)) \cap R_k(\tau_k(s_k)) \neq \emptyset\}$$

De manera evidente, se tiene que $CR_n \subseteq CR_H$.

La resolución del problema de coordinación en el hipercubo se compara con la búsqueda de un camino entre células conexas libres ligando los estados multi-robot inicial $C_n^I = (0, 0, ..., 0)$ y final $C_n^F = (C_{max_1}, C_{max_2}, ..., C_{max_n})$.

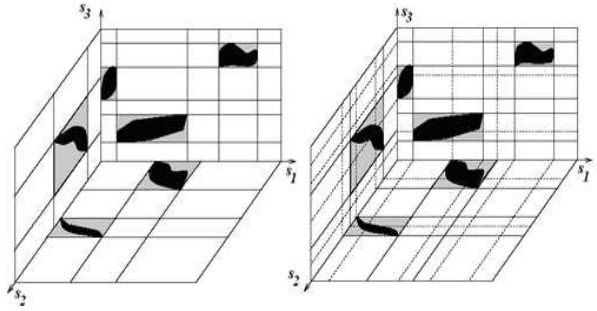


Figura 3.5: Caso de robots: representación del conjunto de diagramas de coordinación de dos robots (en la izquierda) y el hipercubo de dimensión 3 asociado (en la derecha).

La figura 3.5 presente, a la izquierda, el conjunto de los diagramas de coordinación de dos robots, y a la derecha, una representación del hipercubo asociado a la coordinación de tres robots.

3.3. Planificación multi-robot con RRT

El trabajo realizado se enfoca a la planificación de trayectorias para múltiples robots, más específicamente, el problema se puede expresar como sigue: dado un espacio de trabajo, un conjunto de obstáculos poligonales dentro el espacio de trabajo, un conjunto de robots poligonales (cada uno con sus correspondientes configuraciones inicial y final). Planificar una trayectoria para cada robot utilizando el método de los árboles aleatorios de exploración rápida (RRT) en los que cada robot obedece ciertas restricciones de no-holonomía y evita colisiones (con los obstáculos y con otros robots).

El esquema adoptado fue:

i) Planificar una trayectoria para cada robot utilizando el planificador RRTExtExt[1].

ii) Construir el diagrama de coordinación para el número de robots considerados.

[1]Este planificador se utilizó porque considera las restricciones diferenciales de cada robot.

Suponemos que la planificación de trayectorias de los robots puede ser realizada fuera de línea, para evitar las colisiones con los obstáculos estacionarios pero no así con los demas robots. Estas son las metas que esperamos para nuestra coordinación de trayectorias:

- Es posible realizar la planificación de la trayectoria para cada robot independientemente.
- El resultado de las trayectorias debe garantizar que los robots efectúen sus metas.
- Debe ser posible ejecutar las trayectorias en un tiempo preciso de coordinación entre los robots.
- La seguridad de los robots no debe depender de la trayectoria de control actual de los robots individuales.

Asumimos que se puede estimar el tiempo de ejecución requerido para cada segmento de la trayectoria. Sobre estas suposiciones, observamos que el problema de la coordinación de trayectorias, es un problema de sincronización.

Con el enfoque RRT es posible utilizar una planificación centralizada, pero el tiempo de ejecución es mucho mayor que en el caso desacoplado. El algoritmo de coordinación sacrifica completez.

3.3.1. Resultados experimentales

A continuación se presentan los resultados obtenidos para la planificación multi-robot con el planificador RRTExtExt.

El enfoque desacoplado es un enfoque de dos fases, en la primera fase, se genera una trayectoria libre de colisiones para cada robot, considerando solamente los obstáculos en el ambiente e ignorando a los otros robots; en la segunda fase, llamada el ajuste de la velocidad, las velocidades relativas de los robots a lo largo de sus respectivas trayectorias se seleccionan para evitar colisiones entre ellos. El ajuste de la velocidad consiste en una búsqueda en un espacio de coordinación. Este enfoque nos lleva a búsquedas en espacios de mas baja dimensión, si lo comparamos con el enfoque centralizado.

Los resultados mostrados en esta subsección no son exhaustivos y con detalle, debido a que el interés primordial del trabajo se orienta hacia la

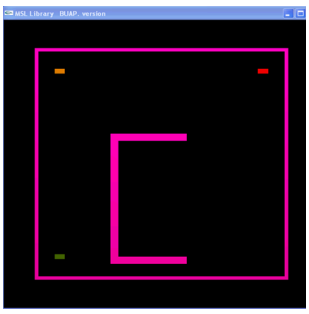


Figura 3.6: Ambiente con 3 robots tipo carro.

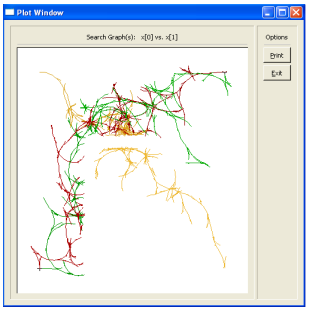


Figura 3.7: Progreso de la planificación RRTExtExt multi-robot

exploración cooperativa. Pero esta parte sirvió de base para poder alcanzar el objetivo principal de la tesis.

La imagen 3.6 muestra el ambiente de trabajo sobre el cual se realizará la planificación de 3 robots tipo carro.

Como se mencionó, cada robot construye su propio árbol RRT intentando extenderlo hacia su configuración final establecida, ver figura 3.7, al mismo tempo, en cada nodo del árbol se construye el diagrama de coordinación que servirá para evitar colisión entre los robots en cada paso de su trayectoria 3.8.

Una vez terminado el proceso, cada robot ejecuta su camino desde la configuración inicial hasta la configuración final.

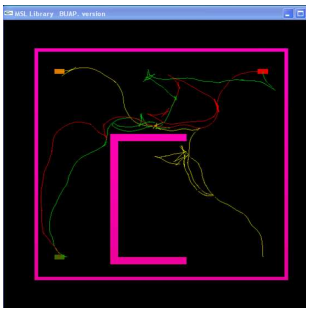
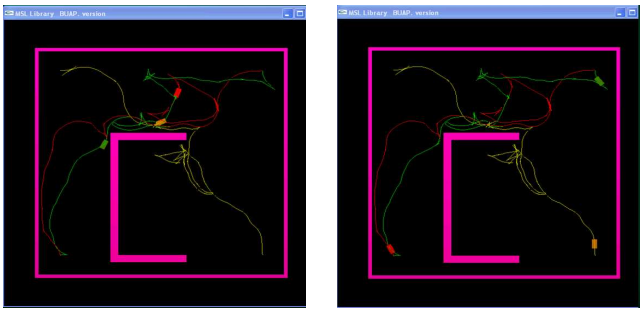


Figura 3.8: Caminos encontrados por los robots con la planificación RRT-ExtExt multi-robot



a) b)

Figura 3.9: Ejecución de los caminos encontrados por el planificador RRT-ExtExt multi-robot.

Capítulo 4

Método SRT para ambientes desconocidos

Oriolo, Vendittelli, Freda y Troso presentan en [34] un método de exploración de ambientes desconocidos usando sensores para un robot móvil. El método se basa en la generación de una estructura de datos incremental aleatoria llamada *árbol aleatorio de exploración usando sensores* (SRT, del inglés Sensor-Based Random Tree), la cual representa un roadmap del área explorada asociada a una región segura. Este método esta inspirado en los árboles aleatorios de exploración rápida (RRTs).

4.1. Método SRT

La exploración de ambientes desconocidos puede considerarse como un problema fundamental para los robots móviles, dado que involucra todas las capacidades fundamentales de estos sistemas, por ejemplo, la percepción, la planificación, la localización y la navegación.

Una definición ampliamente aceptada sobre la exploración es "el acto de moverse a través de un ambiente desconocido mientras se construye un mapa que pueda utilizarse para subsecuentes navegaciones". El rendimiento de las estrategias de exploración debe ser valorado en base a la calidad del mapa obtenido y del tiempo necesario para construirlo. Muchas de las técnicas existentes caen dentro de la clase de exploración basada en fronteras. La lógica de este enfoque es que el robot debe moverse hacia los límites (la frontera) de las áreas seguras exploradas y del territorio desconocido para

maximizar la información obtenida a través de nuevas percepciones.

Es interesante adoptar una perspectiva más general dentro de la Inteligencia Artificial, de acuerdo con la cual, la exploración es "el proceso de seleccionar acciones en aprendizaje activo". En el paradigma de aprendizaje activo, los datos de entrenamiento son obtenidos como resultado de las acciones del aprendiz. En particular, cuando el robot recopila información sobre su ambiente mediante movimiento y sensado, se considera como un caso de aprendizaje activo de orden–sensitivo, por que el flujo de datos es resultado de todas las acciones pasadas del robot. El problema central de la exploración es como seleccionar la siguiente acción. La exploración basada en fronteras se logra cuando el criterio es la maximización de la utilidad de las acciones.

El método de exploración implementado se basa en la generación aleatoria de configuraciones en un área segura local detectada por los sensores. Se crea una estructura de datos llamada *árbol aleatorio de exploración usando sensores (SRT)*, el cual representa el roadmap del área explorada asociado a una región segura (RS). Cada nodo del SRT consiste de una configuración libre y su región segura local (RSL) asociada; la RS es simplemente la unión de todas las RSLs pertenecientes al árbol. La RSL es una estimación del espacio libre circunvecino a una configuración dada del robot; en general, su forma dependerá de las características del sensor pero también puede reflejar diferentes posturas de percepción.

El método de exploración SRT, se presenta bajo la suposición de una perfecta localización del robot, provista por otro módulo. Esto puede suceder en ocasiones (por ejemplo, un sistema GPS usado en misiones planetarias), pero no podemos omitir que tal suposición a menudo es ilógica en ambientes desconocidos y no estructurados.

4.2. Exploración de ambientes desconocidos con el método SRT

El método SRT se introdujo con ciertas consideraciones sobre el robot y el ambiente de trabajo. Más adelante se describe el método de exploración desde un punto de vista general, es decir, independiente de una estrategia de percepción particular. Finalmente, se presenta la variante adoptada y los resultados obtenidos por medio de la herramienta de simulación desarrollada.

4.2.1. Hipótesis de trabajo

El robot debe explorar un espacio de trabajo, es decir, un ambiente con obstáculos. Siguiendo las siguientes suposiciones:

1. El espacio de trabajo es plano, es decir, $\mathbb{R}^2$ o un subconjunto de $\mathbb{R}^2$.
2. El robot es libre de trasladarse en cualquier dirección (un robot holonómico o robot de vuelo libre). De esta forma, el espacio de configuraciones es una copia del espacio de trabajo con los obstáculos crecidos tanto como lo requiera el tamaño del robot.
3. El robot siempre conoce su configuración q, (localización perfecta).
4. El robot esta equipado con un sistema de sensores el cual provee en cada configuración q la estimación del espacio libre circunvecino. Esta estimación, llamada Región Segura Local en q, se denota por S.
5. Una pequeña región del espacio físico circundante al robot y al sistema de sensado en la configuración inicial es libre, este requerimiento es importante porque de lo contrario ningún movimiento puede ejecutarse después del primer sensado.

4.2.2. Algoritmo SRT

El método construye una estructura de datos llamada árbol aleatorio de exploración usando sensores (SRT), que puede considerarse como una variación del árbol aleatorio de exploración rápida (RRT). Así como el RRT, el SRT es un árbol que representa el roadmap del espacio de configuraciones libre. Cada nodo del SRT consiste de una configuración q libre de colisión que ha alcanzado el robot, junto con la descripción de la región segura local S circundante a q percibida por los sensores. El árbol se construye gradualmente, extendiendo la estructura hacia direcciones seleccionadas aleatoriamente de tal manera que la nueva configuración (y el camino que lleva a ella) este contenida en la región segura local. El algoritmo que implementa el método SRT se describe en la figura 4.1.

En cada iteración k del algoritmo, se efectúa un proceso de percepción (es decir, sensado del ambiente y recopilación de datos), para obtener la región S que estima el espacio libre circundante al robot en la configuración actual, q_{act}. Un nuevo nodo, que contiene la configuración q_{act} y su RSL asociada, se agrega al árbol $\mathcal{T}$. La forma de representar S en la estructura

CONSTRUIR_SRT($q_{inic}, K_{max}, I_{max}, \alpha, d_{min}$)

1 $q_{act} = q_{inic}$;
2 **para** *k=1* **a** K_{max}
3 $S \leftarrow$ PERCEPCION(q_{act});
4 AGREGA($\mathcal{T}, (q_{act}, S)$);
5 $i \leftarrow 0$;
6 **repetir**
7 $\theta_{rand} \leftarrow$ DIR_ALEATORIA;
8 $r \leftarrow$ RADIO(S, θ_{rand});
9 $q_{cand} \leftarrow$ DESPLAZAR($q_{act}, \theta_{rand}, \alpha \cdot r$);
10 $i \leftarrow i + 1$;
11 **hasta que** (VALIDA($q_{cand}, d_{min}, \mathcal{T}$) o $i = I_{max}$)
12 **si** VALIDA($q_{cand}, d_{min}, \mathcal{T}$) **entonces**
13 MOVER_A(q_{cand});
14 $q_{act} \leftarrow q_{cand}$;
15 **sino**
16 MOVER_A($q_{act}.padre$);
17 $q_{act} \leftarrow q_{act}.padre$;
18 **Regresa** $\mathcal{T}$;

Figura 4.1: Algoritmo básico de construcción del SRT

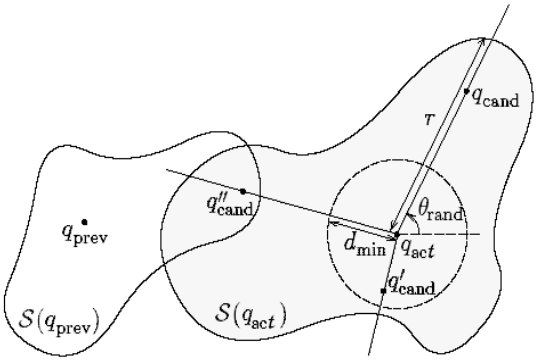


Figura 4.2: Generación de configuraciones candidatas con el método SRT. En este caso, q_{cand} es valida, mientras q' y q'' no lo son, la primera se encuentra a una distancia menor a d_{min} de q_{act} y q'' se ubica en la región segura local de otro nodo.

SRT depende de la estrategia de percepción: en general, podría usarse una descripción algebraica de sus limites.

En el punto de la configuración actual, la función DIR_ALEATORIA genera una dirección aleatoria de exploración θ_{rand} y la función RADIO calcula el radio r de S en la dirección θ_{rand}, ver la figura 4.2. Una nueva configuración candidata q_{cand} se determina tomando un paso de longitud $\alpha \cdot r$ en dirección a θ_{rand}. La constante $\alpha < 1$ garantiza que q_{cand} se encuentra en el área segura S y puede alcanzarse a través de un camino contenido en S; valores próximos a 1 incrementan la capacidad de exploración del algoritmo, mientras que valores más pequeños aumentan el margen de seguridad.

Una vez generada q_{cand} de forma aleatoria en la región segura S, pasa por un proceso de validación efectuado por la función VALIDA. Como se muestra en la figura 4.2, q_{cand} (i) debe estar alejada de q_{act} a una distancia mayor a una distancia mínima prefijada d_{min} y (ii) no debe situarse en la región segura local de otra configuración previa en $\mathcal{T}$. Si la validación tiene éxito, el robot se mueve a q_{cand} y el ciclo se repite. De lo contrario, el algoritmo genera otras configuraciones aleatorias desde q_{act} hasta encontrar una configuración valida o exceder un número máximo de intentos, I_{max}. En el último caso, el robot regresa al nodo padre de q_{act}, donde se ejecuta nuevamente el ciclo. Típicamente, cuando el espacio libre ha sido explorado completamente, el algoritmo fallará en encontrar una nueva dirección de

exploración y el robot hará un proceso automático de retorno a la configuración inicial.

Una comparación del método SRT con el planificador RRT origina los siguientes comentarios:

- En comparación con el RRT, la estructura SRT es una árbol con aristas de longitud variable, dependiendo del radio de la RSL en dirección a θ_{rand}. Por lo tanto, durante la exploración el robot recorrerá longitudes más largas en regiones con pocos obstáculos y más pequeñas en virtud de objetos a su paso. También, no es necesario un chequeo de colisiones ya que las configuraciones candidatas son generadas dentro del área segura.

- Desde el punto de vista de la exploración, el método SRT es substancialmente en profundidad. Dado que el árbol se expande a partir de q_{act}, la posición actual del robot; en contraste con la expansión en anchura típica de los RRT puros, los cuales, no se emplean para exploración usando sensores. La introducción del mecanismo de retroceso es una consecuencia de la naturaleza del recorrido en profundidad del algoritmo SRT.

- El método STR mantiene algunas de las características mas importantes del RRT, como ser conveniente para espacios de configuraciones de altas dimensiones y fácilmente modificable tanto para restricciones holonómicas como no holonómicas.

4.3. Exploración con SRT-Radial

Como se mencionó, la forma de la región segura local S refleja las características del sensor así como la técnica de percepción adoptada. A su vez, la estrategia de exploración estará fuertemente afectada por la forma de S. En [34] se presenta una variante del método llamada SRT-Star, la cual involucra una estrategia de percepción que toma completamente en la información reportada por el sistema de sensado, además de explotar la información proporcionada por los sensores en todas direcciones. En SRT-Star, S es una región con forma similar a una estrella debido a la unión de varios 'conos' con diferentes radios cada uno, ver figura 4.3. El radio del i-ésimo cono η_i es la distancia mínima entre la distancia del robot al obstáculo más cercano o el rango máximo medible con los sensores. Por lo tanto, para poder calcular

r, la función RADIO primero debe identificar a que cono corresponde θ_{rand}.

Por el contrario, bajo la variante implementada en este proyecto la forma de S, idealmente, en ausencia de obstáculos, es circular por lo que es innecesaria la identificación del cono. A esta variante le denominamos SRT-Radial, en la cual una vez generada la dirección de exploración θ_{rand}, la función RADIO traza un rayo desde la ubicación actual hacia el borde de S, la porción comprendida dentro de S representa el radio en la dirección θ_{rand}, ver figura 4.6. Por tanto, en presencia de obstáculos la forma de S se deforma y para diferentes direcciones de exploración las longitudes de los radios varían.

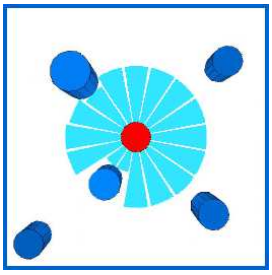


Figura 4.3: Región segura local S obtenida con la estrategia de percepción SRT-Star. Note que la extensión de S en algunos conos es reducida por el rango de alcance del sensor.

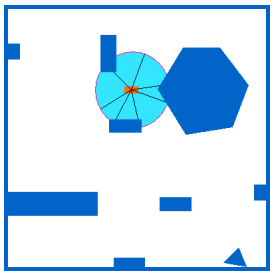
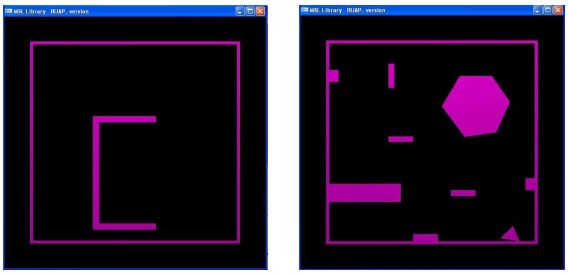


Figura 4.4: Diferentes radios obtenidos en la región segura local S con la estrategia de percepción SRT-Radial y la aplicación de la librería GPC.

4.3.1. Comparativo entre SRT-radial y SRT-Star

En la sección 4.3 hablamos de dos strategias de exploración, SRT-Radial y SRT-Star, ambas fueron comparadas mediante simulación, usamos los mismos ambientes para probar la eficiencia del método SRT-Radial sobre el metodo SRT-Star. El algoritmo SRT fue provado en dos ambientes con los mismos valores para $\mathcal{K}_{max}$, $\mathcal{I}_{max}$, α, d_{min}. Las figuras 4.7 y 4.9 presentan las regiones exploradas y las regiones de seguridad obtenidas con la estrategia SRT-Star para los ambientes mostrados en la figura 4.5, las figuras 4.6 y 4.8 muestran las regiones exploradas y las regiones de seguridad con la

estrategia SRT-Radial. El número final de nodos en el árbol y el tiempo de ejecución son mucho más pequeños con es SRT-radial que con el SRT-star.



a)Ambiente 1 b)Ambiente 2

Figura 4.5: Ambientes usados en el comparativo entre SRT-radial y SRT-Star

4.4. Preliminares del problema de exploración cooperativa

El método exploración cooperativa basada en SRT es presentado bajo las siguientes suposiciones.

1. Todos los robots en el equipo son idénticos.
2. Los robots se mueven en un área de trabajo plana, es decir, $\mathbb{R}^2$ o un subconjunto conectado de este.
3. Cada robot es un polígono que puede moverse en cualquier dirección. La configuración q del robot es por lo tanto la posición del centro del polígono.
4. Cada robot conoce su configuración.
5. Cada robot es equipado con un sistema sensorial el cual provee la región segura local (RSL) $S(q)$, una (posiblemente conservadora) descripción del espacio libre que rodea al robot en q. La RSL es una estrella-formada [1], subconjunto de $\mathbb{R}^2$, cuyo radio máximo es limitado

[1]Esto significa que $S(q)$ es homeomorfica al disco unitario cerrado y el segmento de línea conectando el centro del robot a cualquier punto en S esta completamente contenido en S.

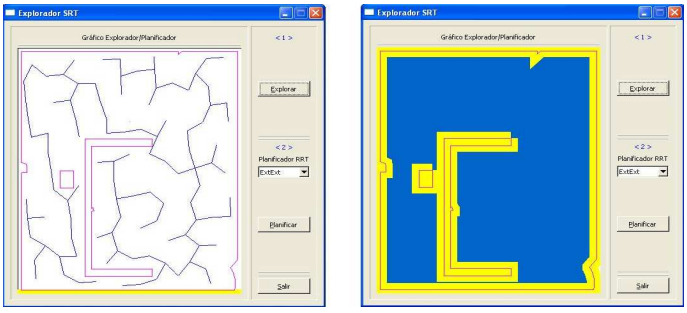


Figura 4.6: A la izquierda ,tenemos el árbol de exploración obtenido con la estrategia SRT-Radial, a la derecha, la región segura y la banda de seguridad. Tiempo de exploración 96.156 seg. y 106 nodos

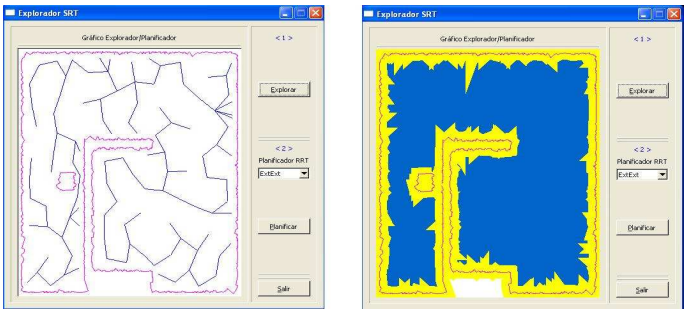


Figura 4.7: A la izquierda ,tenemos el árbol de exploración obtenido con la estrategia SRT-Star, a la derecha, la región segura y la banda de seguridad. Tiempo de exploración 436.725 seg. y 110 nodos

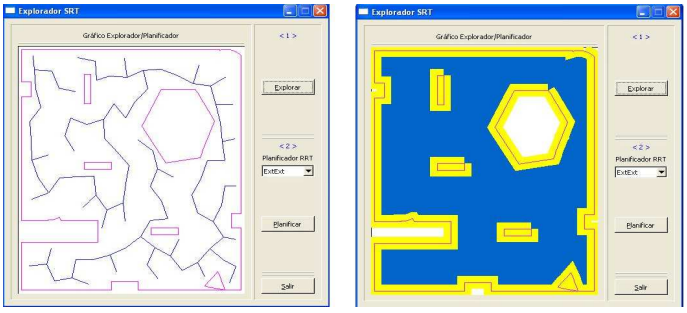


Figura 4.8: A la izquierda ,tenemos el árbol de exploración obtenido con la estrategia SRT-Radial, a la derecha, la región segura y la banda de seguridad. Tiempo de exploración 80.625 seg. y 97 nodos

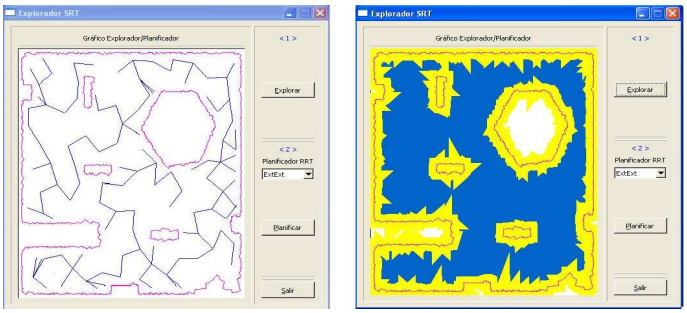


Figura 4.9: A la izquierda ,tenemos el árbol de exploración obtenido con la estrategia SRT-Star, a la derecha, la región segura y la banda de seguridad. Tiempo de exploración 2542.53 seg. y 118 nodos

por el rango de percepción del robot R_p. Cada RSL es almacenada en la memoria del robot con una estampa de tiempo asociada, es decir, en el instante de tiempo en que fue recolectada. En este trabajo de tesis, utilizamos la estrategia SRT-Radial que proporciona mejores resultados que la original versión SRT-Star.

6. Cada robot conoce su número de Identificación.

7. Cada robot puede difundir con un rango de comunicación R_c la información almacenada en su memoria (o porciones relevantes de ella) en cualquier momento. El número de identificación de robot es incluido en la cabecera de cualquier transmisión. El robot esta siempre abierto para recibir comunicación de otros robots dentro de Rc.

Muchas de estas suposiciones son solo tomadas por simplicidad y pueden ser relajadas. Primero, el robot no necesita ser idéntico: por ejemplo, el sistema sensorial puede diferenciarse muy bien en naturaleza y el rango de percepción. La hipótesis de un espacio de trabajo planar es obviamente no restrictiva: los ambientes 3D son perfectamente admisibles mientras el sistema sensorial permita la reconstrucción de una RSL planar para planificar los movimientos del robot. La hipótesis 3 es solo tomada para facilidad de presentación: el método propuesto es fácilmente aplicable a robots con forma arbitraria, por ejemplo sujeto a restricciones no-holonómicas. En cuanto a la hipótesis 4, puede ser eliminada incorporando un modulo de localización en el método SRT como se describe en [12]. Ver la sección de conclusiones para comentarios adicionales.

4.5. Exploración cooperativa basada en SRT

El diseño de las estrategias de exploración cooperativa proviene de la paralelizacion del método SRT básico para un solo robot descrito en la sección 4.2.2, al cual le agregamos esencialmente tres funcionalidades:

i) cooperación, para incrementar eficiencia;
ii) coordinación, para evitar conflictos; y
iii) comunicación como herramienta fundamental para cooperar y coordinar.

EXPLORACION_COOPERATIVA_BASADA_EN_SRT(q_{inic})
1 $\mathcal{T}$.Inic(q_{inic})
2 CONSTRUIT_SRT($q_{inic}, \mathcal{T}$)
3 APOYAR_OTROS(q_{inic})

Figura 4.10: Pseudocodigo del algoritmo exploración cooperativa basada en SRT

En el método de exploración cooperativa basado en SRT, cada robot construye uno o más mapas parciales del ambiente, organizados en una colección de árboles aleatorios de exploración basados en sensores (SRT). Cada nodo de un SRT representa una configuración q la cual fue visitada por al menos un robot, junto con la región segura local asociada $S(q)$. Un arco entre dos nodos representa un camino libre de colisión entre las dos configuraciones. El árbol es incrementalmente construido extendiendo la estructura en la dirección más prometedora vía un mecanismo aleatorio dirigido. La presencia de otros robots en la vecindad es tomada en cuenta en esta etapa con el objetivo de maximizar la información obtenida y garantizar evitar colisiones.

Los pasos básicos de la ejecución del algoritmo de exploración en cada robot se muestran en la figura 4.10. En la primera fase, el procedimiento CONSTRUIR_SRT es ejecutado: el robot construye su propio SRT, $\mathcal{T}$ se establece como raíz en su configuración inicial q_inic. CONSTRUIR_SRT termina cuando el robot es incapaz de expandir $\mathcal{T}$. En este punto, el robot ejecuta el procedimiento APOYAR_OTROS, contribuyendo a la expansión de los SRTs que han sido inicializados por otros robots (ver la sección 4.7). Cuando el procedimiento APOYAR_OTROS termina, el robot ha regresado a la raíz de su propio árbol y su exploración termina.

En el algoritmo de exploración antes mencionado, solo la percepción, la planificación y las funcionalidades del movimiento se hacen explicitas. Claramente, un hilo paralelo de comunicación se ejecuta en cada robot, no descrito aquí por simplicidad (ver [11]). En principio, sin embargo, dos enfoques son posibles. En el primero, el cual es elegido para nuestra presentación, cada robot continuamente difunde su conocimiento, incluyendo lo derivado de otros robots: esto significa que la información relevante para la cooperación y la coordinación con los otros agentes incide dentro del rango de comu-

```
CONSTRUIR_SRT(q_inic, T)
1   q_act ← q_inic;
2   Hacer
3      CONSTRUIR_Y_ESPERAR_GPA();
4      S(q_act) ← PERCIBIR(q_act);
5      AGREGAR(T, (q_act, S(q_act)))
6      G ← CONSTRUIR_GEA();
7      F(q_act) ← FRONTERA_LOCAL(q_act, S(q_act), T ∪ T_i);
8      q_target ← PLANIFICADOR_ALEATORIO(q_act, F(q_act));
9      if ||G|| > 1
10        (G_f, G_u) ← CHECA_FACTIBILIDAD(G);
11        if G_u ≠ ∅
12        q_target ← COORDINA(G_f, G_u);
13     q_act ← MOVER_A(q_target);
14     salir ← (q_act = q_inic) y (F(q_act) = ∅);
15  mientras salir = 0 ;
```

Figura 4.11: Pseudocodigo del procedimiento CONSTRUIR_SRT

nicación R_c que esta inmediatamente disponible para el robot. La segunda solución, apunta a reducir el consumo del ancho de banda, es decir, para establecer comunicación robot a robot solo cuando se necesite.

4.6. Procedimiento CONTRUIR_SRT

El pseudocodigo del procedimiento CONTRUIR_SRT se muestra en la figura 4.11. Primero damos un comentario rápido de los pasos principales, y entonces discutimos su estructura con algo de detalle en el resto de la sección.

En cada iteración de la construcción del SRT, el robot usa toda la información disponible (la propia parcialmente recolectada y la parcialmente adquirida mediante comunicación con los otros robots) para identificar el grupo de agentes enganchados (GEA), es decir, los otros agentes del equipo con los que la cooperación y coordinación son adecuados. Esto es alcanzado por la primera construcción del grupo de agentes Pre-enganchados (GPA), es decir, los robots que son candidatos a pertenecer al GEA, y sincronizados con ellos (CONSTRUIR_Y_ESPERAR_GPA). Entonces, el robot recopila datos

a través de sus sistemas sensoriales, contruye la actual RSL (PERCIBIR) y por consiguiente actualiza su propio árbol $\mathcal{T}$. La GEA actual puede ahora ser construida (CONSTRUIR_GEA). En este punto el robot procesa su frontera local (la porción de su limite actual RSL conduce a áreas que aparecen aun no exploradas) en la base de $\mathcal{T}$ así como cualquier otro árbol $\mathcal{T}_i$ adquirido a través de la comunicación y almacenado en su memoria (FRONTERA_LOCAL).

Si la frontera local es no vacía, el robot genera una configuración aleatoria contenida en la actual RSL y dirigida hacia la frontera local; si no, la configuración objetivo se fija al nodo padre (vuelta hacia atrás) (PLANIFICADOR_ALEATORIO). Si la GEA es compuesta solo por el mismo robot, el robot se mueve directamente a su objetivo. De otro modo, los caminos anticipados del robot en la GEA son checadas para colisiones mutuas, y por consiguiente clasificadas dentro de caminos factibles y no factibles (CHECAR_FACTIBILIDAD). Si el subconjunto G_u de robots con caminos no factibles es no vacía, una fase de coordinación toma lugar la cual quizás confirma o modifica el objetivo actual del robot (COORDINAR). En particular, el movimiento del robot puede ser prohibido simplemente reajustando el objetivo a la configuración actual. La funcion MOVER_A entonces transfiere al robot al objetivo (cuando esto es diferente de q_{act}).

El ciclo principal es repetido hasta la condición de salida verificado en la línea 15: El robot es incapaz de expandir el árbol $\mathcal{T}$ (no hay fronteras locales restantes) y por lo tanto tiene que retroceder a la raíz de su SRT.

4.6.1. Construcción GPA/GEA

Al inicio de CONSTRUIR_SRT, los robots están inmóviles y necesitan identificar a los otros robots cuyas RSLs se puedan solapar con la suya, para cooperar (optimizar la exploración) y coordinar (evitar conflictos) con ellos. Los otros robots pueden estar inmóviles también (en este caso, sus objetivos coinciden con la configuración actual) o moviéndose al objetivo; por lo tanto, una fase de sincronización es necesaria.

Decimos que dos robots son GPA-acoplados si la distancia entre sus configuraciones objetivo esta a lo mas $2R_p$, es decir, dos veces el rango de percepción del sistema sensorial. La GPA del robot es entonces construida agrupando todos los robots para la cual esta puede ser conectada a través de una cadena de GPAs acopladas (Ver la figura 4.12, izquierda). Para al-

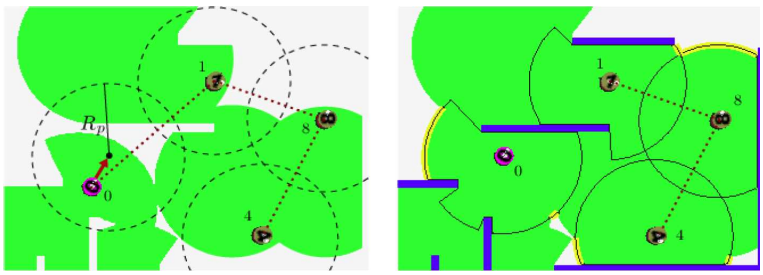


Figura 4.12: Un ejemplo de construcción GPA/GEA. Izquierda: La GPA del robot 1 consiste de los robots 0,1,4,8: el robot 0 esta aun en movimiento hacia su punto objetivo, mientras que los robots 1,4,8 están inmóviles. La áreas de percepción de los robots (anticipada en el caso del robot 0) traslapadas en pares. Derecha: una vez que la RSL ha sido procesada, solo los robots 1,4,8 pertenecen a la GEA del robot 1 desde que sus RSLs se traslapan en pares.

canzar la sincronización, la GPA es calculada y actualizada hasta que todos sus miembros están inmóviles (CONSTRUIR_Y_ESPERAR_GPA).

El rango de comunicación R_c claramente juega un papel importante en la construcción del GPA, para este proyecto se han considerado dos casos en cuanto al rango de comunicación:

i) Rango de comunicación limitado. Desde que la máxima distancia entre el robot y cualquier otro robot con el que este es GPA-acoplado es $3R_p$ (el otro robot puede aun estar moviéndose a su objetivo, el cual sin embargo no puede estar mas lejos que R_p de la configuración actual), es suficiente asumir $R_c \geq 3R_p$ para garantizar que la GPA cuenta para todos los robot que son candidatos para pertenecer a la GEA.

ii) Rango de comunicación ilimitado. Dada la naturaleza de esta comunicación el robot conoce siempre el estado de los otros agentes y por tanto sabrá que robots son candidatos para ser GPA-acoplados, la distancia para ser GPA-acoplados como en (i) se mantendrá en $3R_p$, de esta manera al igual que en el anterior se garantiza que la GPA cuenta para todos los robot que son candidatos para pertenecer a la GEA

Una vez que el robot se ha sincronizado con su GPA, percibida la RSL y actualizado su SRT, construye la GEA, es decir, los robots con los cuales la cooperación y coordinación son actualmente necesarias. Si definimos dos

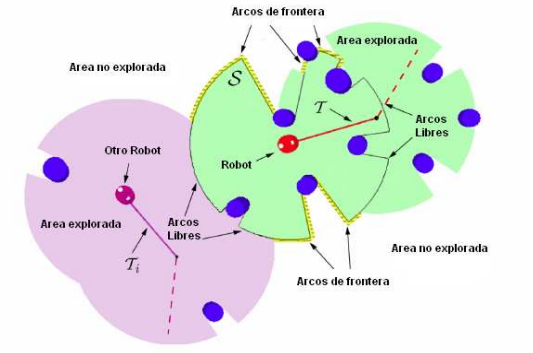


Figura 4.13: Definición de frontera, arcos libres y de obstáculos en la región segura local S.

robots para ser GEA-acoplados cuando sus RSLs se traslapan, la GEA del robot (ver la figura 4.12, derecha) esta compuesta por todas las GPA de robots a las que pueden ser conectadas por una cadena de GEA acopladas (CONSTRUIR_GEA). La fase de sincronización garantiza que todos los robots en la GPA están inmóviles cuando la GEA es procesada. La GEA es una estructura simétrica, esto es lo mismo para todos los robots en el grupo.

La GEA es una piedra angular de nuestro método, como esto identifica un grupo de robots que, en vista de su vecindad, acuerdan espontáneamente cooperar y coordinar con cada uno sobre una base temporal. Una característica importante del GEA, es que esta puede ser construida con un rango limitado de comunicación ($R_c \geq 3R_p$).

4.6.2. Extracción de frontera

Una vez que el robot se ha sincronizado con su GEA, el procedimiento FRONTERA_LOCAL es invocado. Esto procesa la porción de los limites de la RSL $S(q_{act})$ conduciendo a áreas que aparecen no exploradas de acuerdo a la información disponible. Para este fin, el robot usa su propio árbol $\mathcal{T}$ así como cualquier otro árbol $\mathcal{T}_i$ almacenado en su memoria y recibido por comunicaciones presentes o pasadas.

Para encontrar direcciones prometedoras en $S(q_{act})$, su frontera es dividida en arcos de obstáculos , libres y de frontera (ver figura 4.13). Los arcos de obstáculo son localizados a lo largo de las direcciones donde los obstáculos

```
PLANIFICADOR_ALEATORIO(q_act, F(q_act))
1  if F(q_act) ≠ ∅
2     θ_rand ← DIRECCION_ALEATORIA(F(q_act));
3     q_target ← DESPLAZARSE(q_act, θ_rand);
4  si no
5     q_target ← q_act.padre;
6  Regresar q_target
```

Figura 4.14: Pseudocodigo del planificador aleatorio

han sido detectados, mientras que los arcos libres caen dentro de otra RSL conocida (cualquiera perteneciente a $\mathcal{T}$ o a la $\cup\mathcal{T}_i$). Cualquier arco que no sea de un obstáculo no libre es un arco de frontera, y por construcción identifica los limites entre $S(q_{curr})$ y una región no explorada. FRONTERA_LOCAL identifica los arcos de frontera de $S(q_{act})$ directamente del rango de perfil de lectura [7], y los colecta en la frontera local $\mathcal{F}(q_{act})$.

Acorde con la definición de arriba con la cual un arco de frontera no puede pertenecer a árboles que están siendo construidos por otros robots, implementa un mecanismo de cooperación simple descentralizado teniendo como objetivo optimizar el funcionamiento de la exploración del equipo. Tal mecanismo es inherentemente local y contingente porque confía en la comunicación entre los robots.

4.6.3. Planificador aleatorio

Si la frontera local $\mathcal{F}(q_{act})$ de la RSL actual no esta vacía, el planificador aleatorio de la figura 4.14 genera una configuración aleatoria en la dirección de $\mathcal{F}(q_{act})$. En particular, DIRECCION_ALEATORIA selecciona primero una dirección de exploración θ_{rand} eligiendo uno de los arcos de frontera, usando una probabilidad proporcional a la longitud del arco, y entonces por generación de θ_{rand} sobre la base de una distribución normal con media m (La orientación de la bisectriz del arco) y desviación estándar $\gamma/6$ (γ es la anchura angular del arco). La función DESPLAZAR genera una nueva configuración objetivo q_target moviéndose de q_{act} en la dirección θ_{rand} con un cierto tamaño de paso (un porcentaje fijo de la extencion de la RSL en esa dirección). Si la frontera local esta vacía, el planificador fija la configuración objetivo para ser el nodo padre, es decir, recomienda el robot retroceder.

Note que por simplicidad, hemos referido a un punto del robot para explicar el planificador. El tamaño finito (y posiblemente la forma) del robot es fácilmente tomado en cuenta mapeando los arcos de frontera al espacio de configuración, donde la planificación actual tiene lugar. La posibilidad de que el robot este sujeto a restricciones no-holonómicas es considerada por la función MOVER_A, la cual esta a cargo de generar caminos factibles. La controlabilidad del robot garantiza que cualquier configuración objetivo en la RSL pueda ser alcanzada por caminos factibles que están en la RSL.

Gracias a la sincronización realizada por CONSTRUIR_Y_ESPERAR_GPA, todos los robots en una GEA planifican al mismo tiempo, y por lo tanto el mecanismo de cooperación intrínseco a la definición de la frontera local es reforzada en toda la GEA. Este Íntento de acuerdoés realizado sin ningún modulo de decisión centralizado.

4.6.4. Chequeo de factibilidad del camino GEA

Aunque la frontera local del robot no pueda pertenecer a la RSL del otro robot de la GEA (ver la figura 4.13), los dos caminos anticipados pueden aun intersectarse. La figura 4.15 es un ejemplo de los conflictos que pueden presentarse. Por lo tanto, la función CHECAR_FACTIBILIDAD verifica si el camino anticipado de los robots en la GEA $\mathcal{G}$ son todas simultáneamente factibles[2] o no. Con este fin todos los pares de caminos que se intersectan con otros son puestas a un lado, y los robots correspondientes almacenados en el subconjunto no factible $\mathcal{G}_u$ del GEA. Los robots restantes son el subconjunto factible $\mathcal{G}_f$ del GEA. La complejidad de este chequeo es $O(|\mathcal{G}|^2)$.

4.6.5. Coordinación

Si el subconjunto $\mathcal{G}_u$ de robots con caminos no factibles es no vacía, la función de coordinación es invocada (ver la figura 4.16). El primer paso es elegir un robot maestro dentro de $\mathcal{G}$. Esto puede ser complementado en muchas formas a través de un procedimiento determinístico conocido por todos los robots; por ejemplo, el robot con el mayor numero ID puede ser elegido. Dos casos son posibles entonces:

[2]El chequeo de colisiones es realizado en el espacio de configuraciones, No tomamos en cuenta la posibilidad de escalamiento de velocidad a lo largo del camino para evitar colisiones. Los caminos que se intersectan son clasificados como no factibles, y por lo tanto no se permiten en nuestro enfoque.

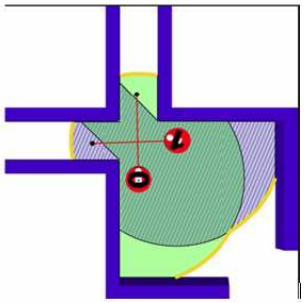


Figura 4.15: Los caminos anticipados de los robots pertenecientes a la GEA pueden intersectarse.

COORDINAR$(\mathcal{G}_f, \mathcal{G}_u)$
1 id_maestro $\leftarrow$ ELECCION_MAESTRO($\mathcal{G}$);
2 Si mi_id = id_maestro
3 $\quad \mathcal{Q}_\mathcal{G} \leftarrow$ ORGANIZAR$(\mathcal{G}_f, \mathcal{G}_u)$;
4 si no
5 $\quad$ ESPERAR
6 Regresar $\mathcal{Q}_\mathcal{G}$(mi_id)

Figura 4.16: (Pseudocodigo de la función coordinación

1. Si el robot es el maestro, este invoca a la función ORGANIZAR, cuya tarea es cambiar el vector $\mathcal{Q}_g$ conteniendo los objetivos de los robots en la GEA así como obtener un movimiento colectivo factible. Aquí, el cambio puede significar lo que sea simplemente aceptando/reajustando el objetivo de un robot para la configuración actual (es decir, autorizando/prohibiendo el movimiento) o agregando una tercera opción, por ejemplo, cambiándolo a un nuevo objetivo. Correspondientemente, hemos ideado dos versiones de la función, ORGANIZAR_1 y ORGANIZAR_2.

2. Si el robot no es el maestro, entra en una fase de espera, la cual termina con la recepción de una señal especifica del maestro.

La operación final es recuperar y regresar el objetivo del robot de $\mathcal{Q}_\mathcal{G}$ (posiblemente modificado).

4.6.6. Organización mediante arbitraje

ORGANIZAR_1 implementa un mecanismo simple de arbitraje en $\mathcal{G}$. En particular, todos los robots contenidos en el subconjunto factible $\mathcal{G}_f$ se les permite moverse (sus configuraciones objetivo se dejan sin cambios). Los robots estan en el subconjunto no factible $\mathcal{G}_u$, no se les permite moverse (su configuración objetivo es inicializada a la actual configuracion) con la excepción de uno solo cuyo movimiento es autorizado (por construcción, esta estrategia esta garantizada para no producir conflictos).

La selección del robot autorizado en $\mathcal{G}_u$ puede realizarse tomando como base varios criterios. El que implementamos selecciona aleatoriamente uno de los robots (cualquiera) cuya frontera local este vacía: esto es cualquiera de los robots cuyo objetivo es su nodo padre (es decir, robots que están ejecutando el procedimiento CONSTRUIR_SRT y al cual el planificador aleatorio ha recomendado el retroceso) o robots que se están moviendo a lo largo de los árboles iniciados por otros robots con el fin de apoyarlos en la expansión (robots que están ejecutando el procedimiento APOYAR_OTROS y aun están en la fase de transferencia). Esta estrategia es motivada por el hecho de que, si su movimiento es no autorizado, tales robots tendrán que esperar para que su camino este limpio, pues no pueden cambiar su objetivo (en comparación a los robots cuya frontera local no esta vacía, a los cuales el planificador aleatorio puede proponer un destino diferente).

Un criterio no ético seria elegir aleatoriamente entre robots en $\mathcal{G}_u$ usando una probabilidad proporcional a la extensión de su frontera local.

4.6.7. Organización mediante replanificación

ORGANIZAR_2 trata de modificar los objetivos de los robots en $\mathcal{G}$ para maximizar el número de movimientos factibles simultáneos. Esto puede realizarse formalizando el problema como sigue.

Considerar el conjunto de configuraciones objetivo $\mathcal{Q}_\mathcal{G}$ asociados a la GEA $\mathcal{G}$. Dos configuraciones objetivo en $\mathcal{Q}_\mathcal{G}$ son llamadas compatibles si pueden ser alcanzadas por los robots correspondientes con caminos que no se intersecten. Sea G el grafo de compatibilidad asociado a $\{\mathcal{G}, \mathcal{Q}_\mathcal{G}\}$ y definido como el grafo indirecto cuyos nodos representan los robots en $\mathbf{G}$ y cuyos arcos unen pares de nodos con objetivos compatibles. Un grupo máximo de G es un subgrafo completo de G con cardinalidad máxima, correspondiente

a un subconjunto máximo de robots con objetivos compatibles. La identificación de grupo máximo es un problema bien conocido NP-completo en el contexto de la teoría de grafos [[14],[15]].

El objetivo ideal de la función ORGANIZAR_2 es modificar el sistema de configuraciones objetivo $\mathcal{Q}_{\mathcal{G}}$ para maximizar la cardinalidad del grupo(s) máximo(s) asociados, con la restricción de que el objetivo de cada robot es cualquiera aceptado, cambiado a otra configuración hacia la frontera local del robot (si esta no es vacía) o a la actual configuración del robot (el movimiento es no autorizado). Este es un problema muy complejo cuya solución requeriría el calculo de grupos máximos como un subproblema. Para encontrar una solución satisfactoria en una cantidad dada de tiempo, hemos adoptado una técnica de búsqueda aleatoria, realizada por el maestro como un juego secuencial con información completa. Una descripción del juego es dada en [11].

4.7. El procedimiento APOYAR_OTROS

El procedimiento APOYAR_OTROS (ver figura 4.17) puede ser dividido en dos fases principales, la cuales son repetidas una y otra vez. En la primera fase, el robot elije otro robot para apoyarlo en su exploración; o, mas precisamente, otro árbol que ayuda a expandir (puede ser que haya mas de un robot actuando en un solo árbol).

En la segunda fase, el árbol seleccionado es alcanzado y el robot trata de expandirlo atando subárboles construidos mediante el procedimiento CONSTRUIR_SRT.

El ciclo principal se repite hasta que el robot ha recibido confirmación de que todos los otros robots han terminado su exploración. En la primera fase, el robot recoge en un conjunto $\mathcal{I}$ los árboles pertenecientes a $\cup\mathcal{T}_i$ que pueden necesitar apoyo para expansión. En particular, definiendo la frontera abierta de un árbol (subárbol) como la suma de las longitudes de las fronteras locales asociadas a sus nodos, un árbol $\mathcal{T}_i$ es puesto en $\mathcal{I}$ si su frontera abierta es al menos igual a una constante $\bar{F}$ multiplicada por el numero de robots que están activos en $\mathcal{T}_i$ según la mas reciente información disponible . Si $\mathcal{I}$ es no vacío, el robot selecciona un árbol particular $\mathcal{T}_s$ de $\mathcal{I}$ de acuerdo a algún criterio (por ejemplo, el árbol con la raíz más cercana, o

```
APOYAR_OTROS(q_act)
1   Hacer
2       para i=1 a n
3         si (FRONTERA_ABIERTA(T_i) ≥ F̄· AGENTE_ACTIVO(T_i)
4           AGREGAR(I, T_i);
5       T_s ← SELECCIONAR(I);
6       Si T_s ≠ NULL
7           q_target ← T_s.raiz;
8           q_act ← MOVER_A(q_target);
9           Hacer
10            CONSTRUIR_Y_ESPERAR_GEA();
11            S(q_act) ← PERCIBIR(q_act);
12            AGREGAR(T_s, (q_act, S(q_act)));
13            G ← CONSTRUIR_GEA();
14            F(q_act) ← FRONTERA_LOCAL(q_act, S(q_act), T, ∪T_i);
15            Si F(q_act) ≠ ∅
16              CONSTRUIR_SRT(q_act, T_s);
17            Si no
18              q_target ← HIJO_ALEATORIO(q_act);
19              Si (q_target = NULL) y (q_act = T_s.raiz)
20                Salir ← 1;
21              Si no
22                Si q_target = NULL
23                  q_target ← q_act.padre;
24                Si |G| > 1
25                  (G_f, G_u) ← CHECA_FACTIBILIDAD(G);
26                  Si G_u ≠ ∅
27                    q_target ← COORDINAR(G_f, G_u);
28                q_act ← MOVER_A(q_target);
29          Mientras (Salir = 0)
30          q_target ← T.raiz;
31          q_act ← MOVER_A(q_target);
32  Mientras EXPLORACION_EJECUTANDOSE()
```

Figura 4.17: Pseudocodigo del procedimiento APOYAR_OTROS

con la actualización más reciente), y moverse a su raíz. Una vez ahí, realiza la construcción de GPA/GEA, percepción y procesamiento de la frontera. Si la frontera local de la raíz no esta vacía, el robot inicia una expansión del subárbol usando el procedimiento CONSTRUIR_SRT. Si no, invoca a la función HIJO_ALEATORIO, la cual realiza una selección aleatoria entre los hijos del nodo, usando una probabilidad proporcional a la frontera abierta del subárbol correspondiente (la función regresa NULL si el subárbol no tiene frontera abierta), y establece el objetivo a los hijos elegidos. El camino factible dentro de la actual GEA es entonces verificada y se realiza una coordinación si se necesita; después de la que, el robot transfiere al objetivo, y el ciclo interno es repetido. Tan pronto como el robot alcance un nodo con una frontera local, iniciara un subárbol de expansión usando el procedimiento CONSTRUIR_SRT. El robot se mantiene tratando de agregar subárboles $\mathcal{T}_s$ hasta que este ha regresado a la raíz de $\mathcal{T}_s$ y su frontera abierta es cero. En este punto, el robot regresa a la raíz de su propio árbol (es decir., su configuración inicial) y se hace disponible para apoyar la expansión de otros árboles.

Esta fase será solo utilizada cuando se maneje un rango de comunicación ilimitado ya que aquí el robot que ayuda dispone en cualquier momento de la información actualizada de los otros robots y por consiguiente de los ultimos estados en los que se encuentran a diferencia del rango de comunicacion limitado en las que solo se cuenta con informacion parcial y probablemente no actualizada.

Capítulo 5

Resultados experimentales

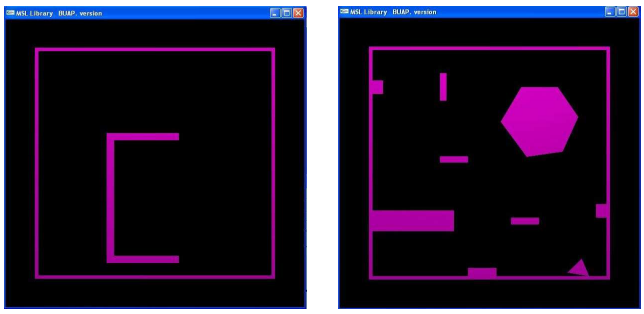
5.1. Explorador SRT multi-robot

En esta sección presentaremos pruebas hechas al sistema desarrollado que lleva por nombre método de exploración cooperativa basada en SRT propuesto. El equipo de exploración esta compuesto por un número variante de robots tomando en cuenta los siguientes valores para cada uno de ellos, $K_{max} = 250$, $I_{max} = 10$, $d_{min} = 3,5$ y $Radio = 11$, donde K_{max} representa el numero de nodos maximos por arbol, I_{max} el número de veces que se intentará encontrar un nuevo nodo candidato en una direccion aleatoria, d_{min} distancia minima entre el nodo candidato y el nodo actual y $Radio$ el radio del sensor.

El funcionamiento del método es evaluado en términos de tiempo de exploración (el tiempo requerido por el ultimo robot del equipo para regresar al inicio). En vista de la naturaleza de nuestro método, los resultados numéricos para cada situación son en promedio sobre diez simulaciones ejecutadas.

Dos diferentes ambientes han sido usados (ver figura 5.1). Ambos ambientes son regiónes cuadradas con una donde cada área puede ser alcanzada de diferentes puntos de acceso.

Consideramos dos posibles despliegues iniciales del equipo. En el primero, los robots son inicialmente dispersados en el ambiente (como si hubieran sido enviados en paracaídas)ver figura 5.3. En la segunda, más realista, la exploración es iniciada con los robots agrupados en un racimo ver figura 5.2.



a)Ambiente 1 b)Ambiente 2

Figura 5.1: Ambientes usados en las pruebas del sistema

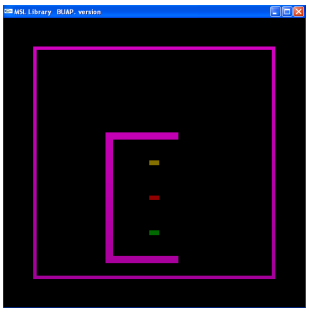


Figura 5.2: Ambiente con robots en inicio no disperso.

La figura 5.6 y la tabla 5.1 se refieren a resultados obtenidos en el ambiente 1 con un inicio disperso. En particular, la figura 5.4 muestra el progreso de la exploración con un equipo de 4 robots en el caso de comunicación ilimitada. Note como cada robot construye su propio SRT y aquel que termina entra en fase de apoyo, el la figura 5.4.b se muestran los arboles de exploracion de los robots unidos, lo cual indica que la fase de apoyo se llevo acabo. la figura 5.5 muestra el progreso de la exploración con un equipo de 4 robots en el caso de comunicación limitada a un rango Rc=2Rp. Note como cada robot construye su propio SRT y nunca entra en la fase de apoyo. Encontramos un comportamiento común cuando los robots son eventualmente distribuidos al inicio.

La figura 5.9 y la tabla 5.2 se refieren a resultados obtenidos en el ambiente 2 con un inicio disperso. En particular, la figura 5.7 muestra el pro-

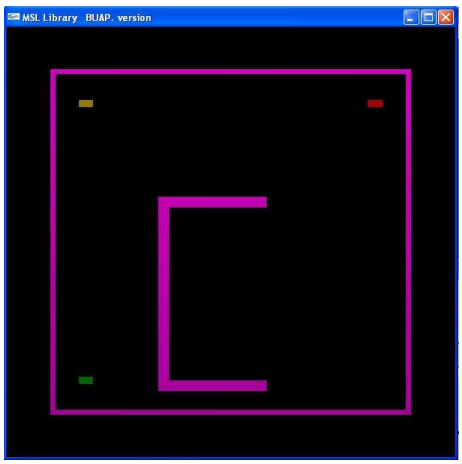
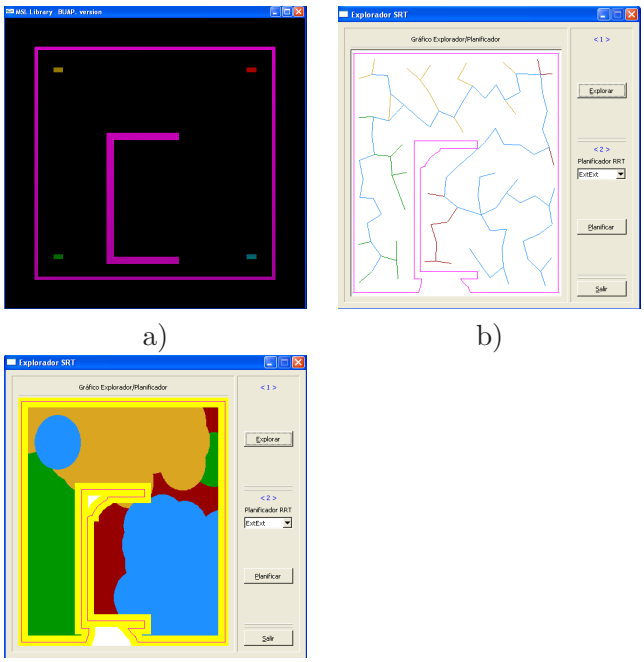


Figura 5.3: Ambiente con robots en inicio disperso.

greso de la exploración con un equipo de 4 robots en el caso de comunicación ilimitada. Note como cada robot construye su propio SRT y aquel que termina entra en fase de apoyo, el la figura 5.7.b se muestran los árboles de exploración de los robots unidos, lo cual indica que la fase de apoyo se llevo acabo. la figura 5.8 muestra el progreso de la exploración con un equipo de 4 robots en el caso de comunicación limitada a un rango Rc=2Rp. Note como cada robot construye su propio SRT y nunca entra en la fase de apoyo. Encontramos un comportamiento común cuando los robots son eventualmente distribuidos al inicio.

Las Figuras 5.10 y 5.12 muestran un típico progreso de exploración de los ambientes 1 y 2 con inicio agrupado, las tablas 5.3 y 5.4 presentan los resultados experimentales del desarrollo de la exploración tanto con la estrategia de rango de comuniación limitado como ilimitado. La figura 5.10.b y figura 5.12.b resume el funcionamiento del equipo. Note que en este caso el tiempo de exploración asintótico tiende a un valor constante diferente de cero, el cual representa aproximadamente el tiempo requerido por un solo robot para realizar un viaje redondo entre el centro del grupo y el punto más lejano del ambiente. En particular, en las figura 5.10 y 5.12 se observa el progreso de la exploración para un equipo de 4 robots con inicio agrupado.



a) b)

c)

Figura 5.4: Ambiente 1 con robots en inicio no disperso y rango de comunicación ilimitado.

5.2. Comparación entre SRT clásico y SRT multi-robot

Habiendo discutido completamente el método SRT multi-robot y realizado pruebas exaustivas a las estrategias de exploración multi-robot desarrolladas, a continuación efectuaremos una serie de pruebas que serviran de comparativo entre el método SRT clásico y el metodo SRT multi-robot.

La figuras 5.15 muestra el ambiente sobre el cual se realizó la exploración usando el metodo SRT clásico, en comparación, la figura 5.16 muestra el ambiente explorado con el método SRT multi-robot.

La tabla 5.5 muestra el tiempo de exploración mono robot en color amarillo y los tiempos de exploración multi-robot en color blanco, la sección multi-robot incluye tanto el tiempo de exploración de la estrategia de rango

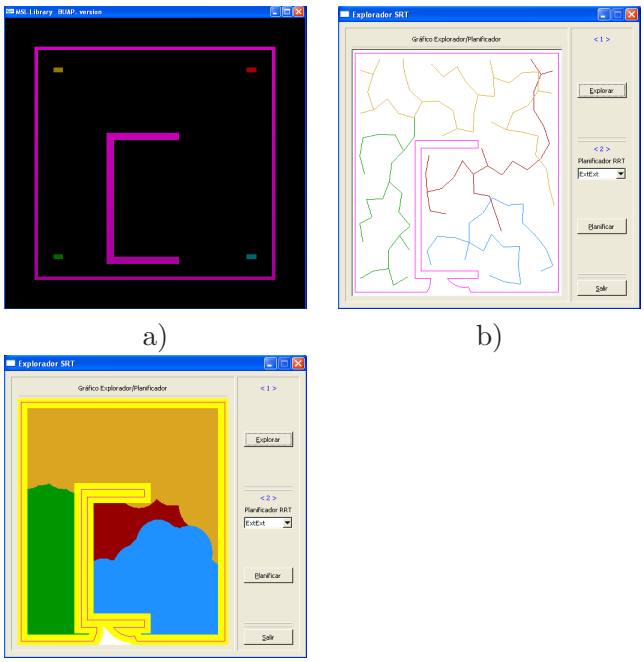


Figura 5.5: Ambiente 1 con 4 robots en inicio disperso y rango de comunicación limitado.

de comunicación ilimitado como de rango de comunicación limitado de 2,3,4 y 5 robots.

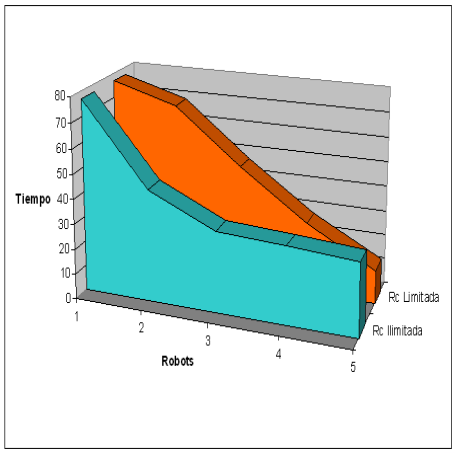


Figura 5.6: Exploración del ambiente 1 en inicio disperso: para rango de comunicación no limitado (color verde) y limitado (color anaranjado).

Explorador SRT multi-robot		
No. de Robots	Tiempo de exploración para la estrategia con rango de comunicación limitado	Tiempo de exploración para la estrategia con rango de comunicación ilimitado
1	78.672 seg.	78.672 seg.
2	43.582 seg.	68.891 seg.
3	35.323 seg.	45.021 seg.
4	33.413 seg.	35.29 seg.
5	31.25 seg.	9.980 seg.

Cuadro 5.1: Exploración del ambiente 1 en inicio disperso

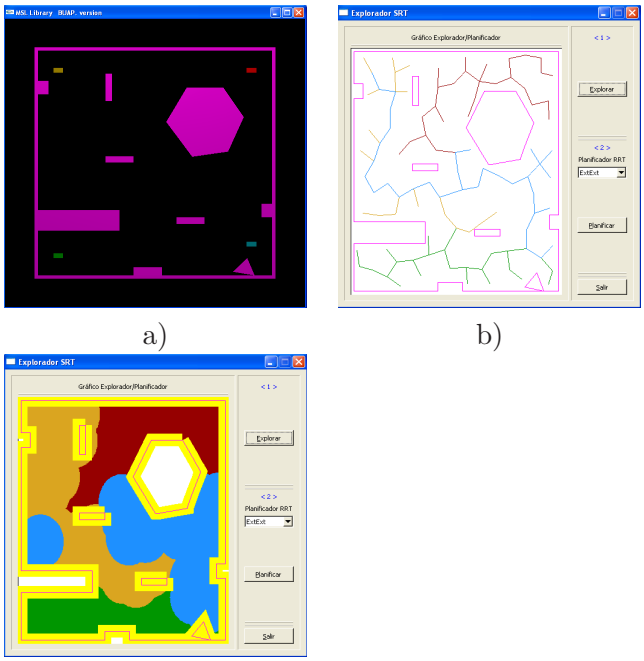


Figura 5.7: Ambiente 2 con 4 robots en inicio disperso y rango de comunicación ilimitado.

Explorador SRT multi-robot		
No. de Robots	Tiempo de exploración para la estrategia con rango de comunicación limitado	Tiempo de exploración para la estrategia con rango de comunicación ilimitado
1	70.195 seg.	70.195 seg.
2	40.586 seg.	39.622 seg.
3	39.265 seg.	37.476 seg.
4	35.624 seg.	28.941 seg.
5	30.201 seg.	32.331 seg.

Cuadro 5.2: Exploración del ambiente 2 en inicio disperso

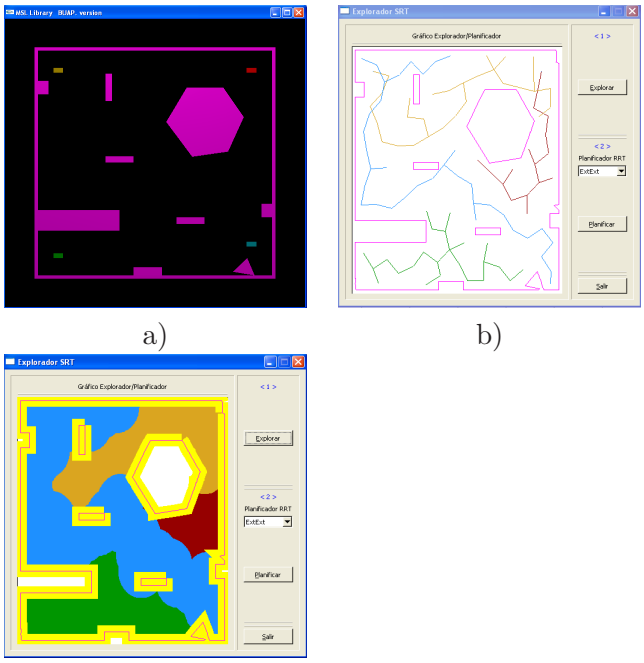


Figura 5.8: Ambiente 2 con 4 robots en inicio disperso y rango de comunicación limitado.

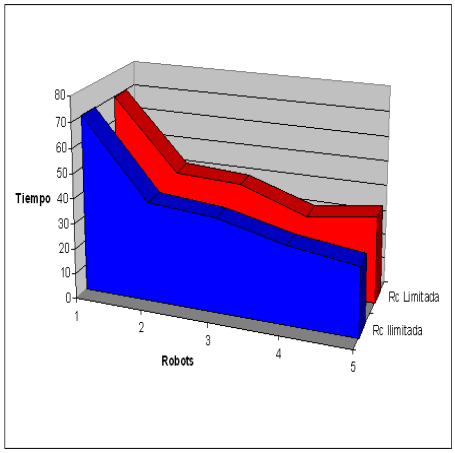


Figura 5.9: Exploración del ambiente 2 con inicio disperso: para rango de comunicación no limitado (color azul) y limitado (color rojo).

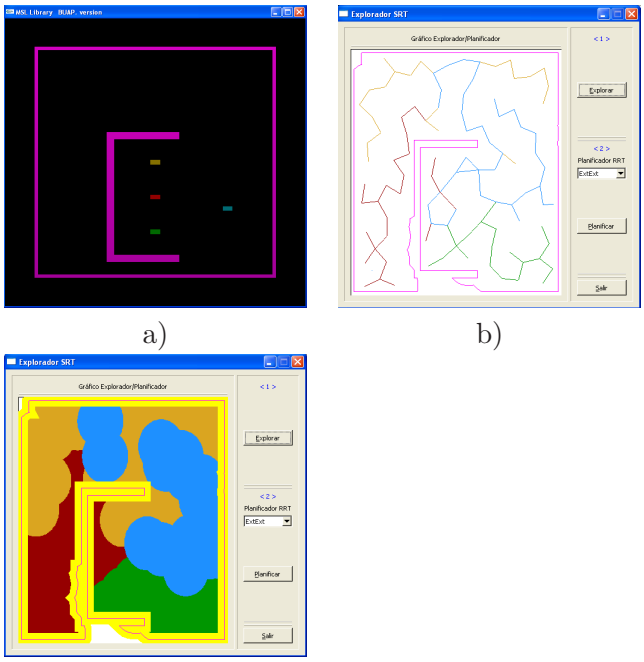


Figura 5.10: Ambiente 1 con 4 robots en inicio agrupado.

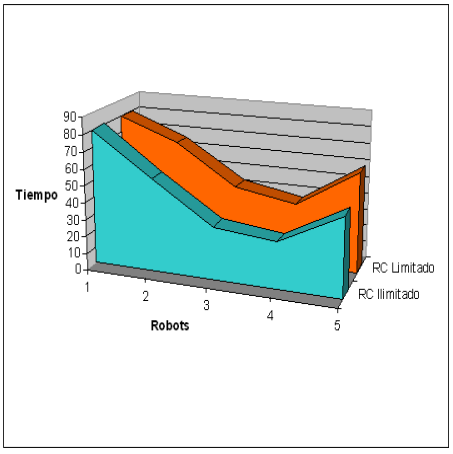
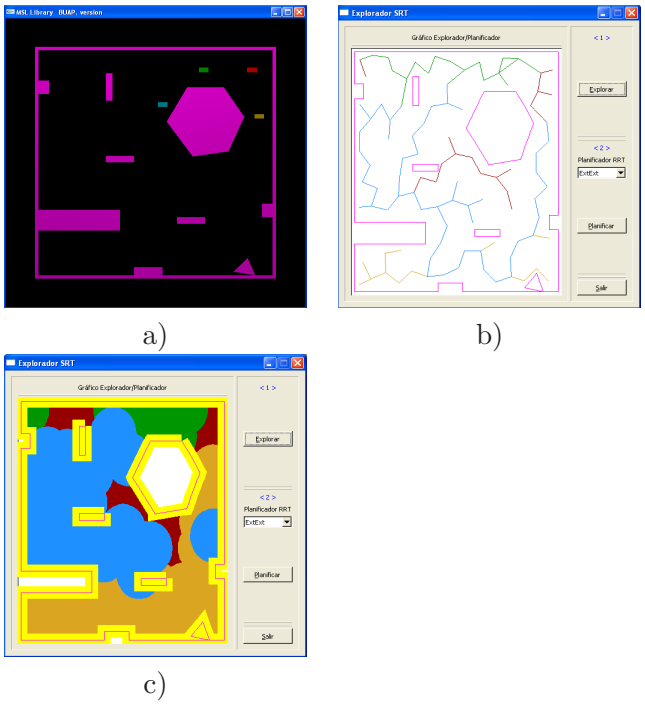


Figura 5.11: Exploración del ambiente 1 en inicio agrupado: para rango de comunicación no limitado (color verde) y limitado (color anaranjado).

Explorador SRT multi-robot		
No. de Robots	Tiempo de exploración para la estrategia con rango de comunicación limitado	Tiempo de exploración para la estrategia con rango de comunicación ilimitado
1	80.562 seg.	80.562 seg.
2	57.214 seg.	64.492 seg.
3	34.928 seg.	40.424 seg.
4	30.720 seg.	32.935 seg.
5	51.079 seg.	56.836 seg.

Cuadro 5.3: Exploración del ambiente 1 en inicio agrupado



a) b)

c)

Figura 5.12: Ambiente 1 con 4 robots en inicio agrupado.

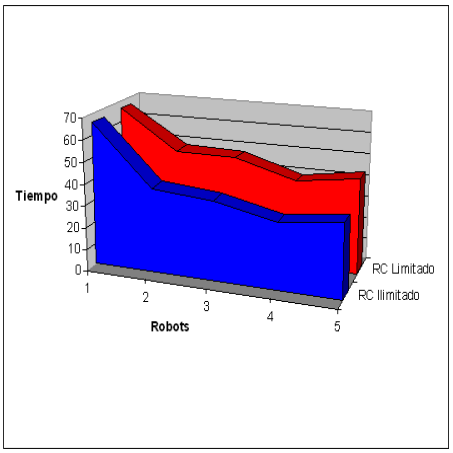


Figura 5.13: Exploración del ambiente 2 en inicio agrupado: para rango de comunicación no limitado (color azul) y limitado (color rojo).

Explorador SRT multi-robot		
No. de Robots	Tiempo de exploración para la estrategia con rango de comunicación limitado	Tiempo de exploración para la estrategia con rango de comunicación ilimitado
1	67.872 seg.	67.872 seg.
2	40.023 seg.	45.928 seg.
3	38.725 seg.	43.954 seg.
4	34.378 seg.	37.573 seg.
5	37.281 seg.	41.134 seg.

Cuadro 5.4: Exploración del ambiente 2 en inicio agrupado

Figura 5.14: Ambiente utilizado para la comparación entre SRT clásico y SRT multi-robot

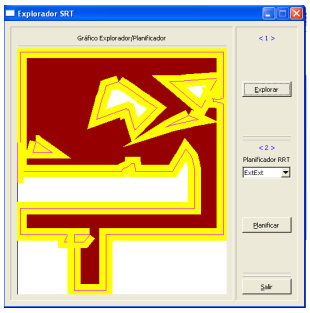


Figura 5.15: Región explorada mediante SRT clásico

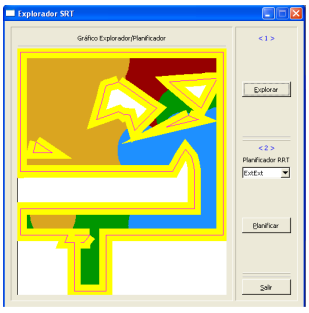


Figura 5.16: Región explorada mediante SRT multi-robot

Comparativo entre SRT clásico y SRT multi-robot		
No. de Robots	Tiempo de exploración para la estrategia con rango de comunicación limitado	Tiempo de exploración para la estrategia con rango de comunicación ilimitado
SRT clásico		
30.438 seg.		
SRT multi-robot		
2	28.014 seg.	30.438 seg.
3	20.313 seg.	28.704 seg.
4	17.997 seg.	16.035 seg.
5	8.625 seg.	12.513 seg.

Cuadro 5.5: Comparativo de tiempos entre SRT clásico y SRT multi-robot

Capítulo 6

Conclusiones y trabajos futuros

Hemos presentado, la implementación de un método de exploración para ambientes desconocidos usando información sensorial provista por un robot móvil omnidireccional. También adaptamos el método SRT con la estrategia SRT-radial para tomar ventaja de la información proporcionada por los sensores en una forma diferente, del original SRT propuesto (de hecho se presentó en el capítulo 4 un comparativo entre las estrategias SRT-Star y SRT-Radial). La robustez de la aproximación propuesta fue probada con muchos problemas simulados.

Después de muchos experimentos realizados con ambas estrategias, notamos que el método SRT no hace una distinción entre obstáculos y áreas no exploradas. De hecho el límite de la región segura local (RSL) puede indiferentemente describir el límite del rango del sensor o el perfil de los objetos. Esto significa que durante la fase de exploración, el robot puede aproximarse a áreas que parecen estar ocupadas. Además, una diferencia importante de SRT con otros métodos, es la forma en la cual se representa el ambiente. El espacio libre estimado durante la exploración es simplemente la unión de las RSLs asociadas con los nodos del árbol. Sin embargo, con pocas operaciones (relativamente simples) de post proceso, permitirían al método procesar una descripción global de la RSL, la cual sería de mucha utilidad en tareas de navegación.

Yamauchi ha demostrado que la exploración basada en frontera puede ser usada en ambientes de interior donde las paredes y obstáculos pueden

estar en orientaciones arbitrarias por ejemplo. Podemos decir que nuestra aproximación es una modificación del método SRT basado en frontera. La diferencia con el método basado en frontera original radica en el hecho que nuestra aproximación no usa un mapa global para identificar la frontera de la región explorada, y aún es de carácter probabilista [46].

Las conclusiones anteriores son para el caso de un solo robot móvil, se implementó como parte de este trabajo de tesis, la estrategia SRT-Star para verificar que efectivamente la propuesta realizada en la tesis de Judith Espinoza de SRT-Radial es más robusta [10]. A continuación se presentan las conclusiones para el trabajo principal de esta tesis, la exploración cooperativa.

Presentamos estrategias probabilistas para la exploración cooperativa basadas en el método SRT (árbol aleatorio de exploración basado en sensores). La coordinación local en el caso del rango de comunicacion limitado y/o la comunicacon global en el caso del rango de comunicacion ilimitado garantiza que el movimiento colectivo del equipo sea factible desde el punto de vista de las colisiones.

Despues de las pruebas exhaustivas realizadas, se encontraron diferencias importantes en las estrategias implementadas:

- Para el caso de la estrategia con rango de comunicación limitado, se definió un rango de comunicación de dos veces el rango de percepcion. Se pudo notar que en algunas ocasiones, los robots exploran en más de una ocasión una área determinada del mapa (esto es debido a que solo se cuenta con infomación parcial de los árboles obtenida cuando los robots exploradores se encuentran dentro del rango de comunicación). En lo que respecta al tiempo de ejecución, esta particularidad, provoca que el tiempo se incremente considerablemente. Obviamente, la calidad del mapa es superior respecto de la otra estrategia implementada (esto se debe a que hay redundancia de las áreas exploradas).
- En la estrategia con rango de comunicación ilimitado, el tiempo de ejecución se reduce considerablemente debido que ningún robot queda inactivo después de terminar su propia exploración, esto se consigue gracias a la fase de apoyo.

Cuando se tiene un inicio agrupado con rango de comunicación limita-

do, no es necesaria la fase de apoyo, se observa que el método lleva acabo la exploración en menor tiempo que la exploración de un solo robot, pero en comparacion con la estrategia con rango de comunicación ilimitado, el tiempo de exploracion es mayor, esto debido a que algunos robots quedan encerrados y por lo tanto inabilitados para ayudar en la exploración, estos robots quedan estacionados mientras los que quedan libres realizan la exploración.

Con una apropiada distribucion de los robots en el ambiente a explorar, es decir con un inicio disperso, la exploracion se lleva acabo de forma más rapida y eficiente.

Como trabajos futuros se plantean varias metas:

- Dado que asumimos que se cuenta con un modulo de localización, sería adecuado (esto redundaría en mejores resultados al momento de llevar las estrategias al caso de robots reales) contar con un modulo de localización. Por ejemplo, sería interesante investigar sobre los métodos basados en características (o rasgos) naturales para localización.
- Una extensión natural sería el considerar robots heterogéneos. Esto obviamente sería más adecuado, debido a que en los problemas reales de exploración, los robots son diferentes (cinematicamente hablando, en sensores, etc).
- Otro aspecto importante en este tipo de exploración, es el considerar la incertidumbre que produce el sistema de comunicación entre los robots.
- Es claro, que una experimentación con robots reales, sería factible de realizarse.
- Debido a que las estrategias adaptan los planificadores RRT (los cuales son adecuados para manejar muchos grados de libertad y espacios de altas dimensiones), esta característica nos permitiría trabajar en un proyecto inherentemente más complicado, la exploración cooperativa de robots mamipuladores móviles.

Apéndice A

Librería MSL

La librería Motion Strategy Library, MSL[1], creada por Steve Lavalle y su grupo de colaboradores, permite un fácil desarrollo y prueba de algoritmos de planificación de movimientos para una amplia variedad de aplicaciones. La arquitectura del software es orientada a objetos y el diseño general es altamente modular.

Actualmente MSL incluye planificadores que usan árboles de exploración rápida (RRTs), roadmaps probabilísticos (PRMs), y programación dinámica directa (FDP).

A.1. Descripción general

MSL consiste de siete clases jerárquicas en C++, cada una de ellas sirve para un proposito independiente. La relación entre ellas se muestra en la figura A.1. Este no es un diagrama jerárquico; solo muestra el flujo de la información a través de la jerarquía.

Cada una de las siete clases jerárquicas se explican brevemente a continuación:

- **Model:** Las clases Model contienen simuladores incrementales que modelan la cinemática y dinámica de una variedad de sistemas mecánicos. Los métodos permiten que los algoritmos de planificación calculen el

[1] http://msl.cs.uiuc.edu/msl/

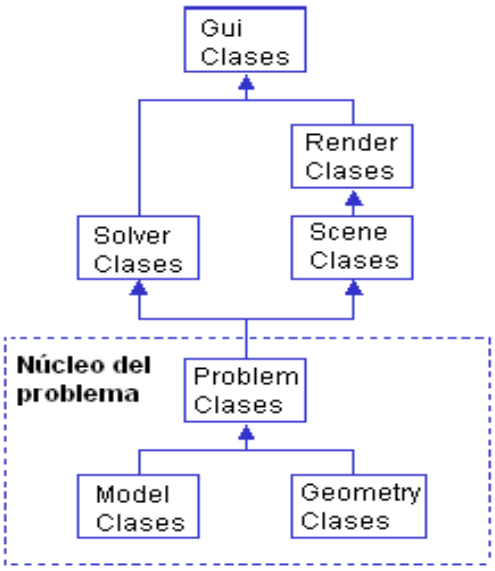


Figura A.1: Clases que constituyen la librería MSL.

siguiente estado del sistema, dado el estado actual, un intervalo de tiempo, y una entrada de control aplicada sobre ese intervalo.

- **Geom:** Estas clases definen las representaciones geométricas de todos los obstáculos del ambiente, y cada parte del robot. Los métodos permiten a los algoritmos de planificación determinar si cualquier parte del robot esta en colisión con otra parte o con los obstáculos del ambiente.

- **Problem:** Esta es una clase interface para un planificador, la cual abstrae al diseñador del algoritmo de planificación lejos de detalles específicos tales como la detección de colisión, y la simulación dinámica. Cada instancia del problema incluye tanto una instancia de la clase Model como una de Geom. También se incluye un estado inical y uno final, lo cual permite que un problema sea resuelto, típicamente por un algoritmo de planificación.

- **Solver:** Hay actualmente un tipo de solucionador, el cual es una jerarquía de planificadores. Un objeto Solver es inicializado con una instancia de la clase Problem, y un método busca una estrategía de movimiento que resuelva el problema.

- **Scene:** Esta es una clase interface que calcula las configuraciones de todos los cuerpos que serán desplegados por el método de renderizado. Esta clase recibe la mayoría de la información directamente de la clase problem, pero incluye información adicional relevante para renderizar las imagenes, como el punto de vista de la cámara.

- **Render:** Esta jerarquía de clases contiene diferentes implementaciones de soluciones de renderizado gráfico. Por ejemplo, cuando una interface de usuario gráfica (GUI) solicita que el camino solución sea animado, un método en una clase Render despliega los cuerpos en movimiento usando configuraciones obtenidas de la clase Scene. Cada clase derivada en Render corresponde a un diferente sistema gráfico. Actualmente, hay renderizadores para desarrolladores SGI IRIS, Open GL, Open Inventor. La flexibilidad proporcionada por estas clases permite fácilmente extensiones creadas por otras librerías gráficas y plataformas.

- **Gui:** La interface gráfica de usuario (GUI) esta diseñada como una jerarquía de clases para generar interfaces de usuario especificas, diseñadas para una variedad de problemas de estrategía de movimientos y algoritmos de planificación. Actualmente, hay una clase que sirve como la GUI para todos los planificadores que utilizan RRT. Cada instancia de la GUI incluye una instancia de una clase planificador-RRT y una instancia de una clase Render. Usando este esquema, puede usarse el mismo diseño básico de la GUI, sin tener en cuenta métodos de renderizado en particular.

A.2. Librerías utilizadas por MSL

- **FOX C++ GUI Toolkit.** Es una librería de clases en C++ para construir Interfaces Gráficas de Usuario.

- **Proximity Query Package (PQP).** Este paquete fue desarrollado por la universidad de Carolina del Norte, y es libre para uso no comercial. Ejecuta una eficiente detección de colisión y cálculos de distancia para una colección de triángulos en un mundo 3D. PQP es generalmente fácil de instalar.

- **Open GL, GLUT OpenGL, o una API equivalente, como MesaGL**. Es una API para desarrollar aplicaciones gráficas 2D y 3D. En MSL se requieren para efectuar el renderizado en 3D usando el render GL. Las librerías necesarias pueden obtenerse de forma libre para

la mayoría de las plataformas (por ejemplo, son incluidas en las distribuciones de Linux RedHat). Los archivos de la librería son libglut, libGLU y libGL. El archivo libglut proviene del paquete glut, el cual provee algunas utilerías GL. También existen en su versión windows y son fáciles de instalar.

- **Open Inventor.** Esta librería únicamente es requeridas si se quiere utilizar el render que lo implementa. Actualmente tiene todas las caracteristicas del render que usa GL, más algunos otros. Se recomienda ya que el sombreado es más exacto.

- **OpenGL Performer.** Esta librería es requerida sólo si se pretende utilizar el render desarrollado para la misma. Es deseable sólo si se quiere desarrollar gráficos de alto rendimiento con MSL. También es necesario, si el objetivo son modelos artísticos.

A.3. Crear un problema nuevo

Cada problema se describe por un directorio completo de archivos. La mayoria de ellos son archivos ASCII que son fáciles de leer y escribir. Para hacer un nuevo ejemplo, algunos archivos son necesarios y otros opcionales. Se asumen valores por default para los parámetros cuando los archivos no estan presentes. Algunos modelos en particular podrían requerir archivos que no son usados por otros modelos. A excepción de leer el código, una forma fácil de encontrar que archivos podrían ser necesarios para un modelo en particular es modificar un ejemplo incluido en la distribución de la librería y usar el mismo modelo.

Los archivos siguientes son requeridos para todos los ejemplos:

- *GeomDim:* La dimensión del ambiente (2 o 3).

- *ModelXXX:* Un archivo nombrado exactamente después que el modelo de movimiento es usado. Por ejemplo, Model2DRigid y Model3DRigidMulti.

- *GeomXXX:* Un archivo nombrado exactamente después del modelo geométrico, tal como GeomPQP3DRigid para un cuerpo rígido hecho de triángulos en un mundo 3D (con el uso del paquete PQP para la deteccion de colisiones).

- *InicialState:* El estado inicial del problema.

- *GoalState:* El estado deseable meta o final.
- *Robot:* Un modelo del robot, especificado como una lista de polígonos si $GeomDim = 2$ o una lista de triángulos 3D si $Geomdim = 3$. Para problemas que involucran múltiples cuerpos los robots son nombrados Robot0, Robot1, ... , Robotk, para k robots.

Los siguientes archivos son opcionales:

- *ModelDeltaT:* El incremento de tiempo a ser usado para la integración númerica de la ecuación de transición de estado (ecuaciones de movimiento).
- *PlannerDeltaT:* El incremento de tiempo usado por el paso de planificación incremental.
- *Obst:* Una lista de obstáculos estacionarios, especificados como una lista de polígonos si $Geomdim = 2$ o una lista de triángulos 3D si $Geomdim = 3$.
- *EnvList:* Una lista de nombres de archivos que corresponden a los modelos estacionarios que pueden cargarse o renderizarse. Si EnvList no existe, entonces Obst es cargado y renderizado.
- *BodyList:* Una lista de nombres de archivos que corresponden a cuerpos moviles. Si BodyList no existe, entonces Robot o Robot0, Robot1, ..., Robotk son cargados.
- *LowerState:* El vector estado con los valores más bajos posibles. Cada elemento es el valor más pequeño para esa variable de estado.
- *UpperState:* El vector estado con los valores más altos posibles. Cada elemento es el valor más grande para esa variable estado.
- *RRTXXX:* Una selección del planificador por default, tal como RRTExtExt o RRTConCon.
- *Inputs:* Una lista de vectores de entrada para ser aplicados a la ecuación de transición de estado. Creando este archivo, podemos anular los valores por defecto de la clase constructor. Por ejemplo, un carro que puede ir hacia delante o hacia átras puede convertirse en un carro de avace solo delantero cambiando este archivo.

- *ViewingPosition:* La posición (x, y, z) de la cámara que se usará en el renderizado.
- *ViewingDirection:* Dirección a la que la cámara debería apuntar (un vector 3D).
- *ViewingOrientation:* Una posible rotación en la dirección del punto de vista de la cámara.
- *LowerWorld:* El valor más pequeño en (x, y, z) del ambiente.
- *UpperWorld:* El valor más grande en (x, y, z) del ambiente.
- *Holonomic:* Permite a un planificador conocer que el problema es holonómico con una ecuación de transición de estado de la forma $\dot{x} = u$. En este caso, el desempeño puede ser mejorado considerablemente ignorando las entradas y realizando interpolación líneal para generar movimientos locales.

A.4. Jerarquía de clases

A continuación presentamos los diagramas de las clases principales que constituyen a MSL.

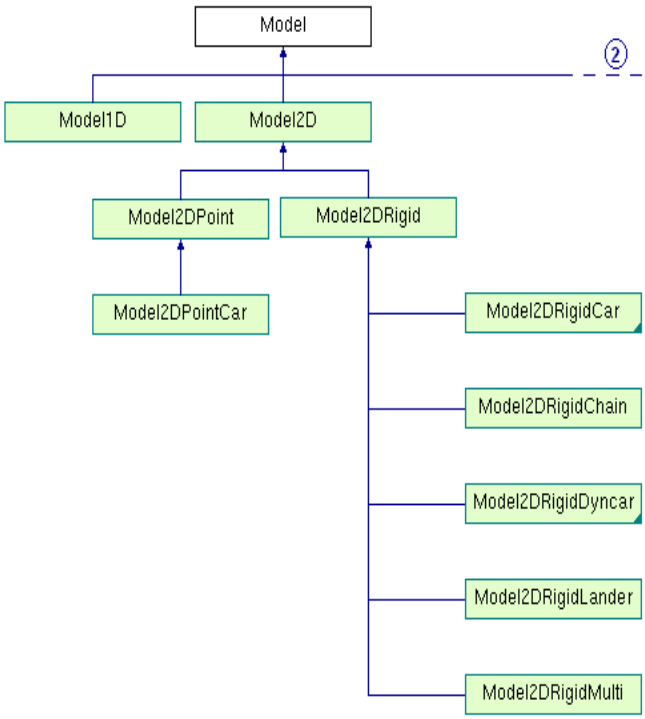


Figura A.2: Jerarquía clase Model

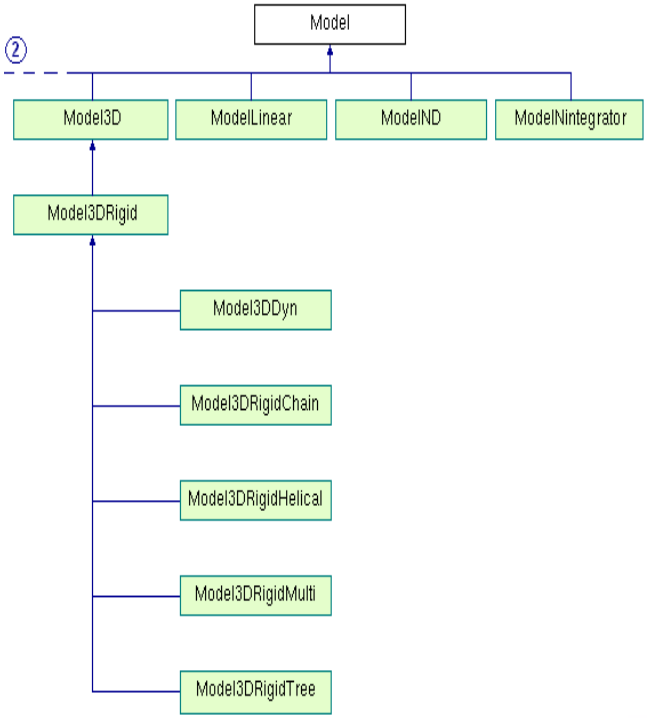


Figura A.3: Jerarquía clase Model

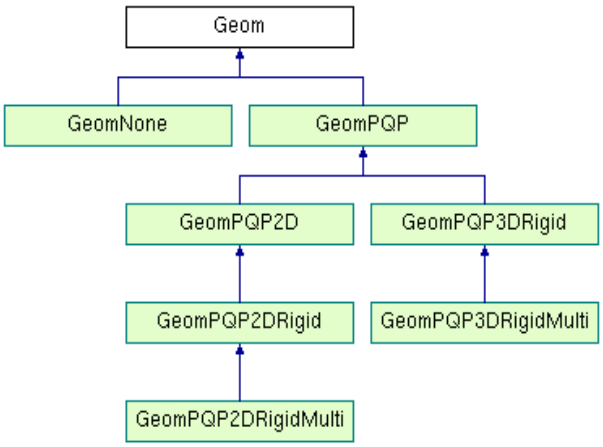


Figura A.4: Jerarquía clase Geom

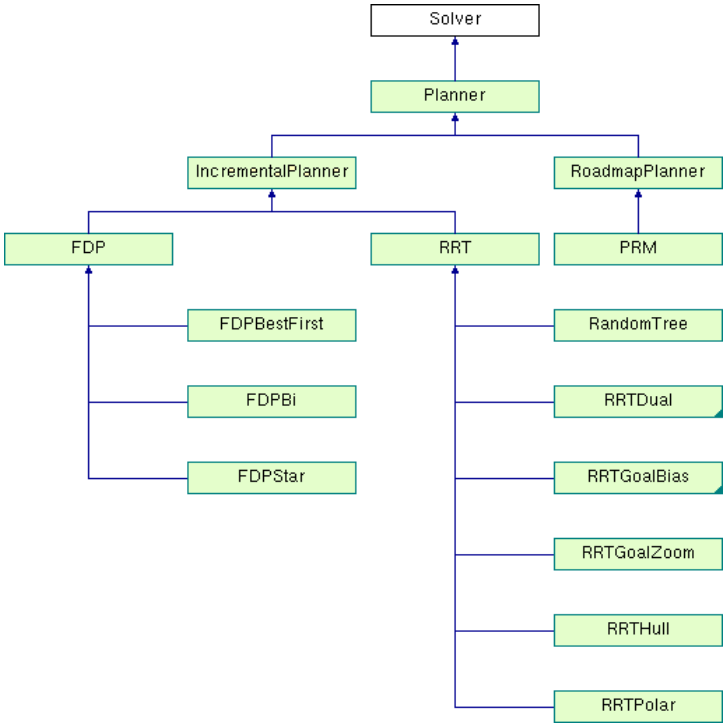


Figura A.5: Jerarquía clase Solver

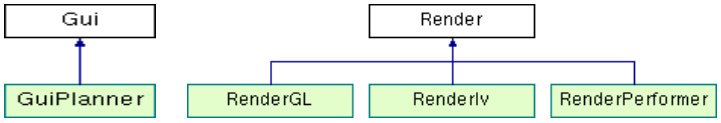


Figura A.6: Jerarquía clase Gui y Render.

Apéndice B

Libreria GPC

Tradicionalmente el recorte de polígonos se ha usado para recortar las porciones de un polígono que se encuentran fuera de la ventana del dispositivo de salida (por ejemplo la pantalla) para prevenir efectos indeseables. El recorte de un polígono arbitrario contra otro polígono arbitrario ha sido una tarea compleja. Las soluciones existentes son limitadas a cierto tipo de polígonos o tienden a ser muy complejas y consumir demasiado tiempo de ejecución.

En la librería GPC (General Polygon Clipping) se implementa un nuevo algoritmo de recorte de polígonos. Las técnicas usadas se basan en el método de recorte de polígonos de Bala R. Vatti [45]. Tanto el polígono o conjunto de polígonos a recortar (polígono sujeto) como el polígono utilizado para el corte, (polígono de corte), pueden ser convexos o concavos, interceptarse a sí mismos, contener huecos, o estar comprendidos en varios contornos disjuntos. La librería extiende el algoritmo de Vatti para permitir lados horizontales y manejar de manera robusta la coincidencia de lados en los polígonos. Las operaciones que se pueden realizar con dicha librería son: *intersección, or-exclusivo, unión y diferencia* del polígono sujeto con el polígono de recorte. La salida puede ser el contorno de otro polígono o una lista de triángulos (tristrip).

B.1. Descripción

Un polígono genérico (o un 'conjunto poligono') consiste de cero o más limites disjuntos de una composición arbitraria. Cada limite se le denomina

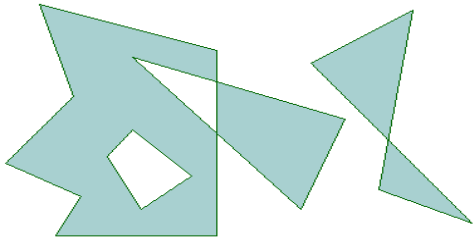


Figura B.1: Polígono genérico con cuatro contornos.

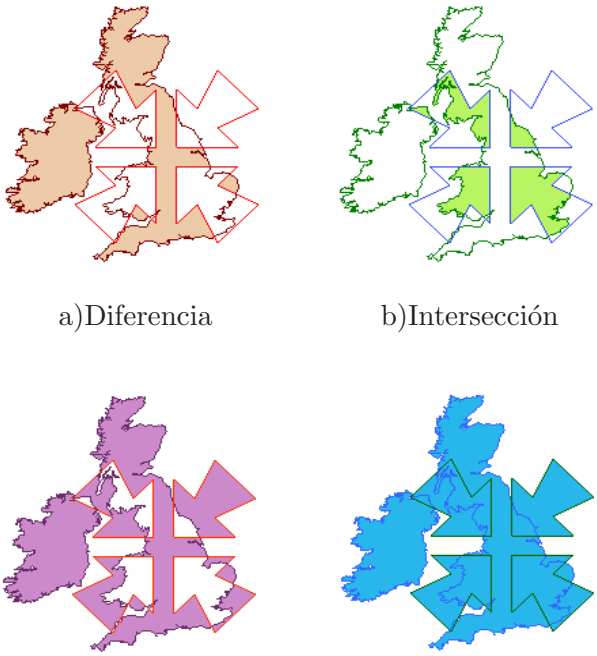
"contorno", y puede ser como ya se menciono, convexo, concavo o intersectarse a sí mismo. Los huecos internos pueden formarse debido a los contornos. Ver la figura B.1, donde se muestra un conjunto polígono constituido por cuatro contornos. A la izquierda tenemos un contorno concavo de siete lados que contiene otro contorno de cuatro lados, el cual forma un hueco en la figura envolvente. Un tercer contorno triangular, intercepta el límite del primero. Finalmente a la derecha hay un contorno disjunto que se intercepta a sí mismo de cuatro lados.

La librería soporta cuatro tipos de operaciones de recorte: la diferencia, intercepción, or-exclusivo o unión de dos polígonos genéricos. La figura B.2 muestra los tipos de operación, en cada caso el polígono resultante es resaltado con un color de relleno.

B.2. Funciones

La librería proporciona ocho funciones. Dos de ellas estan dedicadas a la lectura y escritura de datos entre los archivos de los polígonos y las estructuras propias de la librería.

```
void gpc_read_polygon(FILE        *fp,
                      int          read_hole_flags,
                      gpc_polygon *polygon);
void gpc_write_polygon(FILE        *fp,
                       int          write_hole_flags,
                       gpc_polygon *polygon);
```



a)Diferencia b)Intersección

c)Or-exclusivo d)Unión

Figura B.2: Operaciones de recorte que maneja la librería GPC.

```
2
3
1
  4.0   4.0
  6.0   5.0
4
0
  2.0   1.0
  8.0   2.0
  7.0   9.0
  1.0   7.0
```

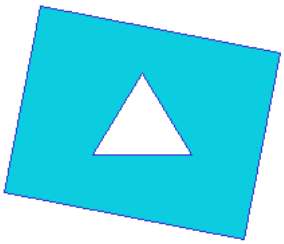


Figura B.3: Polígono y su correspondiente archivo ascii

El manejo de polígonos es por medio de un archivo ASCII con el siguiente formato:

$\langle numero - contornos \rangle$
$\langle numero - vertices - primer - contorno \rangle$
$[\langle bandera - huecos - primer - contorno \rangle]$
$\langle lista - vertices - primer - contorno \rangle$
$\langle numero - vertices - segundo - contorno \rangle$
$[\langle bandera - huecos - segundo - contorno \rangle]$
$\langle lista - vertices - segundo - contorno \rangle$
$\vdots$

La bandera de huecos es opcional, las operaciones de recorte establecerán correctamente las banderas de huecos en el polígono resultante. La figura B.3 muestra un ejemplo de un polígono simple con un hueco triangular en un cuadrilatero y su correpondiente archivo ASCII con las banderas de huecos incluidas.

La función `gpc_add_contour()` facilita la fusión de un nuevo contorno con un poligono existente. El parámetro final indica si el contorno sería considerado como hueco o no, los valores que utiliza son 1 (true) o 0 (falso) según corresponda.

```
void gpc_add_contour(gpc_polygon     *p,
                     gpc_vertex_list *c,
                     int              hole);
```

La siguiente función desarrolla las operaciones de recorte,

```
void gpc_polygon_clip(gpc_op       operation,
                      gpc_polygon *subject_p,
                      gpc_polygon *clip_p,
                      gpc_polygon *result_p);
```

si el resultado se requiere en un tristrip se utiliza la siguiente función,

```
void gpc_tristrip_clip(gpc_op        operation,
                       gpc_polygon  *subject_p,
                       gpc_polygon  *clip_p,
                       gpc_tristrip *result_t);
```

En ambos casos el primer parámetro especifica la operación de recorte (GPC_DIFF, GPC_INT, GPC_XOR, GPC_UNION). Los siguientes parámetros son el polígono sujeto y el de corte. En el parámetro final se almacenará el resultado, ya sea como polígono o una colección de tristrip.

Un polígono puede convertirse en su equivalente tristrip con la función,

```
void gpc_polygon_to_tristrip(gpc_polygon   *source_p,
                             gpc_tristrip *result_t);
```

Las últimas dos funciones son para liberar la memoria utilizada por las estructuras de datos para los polígonos y para los tristrips,

```
void gpc_free_polygon(gpc_polygon *p);
void gpc_free_tristrip(gpc_tristrip *t);
```

B.3. Huecos y contornos externos

Un contorno se clasifica como un *contorno externo* si su vértice más alto (o más a la derecha, cuando el contorno es horizontal) forma un *máximo local externo (MLE).* Si este vértice forma un *máximo local interno (MLI)* el contorno se clasifica como un *contorno de un hueco.*

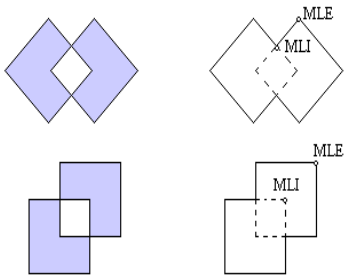
Cuando los lados del contorno se cruzan (por ejemplo en figuras que se interceptan a sí mismas) siempre generan un vértice máximo local debajo de la intersección (o cuando uno de los lados es horizontal, a la izquierda de la intersección) lo cual conecta las partes de los dos contornos que se encuentran por debajo o a la izquierda del punto de intersección. Las partes de los lados que surgen en el costado opuesto al punto de intersección originan un nuevo vértice mínimo local. El contorno cerrado generado nunca se interceptará a sí mismo.

Estas reglas tienen implicación con respecto a como descomponer las figuras que se interceptan a sí mismas en un conjunto de contornos cerrados que no se intercepten. La figura B.4.a) muestra ejemplos de cuadrados interceptandose, cada uno creará un contorno en su interior. En cada caso, el vértice máximo arriba o a la derecha del contorno interno forma un máximo interno, por tanto, el contorno es clasificado como un hueco (línea punteada). El correspondiente contorno exterior es considerado externo ya que termina con un vértice máximo externo.

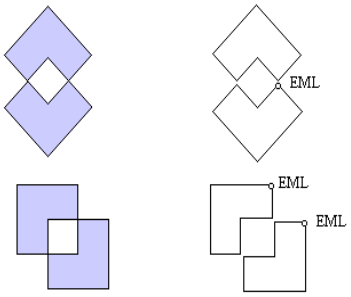
Los ejemplos en la figura B.4.b), muestran figuras similares que se intersectan a sí mismas, sin embargo, cada una crea dos contornos externos tocandose en el punto de intersección. En resumen, los contornos generados por el recortador no sólo son afectados por la forma de los polígonos entrantes sino también por su orientación.

B.4. Asociación de los huecos con el contorno externo

La actual versión simplemente utiliza las banderas para identificar cuales contornos pueden ser considerados huecos y cuales externos. Descubriendo cuales huecos se sitúan en que contornos externos toma un poco más de trabajo por parte del usuario, Una forma de asociar los huecos $H_1, H_2, \ldots, H_n$ con los contornos externos $E_1, E_2, \ldots, E_n$ es usar la librería para calcular la diferencia del i-ésimo hueco con cada contorno externo E_j, para todo j de 1 a m. Cualquier diferencia que obtenga un resultado vacio indica que el i-ésimo hueco se encuentra en el contorno externo j.



a)



b)

Figura B.4: Contornos que se interceptan a sí mismos. a) Los contornos se descomponen en un contorno externo que contiene un hueco en su interior. b) Los contornos, sin embargo, se descomponen en dos contornos externos tocandose en el punto de intersección.

B.5. Lados coincidentes y casi-coincidentes

El cortador fusiona los lados que son coincidentes. Los contornos adyacentes del polígono sujeto y del cortador que comparten un lado en común los fusionará para formar un simple contorno bajo la operación de *unión*, y producira un resultado nulo a lo largo del límite compartido con la operación de *intersección*. La presición de los límites numéricos es probablemente la causa de la leve perdida de registro de los lados coincidentes, resultando en un error en la fusión. Incrementando el valor de la constante `GPC_EPSILON` en `gpc.h` influirá en la fusión de lados casi-coincidentes. Sin embargo, pueden resultar figuras poligonales incorrectas si se da a `GPC_EPSILON` un valor bastante grande.

Bibliografía

[1] J. M Ahuactzin and A. Portilla. A basic algorithm and data structures for sensor-based path planning in unknown environments. *Proceedings of the IEEE, International Conference on Intelligent Robots and Systems*, 2000.

[2] N. M. Amato, O. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based prm for 3d workspaces. *Workshop on the Algorithmic Foundations of robotics*, pp. 155-168, 1998.

[3] Antonio Franchi, Luigi Freda, Giuseppe Oriolo, and Marilena Vendittelli. . A Randomized Strategy for Cooperative Robot Exploration. *IEEE International Conference on Robotics and Automation*, 2007.

[4] P. Bessiére, E. Mazer, and J. M. Ahuactzin. The ariadne's clew algorithm: Global planning with local methods. *Workshop on the Algorithmic Foundations of robotics*, 1994.

[5] W. Burgard, M. Moors, C. Stachniss, and F. Schneider. . Coordinated multi-robot exploration. *IEEE Trans. on Robotics and Automation*, 1(3):376-386, 2005.

[6] J. F. Canny. The complexity of robot motion planning. *MIT Press, Cambridge, MA*, 1988.

[7] H. Choset and J. Burdick. Sensor based motion planning: The hierarchical generalized voronoï graph. *Workshop on Algorithmic Foundations of Robotics*, pp. 1643-1648, 1996.

[8] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497–516, 1957.

[9] M. Erdmann and T. Lozano-Pérez. On Multiple Moving Objects. *In IEE Internacional Conference on Robotics and Automation, San Francisco (USA)*, pages 1419-1424, 1986.

[10] J. Espinosa León, A. Sánchez López. Estrategias para la exploración de ambientes desconocidos en robótica móvil. *Tesis de Maestría, Benemérita Universidad Autónoma de Puebla*, 2006.

[11] A. Franchi, L. Freda, G. Oriolo, and M. Vendittelli. A randomized strategy for cooperative robot exploration. *Universitá di Roma, La Sapienza*,2006.

[12] L. Freda, F. Loiudice, and G. Oriolo. . A randomized method for integrated exploration. *2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006.

[13] L. Freda and G. Oriolo. Frontier-based probabilistic strategies for sensor-based Exploration. *IEEE Int. Conf. on Robotics and Automation.* pages 3892- 3898, 2005.

[14] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. *W. H. Freeman, New York, NY.*, 1983.

[15] F. Harary. Graph Theory. . *Addison-Wesley, Reading, MA*, 1994.

[16] A. Howard, M. Mataric, and S. Sukhatme. . An incremental deployment algorithm for mobile robot teams. *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2849-2854, 2003.

[17] D. Hsu, L. E. Kavraki, J. C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. *Workshop on the Algorithmic Foundations of robotics*, pages 141–154, 1998.

[18] L. E. Kavraki, P. Švetska, J-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[19] K. Kant and S.W Zucker. Toward Efficient Trajectory Planning: The Path-Velocity Decomposition. *The International Journal of Robotics Research* , vol. 5, no. 3, pages 72-89, 1986.

[20] J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai. . A practical, decision-theoretic approach to multi-robot mapping and exploration. *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 3232-3238, 2003.

[21] K.Nagatani, Y. Iwai, and Y. Tanaka. Sensor based navigation for car-like mobile robots using generalized voronoï graph. *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, 2001.

[22] E. Kruse, R. Gutshe, and F. M. Wahl. Efficient, iterative, sensor based 3-d map building using rating funtions in configuration space. *Proceedings of the IEEE, Internacional Conference on Robotics & Automation*, pages 1067–1072, 1996.

[23] J. C. Latombe. Robot motion planning. *Kluwer Academic Publications*, 1991.

[24] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. *Department of Computer Science*, 1998.

[25] S. M. Lavalle and Z. S. Peng Cheng. RRT-based trayectory desing for autonomous automobiles and spacecraft. *Archives of control sciences*, 11(3):51–78, 2001.

[26] S. M. LaValle and J. J. Kuffner. RRT-connect: An efficient approach to single-query path planning. *Proceedings of the IEEE, Internacional Conference on Robotics & Automation*, pages 995–1001, 2000.

[27] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Algorithmic and Computational Robotics: New Directions*, 20(5):378–400, 2001.

[28] S. M. Lavalle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2001.

[29] Stéphane Leroy. Deadlock Outils géométriques pour la planification de chemins de robots mobiles non holonomes. *Thèse de Doctorat, Université Paul Sabatier de Toulouse*,1998.

[30] T. Lozano-Perez. Spatial planning: A configuración approach. *IEEE Transactions on Computers*, 32(2):108–120, 1983.

[31] V. Lumelsky and A. Stepanov. Path planning strategies for a point mobile automaton moving amongst unknown obstacles of arbitrary shape. *Algorithmica*, 3(4):403–430, 1987.

[32] E. Mazer, J. M Ahuactzin, and P. Bessi´ere. The ariadne's clew algorithm. *J. Artificial Intell. Res.*, (9):403–430, 1998.

[33] P. O'Donell and T. Lozano-Perez. Deadlock-free and collision-free coordination of two robot manipulators. *IEEE Int. Conf. on Robotics and Automation* , pp. 484-489, 1989.

[34] G. Oriolo, M. Venditelli, L. Freda, and G. Troso. The SRT method: Randomized strategies for exploration. *Proceedings of the IEEE, Internacional Conference on Robotics & Automation*, 2004.

[35] M. H. Overmars and P. Švetska. A probabilistic learning approach to motion planning. *Algorithmic Foundations of robotics*, pages 19–37, 1995.

[36] N. S. V. Rao, W. Shi S. Kareti, and S. S. Iyengar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. *Department of Computer Science, Old Dominion University*, 1993.

[37] J. A. Reeds and L. A. Shepp. Optimal pahts for car that goes both forwards and backwards. *Pacific J. Math*, 145(2):367–393.

[38] P. Renton, M. Greenspan, H. A. Elmaraghy, and H. Zghal. Plan-N-Scan: A robotic system for collision-free autonomous exploration and workspace mapping. *Journal of intelligent and robotics systems*, (24):207–234, 1999.

[39] A. Sanchez and R. Zapata. Sensor-based probabilistic roadmaps for car-like robots. *Proceedings of the IEEE, International Conference on Intelligent Robots and Systems*, 2000.

[40] J. Schwartz and M. Sharir. On the 'Piano Movers'Problem: Coordinating the Motion of Several Independent Bodies: the special case of Circulare Bodies Moving amidest Polygonal Obstacles. *International Journal of Robotics Research* , vol 2, no. 3, pages 46-75.

[41] S. Sekhavat, P. Svestka, J. P. Laumond, and M. H. Overmars. Multilevel path planning for nonholonomic robots using semiholonomic subsystems. *Int. J. Robot. Res.*, (17):840–857.

[42] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. . Coordination for multi-robot exploration and mapping. *In 17th AAAI/12th IAAI* , pages 852-858, 2000.

[43] P. Svestka, J. P. Laumond, and M. H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. *In Proc. of IEEE International Conference on Roboticss and Automation*, 1995.

[44] D. Vallejo, C. Jones, and N. Amato. An adaptative framework "for single shot"motion planning. *Texas A&M*, 1999.

[45] B. R. Vatti. A generic solution to polygon clipping. *Communications of the ACM*, 3(7):56–63.

[46] B. Yamauchi. A Frontier-based approach for autonomous exporation. *IEEE, Int. conf. on Robotics and Automation*, pages 146-151, 1997.

[47] B. Yamauchi. Frontier-based exploration using multiple robots. *Conf. on Autonomous Agents*, pages 47-53, 1998.

[48] Y. Yu. *An information theorical incremental approach to sensor-based motion planning for eye-in-hand systems.* PhD thesis, Simon Fraser University, 2000.

[49] Y. Yu and K. Gupta. Sensor-based probabilistic roadmaps: Experiments with an eye-in-hand system. *Journal of Advance Robotics, Robotics Society of Japan.*, 2000.

[50] R. Zlot, A. Stenz, M. Dias, and S. Thayer. . Multi-robot exploration controlled by a market economy. *IEEE Int. Conf. on Robotics and Automation* , pages 3016-3023, 2002.