Benemérita
Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

"Estrategias para la exploración de ambientes desconocidos en robótica móvil "

Tesis Profesional

Que para obtener el grado de:

Maestro en Ciencias de la Computación

Presenta
Judith Espinoza León

Asesor
Dr. Abraham Sánchez López

Puebla, Pue. **Abril 2006**

Tesis de grado sustentada por

Judith Espinoza León

como requisito parcial para obtener el Grado de

Maestría en Ciencias de la Computación.

Aceptada por el

Dr. Abraham Sánchez López
Asesor

Dra. María Auxilio Osorio Lama
Presidente

Dr. Manuel I. Martin Ortiz
Secretario

Abril de 2006

Agradecimientos

Gracias a Dios por cuidarme y dejarme vivir esta experiencia tan gratificante al lado de las personas que amo. Por cuidar de mi familia y seres queridos mientras no estaba con ellos.

Gracias al Dr. Abraham Sánchez López, en primer lugar, por ser mi profesor y desarrollar en mi la inquietud por esta área, en segundo lugar, por aceptarme como tesista y creer que podía desarrollar este trabajo, en tercer lugar, por apoyarme en todo momento en la realización del mismo y por último, por su paciencia al escucharme y ayudarme en otros aspectos de mi vida. Muchas gracias Dr. Abraham.

Gracias a la Dra. Maria Auxilio Osorio Lama, por ser parte de mi formación academica y por darme ese gran apoyo para concluir este proyecto, sin usted hubiera sido más difícil. Gracias.

Gracias al Dr. Manuel Martin Ortiz, por ser parte de mi formación academica, por ser un magnifico profesor y por apoyarme con sus observaciones en el proyecto.

Gracias a mi hermana Rosa Espinoza León, por creer en mi y siempre apoyarme como hermana, como amiga y en ocasiones como padre y madre. Gracias por tomar en tus manos una responsabilidad que no era tuya. Perdoname todas las travesuras y errores.

Gracias Cristi, por ser mi hermana y mi amiga. De igual forma, por que nunca me has abandonado. A mis hermanos Angel, Panfilo y Delfino que aunque no los tengo cerca físicamente los llevo en mi y se que de alguna forma me cuidan y apoyan. A todos los quiero mucho hermanos, no se que haría sin ustedes.

Gracias a mis padres por darme la vida y con ello la oportunidad de llegar a ser lo que soy. Gracias por su amor y apoyo. Gracias Abuelo por cuidarme desde donde estes.

Gracias a Chelita, a Alfre y a Ethel, se que un gracias no es suficiente, los quiero mucho. Y a todos mis amigos y amigas, tanto en la escuela como externamente, les agradesco el estar conmigo, compartir mis alegrias y sobre todo por soportar mis enojos, tristesas y malos ratos.

Con todo lo que soy y todo lo que tengo, Gracias Amor, por decirme, “si, adelante, siempre que sea para ser mejor, hazlo”. Por que a pesar de tantos problemas, de inicio a fin, sin exclusión de situación, estas conmigo. Te ... .

A todos los lectores gracias por su paciencia a tan larga lista de agracecimientos.

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 un modelo apropiado del ambiente, estructuras de datos compactas para representar los modelos y 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 [11]. PM tiene su origen en la robótica, donde una característica fundamental de los robots autónomos es la planificación de movimientos libres de colisión para alcanzar un objetivo especifico. Sin embargo, encontramos aplicaciones fuera de la robótica tradicional, en dominios tan dispersos como el diseño de manufacturas, animación por computadora, simulación de cirugías médicas y biología computacional, entre otros.

En su enfoque más simple, PM es un problema esencialmente geométrico: dada una descripción geométrica de un objeto y un ambiente estático, nuestro objetivo es encontrar un camino para que el objeto se mueva desde una configuración inicial hasta

una configuración final. A esto se le llama *el problema básico de la planificación de movimientos* [11]. Una dificultad importante al desarrollar un algoritmo general sobre PM (o un planificador de movimientos como frecuentemente se le conoce) es el gran número de grados de libertad (dofs)[1] que puede tener un objeto. Desafortunadamente el algoritmo más rápido que cumple con la propiedad de completitud es exponencial con respecto al número de grados de libertad de un objeto en movimiento [4]. Además, el movimiento de los objetos a menudo esta sujeto a restricciones físicas. Por ejemplo, un carro no puede moverse lateralmente.

Lozano-Pérez [17] introduce el concepto de *espacio de configuraciones*, en el cual el robot es tratado como un simple punto y el problema de planificación de movimientos se transforma, para encontrar un camino uni-dimencional en el espacio de configuraciones libre. El espacio de configuraciones (C-space) se compone de todas las posibles posiciones del robot. En el caso de los robots móviles cuando se les considera como un simple punto, el espacio físico y el espacio de configuración son idénticos.

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. Especificamente, 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

[1] Del inglés, degrees of freedom.

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, que para nuestra aplicación son estáticos. 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 clásica. 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.

Para mayor eficiencia y seguridad, se han incrementado las herramientas de percepción (varios tipos de sensores, incluyendo por ejemplo, cámaras, sensores laser, etc.), para obtener la información más pertinente acerca del ambiente. Sin embargo, no es muy fácil procesar los datos bajo restricciones en tiempo real. Estas restricciones a menudo conducen a una degradación sobre la exactitud y la riqueza de la información.

En muchas aplicaciones prácticas, es cada vez más importante el uso de sensores para actualizar la información sobre el ambiente y guiar el movimiento del robot a través de estrategias de retroalimentación en línea. Actualmente, se han obtenido importantes avances en algoritmos de control de robots usando sensores, sin embargo, aún es difícil encontrar algoritmos prácticos de planificación de movimientos que empleen automáticamente estas capacidades para lograr una tarea. De lo anterior, surge la

motivación principal para el desarrollo de este proyecto de tesis, ya que implementaremos algoritmos con nuevas estrategias de exploración para robots móviles, que usan sensores, en ambientes desconocidos.

1.1. Visión global

Debido a la dependencia exponencial de los algoritmos de planificación de movimientos en el número de grados de libertad se ha considerado al muestreo aleatorio como una técnica fundamental para atacar este problema. Sacrificando una cantidad limitada de completitud, el muestreo aleatorio mejora significativamente la eficiencia de los planificadores de movimientos, haciéndolos prácticos para un amplio rango de aplicaciones.

La planificación de caminos para espacios de configuración de altas dimensiones ha sido muy exitosa con métodos aleatorios [2, 7, 8, 16, 21]. Este conjunto de algoritmos captura la conectividad del espacio de configuración libre del robot en una estructura especial finita, un grafo, tal como el algoritmo de hilo de Ariadna (ACA) [3], los métodos de roadmap probabilístico (PRM) [8, 21] y los árboles aleatorios de exploración rápida (RRT) [14].

En el grafo, cada nodo simboliza una configuración libre del robot y una arista entre dos nodos representa el hecho de encontrar un camino libre de colisión entre las correspondientes configuraciones por medio de un método local simple. Existen diferentes estrategias para colocar las configuraciones (nodos) y para conectarlas (por ejemplo, OBPRM[2], el algoritmo de Hsu [7], etc.).

El grafo (o árbol) en los enfoques roadmaps se construye sin representar explícitamente los obstáculos y por lo tanto, es particularmente útil en la representación de espacios de configuraciones con altas dimensiones.

En el enfoque PRM, el problema planteado es de aprendizaje activo de orden-libre,

ya que lo que se observa del ambiente depende sólo de la última acción (búsqueda aleatoria), esto es una consecuencia de la disponibilidad previa del mapa del ambiente. Por lo cual, los planificadores aleatorios son considerados como estrategias de exploración orientadas a metas, basados en la selección aleatoria de acciones.

La navegación usando sensores permite al robot explorar un ambiente desconocido, con la ayuda de un simple y débil sensor. Uno de los algoritmos más famosos es el propuesto por Lumelsky ("Algoritmo Bug") [18]. Choset y Burdick plantean en [5] el grafo jerárquico generalizado de Voronoï (HGVG), el cual es un roadmap básico en la planificación de movimientos. Estos autores se enfocan exclusivamente en la construcción de roadmaps.

Otro enfoque para la construcción de roadmaps fue propuesto por Yu y Gupta [30], se usa para resolver la planificación de movimientos libres de colisión utilizando sensores para robots articulados (manipuladores). Su enfoque construye el mapa incrementalmente representando la conectividad del espacio libre de configuración a medida que el sensor va descubriendo el espacio libre.

Otros trabajos presentan una variante del grafo generalizado de Voronoï (GVG) [9], los autores proponen un algoritmo de planificación de movimientos para robots de tipo carro (car-like), que construye caminos suaves locales. Básicamente, se genera un camino inicial usando un algoritmo convencional, se aplica la teoría del GVG, después el camino se deforma suavemente para introducir las restricciones no-holonómicas. Desafortunadamente, su solución no es simple y computacionalmente tiene un costo muy alto en comparación con el enfoque convencional del GVG.

Una versión del algoritmo hilo de Ariadna se propone en [1], el cual busca incrementalmente en el espacio libre y calcula un camino a una configuración meta para un robot planar de dos grados de libertad. Los autores asumen que el espacio de trabajo es parcialmente conocido y puede ser ampliado usando un sensor de tipo laser.

En [25] se propone una versión de Lazy PRM, para tratar problemas de planificación de movimientos en ambientes desconocidos para robots tipo carro. El algoritmo genera un lazy roadmap inicial para muestrear el espacio de trabajo. En cada paso se actualiza el roadmap, "eliminando aristas"que pudieran conducir a configuraciones en colisión.

Los roadmaps tienen varias propiedades: *(a)* conectividad, *(b)* accesibilidad y *(c)* la habilidad de tomar cualquier punto de inicio en el roadmap (del inglés departability). Estas propiedades implican que el planificador puede construir un camino entre cualesquiera dos puntos del espacio libre encontrando, en primera instancia, un camino libre de colisiones en el mapa (a), recorrer el mapa en la vecindad del punto meta (b) y entonces construir un camino libre de colisiones de un punto en el mapa al destino (c).

Los roadmaps probabilísticos adaptados al caso de la navegación con sensores, construyen una estructura de datos llamada "grafo (o árbol) aleatorio usando sensores", por medio de configuraciones aleatorias. Esta estructura representa un roadmap del área explorada asociada a una región segura, es decir, al espacio libre percibido por el robot durante la exploración.

Otro algoritmo que captura la conectividad del espacio de configuración libre del robot en una estructura grafo finita, son los árboles aleatorios de exploración rápida (RRT) [16], los cuales son particularmente convenientes para manejar las restricciones no holonómicas (incluyendo la dinámica) y los altos grados de libertad. Un RRT genera nodos iterativamente en un enfoque más complejo. El crecimiento del árbol se basa en dos configuraciones. Una configuración, x_{rand}, es seleccionada de forma aleatoria en el espacio de configuraciónes. La otra, x_{near}, es la configuración en el árbol más cercana a x_{rand}. En cada iteración se genera un conjunto de configuraciones C. El RRT se expande iterativamente aplicando entradas de control que conducen suavemente al sistema hacia los nodos seleccionados de manera aleatoria, la configuración libre en C a cierta distancia de x_{near} se agrega al árbol.

1.2. Bosquejo

En este trabajo, tratamos con el problema de planificación de movimientos usando sensores para un robot de tipo carro en un ambiente inicialmente desconocido. Se utiliza el enfoque de los planificadores RRTs para el manejo de las restricciones diferenciales del sistema. En el capítulo 2 presentamos un análisis general de estos métodos. En el capítulo 3 describimos algunos de los principales trabajos realizados en la planificación de movimientos en ambientes desconocidos usando sensores. Otro método para exploración de ambientes desconocidos usando sensores, el método SRT, se estudia con mayor detenimiento en el capítulo 4. En este capítulo también desarrollamos una variante del método SRT. En el capítulo 5 se presentan las estrategias de exploración desarrolladas como trabajo de tesis. Finalmente, la conclusión y discusión de trabajos futuros se presentan en el capitulo 6.

Capítulo 2

Árboles aleatorios de exploración rápida

Presentaremos los progresos actuales en el diseño y análisis de algoritmos de planificación de caminos basados en árboles aleatorios de exploración rápida (RRT), técnica desarrollada por Steven M. LaValle y su grupo de colaboradores en la universidad de Illinois, EU, [12, 13, 15, 16, 14]. La base de estos métodos es la construcción incremental de árboles de búsqueda que intentan explorar rápida y uniformemente el espacio de estados, ofreciendo beneficios similares a los obtenidos por otros métodos exitosos de planificación aleatoria, como los métodos de roadmap probabilisticos (PRM) [8, 21]. Además, los RRTs son, particularmente, convenientes para problemas que involucran restricciones diferenciales. Expondremos las propiedades teóricas básicas para planificadores que usan RRTs. Así, como la discusión y comparación de varios planificadores que usan este enfoque.

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:** $x_{ini} \in X$ y $X_{meta} \subset X$

3. **Detector de colisión:** Una función, $D : X \rightarrow \{verdadero, falso\}$, 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_{ini} 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_{libre} 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 cercania 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 más cuerpos complicados en un mundo 2D o 3D. La tarea de la planificación de caminos es calcular un camino continuo desde una configuracón inicial, q_{ini} a una configuración meta q_{meta}. De esta manera, $X = C$, $x_{ini} = q_{ini}$, $x_{meta} = q_{meta}$, y $X_{libre} = C_{libre}$, este último denota al conjunto de configuraciones en el cual los cuerpos no estan 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_{libre}. 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 estan los cuerpos geometricos de violar las restricciones del ambiente. Este, se utiliza para asegurar que las configuraciones intermedias estan 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, $x_{nuevo} = x + u\Delta t$, 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 restriciones 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 más general, sería $\dot{x} = f(x, u)$. A esta forma se le denomina, *ecuación de transición de estado* o *ecuación de movimiento*. Al 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.

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 mayoria de los modelos son propios de la teoría de control. El planificador podría usarse para calcular una estrategía 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

[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.

algunas secciones del muestreo, y produciría nuevos estados. También pueden manejarse problemas que involucren restriciones de cerradura cinemática.

2.2. Árboles aleatorios de exploración rápida

El árbol aleatorio de exploración rápida (RRT, del inglés, Rapidly-Exploring Random Tree) fue presentado en [12] 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 arbol hacia ellos. Por lo menos, se han propuesto otras dos técnicas aleatorias de planificación de caminos (planificación holonómica): el algoritmo de Hilo de Ariadna [19] y los planificadores en [30]. 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 [26, 27], en donde, el método roadmap probabilístico se combina con técnicas de conducción no holonómica para planificar caminos con robots moviles rodantes.

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 selec-

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

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

Figura 2.1: Algoritmo básico de construción del RRT

cionarse 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 implicitamente 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_{nuevo} y u_{nuevo}, respectivamente. Pueden ocurrir tres situaciones: *Alcanzado*, el nuevo vértice alcanza al estado muestreado x, (para la planificación no-holonómica, tendriamos un umbral, $\|x_{nuevo} - x\| < \epsilon$ para un ϵ pequeño, $\epsilon > 0$); *Avanzado*, un nuevo vértice $x_{nuevo} \neq x$ es agregado al RRT. *Atrapado*, NUEVO_ESTADO falla en producir un nuevo estado que se encuentre en X_{libre}.

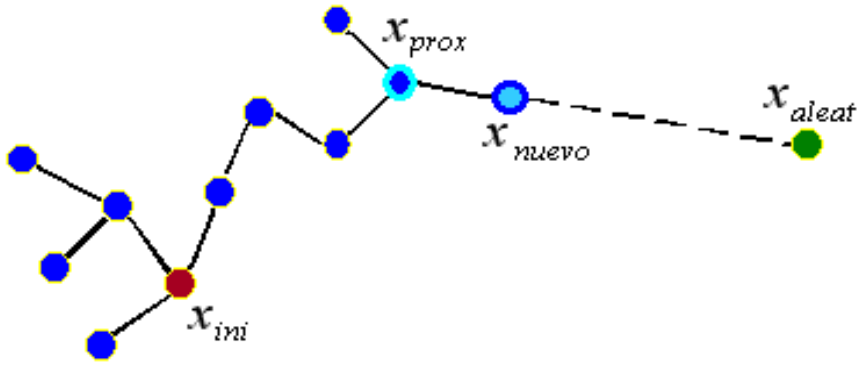


Figura 2.2: Operación de la función EXTENDER

2.3. Diseño de planificadores de caminos

Generalmente consideramos al RRT como un bloque en la construcción de un planificador eficiente. Por ejemplo, podriamos usar un RRT para escapar de un mínimo local en un planificador de caminos aleatorio con campos de potencial. En [28], 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, tales como: restricciones diferenciales, el tipo de algoritmo para la detección de colisiones y/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, cubrirán un componente conectado de X_{libre}, llegando arbitrariamente cerca de cualquier x_{meta} 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_{meta} o un estado toma-

do de cualquier parte del espacio de estados. Incluso con una probabilidad pequeña (por ejemplo, 0.05) de eligir x_{meta}, RRT_GoalBias usualmente converge al objetivo mucho más rápido que el RRT básico. Pero, sí 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 concentricos. Hay muchas investigaciones 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 estan lejos de colisionar, se puede tomar un paso grande. Además de seguir esta idea, ¿qué tan lejos debe estar x_{nuevo} de x_{prox}?, ¿Trataríamos de conectar x_{prox} con x_{aleat}?. 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

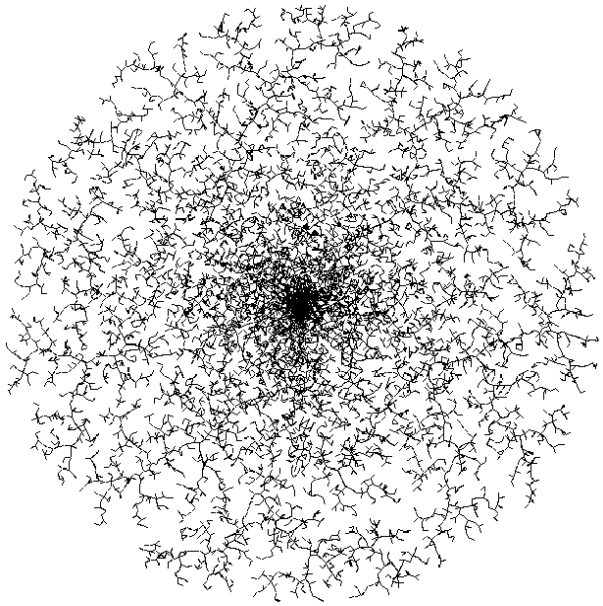


Figura 2.3: Un RRT en 2D construido usando un muestreo sesgado

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

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

```
RRT_BIDIRECCIONAL(x_ini, x_meta)
1  T_a.inic(x_ini); T_b.inic(x_meta);
2  para k=1 a K
3     x_aleat ← ESTADO_ALEATORIO();
4     si (EXTENDER(T_a, x_aleat) <> Atrapado) entonces
5        si (EXTENDER(T_b, x_nuevo) = Alcanzado) entonces
6           Regresa CAMINO(T_a, T_b);
7     INTERCAMBIAR(T_a, T_b);
8  Regresa Fallo;
```

Figura 2.5: Planificador bidireccional usando un RRT

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_{ini} y el otro apartir de x_{meta}; 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_BIDIRECCIONAL***, puede compararse con el algoritmo CONSTRUIR_RRT de la figura 2.1. El RRT_BIDIRECCIONAL 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 [16] 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 compensación 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 [15] 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-nolonómica serían aún más dificiles 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_{aleat}. 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 estan a una distancia fija de x_{aleat}, de acuerdo con la métrica. Si una conexión tiene éxito entonces los dos arboles 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

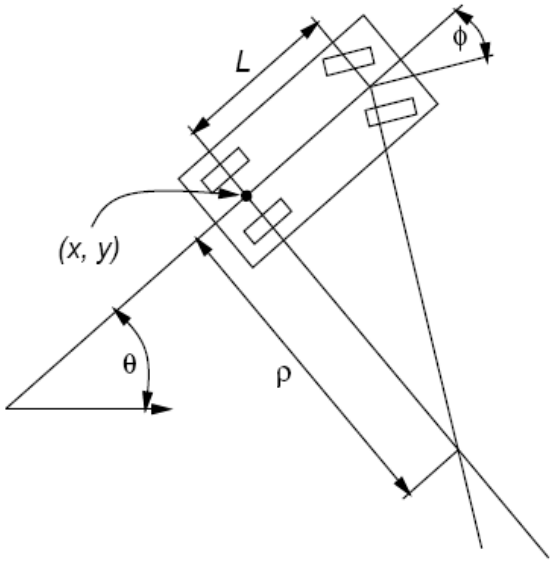


Figura 2.6: Modelo cinemático para un robot tipo carro

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 tri-dimensional, ver la figura 2.6, cualquier estado se representa por medio de (x, y, θ) y su ecuación de transición de estado es:

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

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 [6]. Si el carro se mueve hacia atrás y hacia delante, $s = -1$ o $s = 1$, obtenemos un carro Reeds-Shepp [23]. 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 bi-dimensional 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 smooth", agregando el ángulo de conducción como variable de estado. La entrada es la velocidad angular, ω, del angulo 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. Las figuras 2.8.a y 2.8.b ilustran 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.

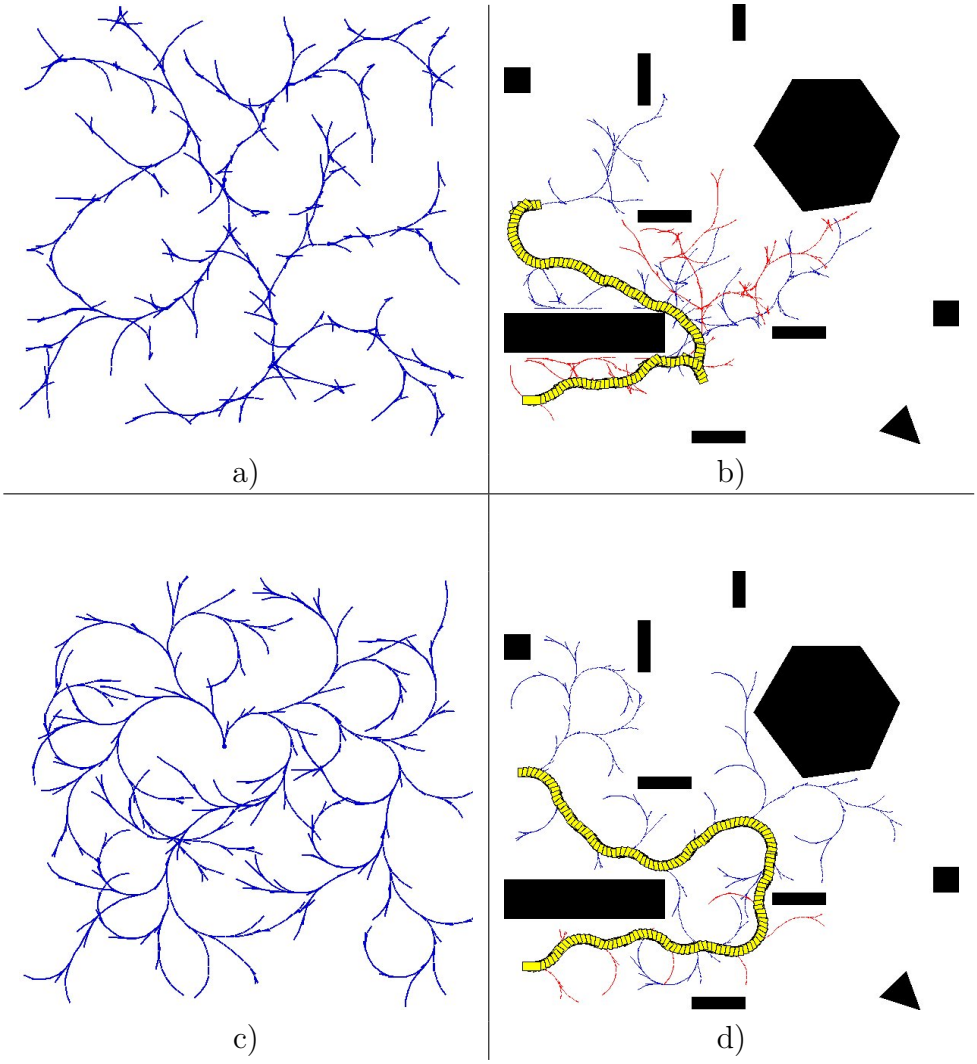


Figura 2.7: a) Árbol de exploración rápida para un carro Reeds-Shepp; b) Camino calculado usando un carro Reeds-Shepp; c) RRT para un carro Dubins; d) Camino calculado para el carro Dubins. Ambos caminos se obtuvieron con el planificador RRT-Bidireccional

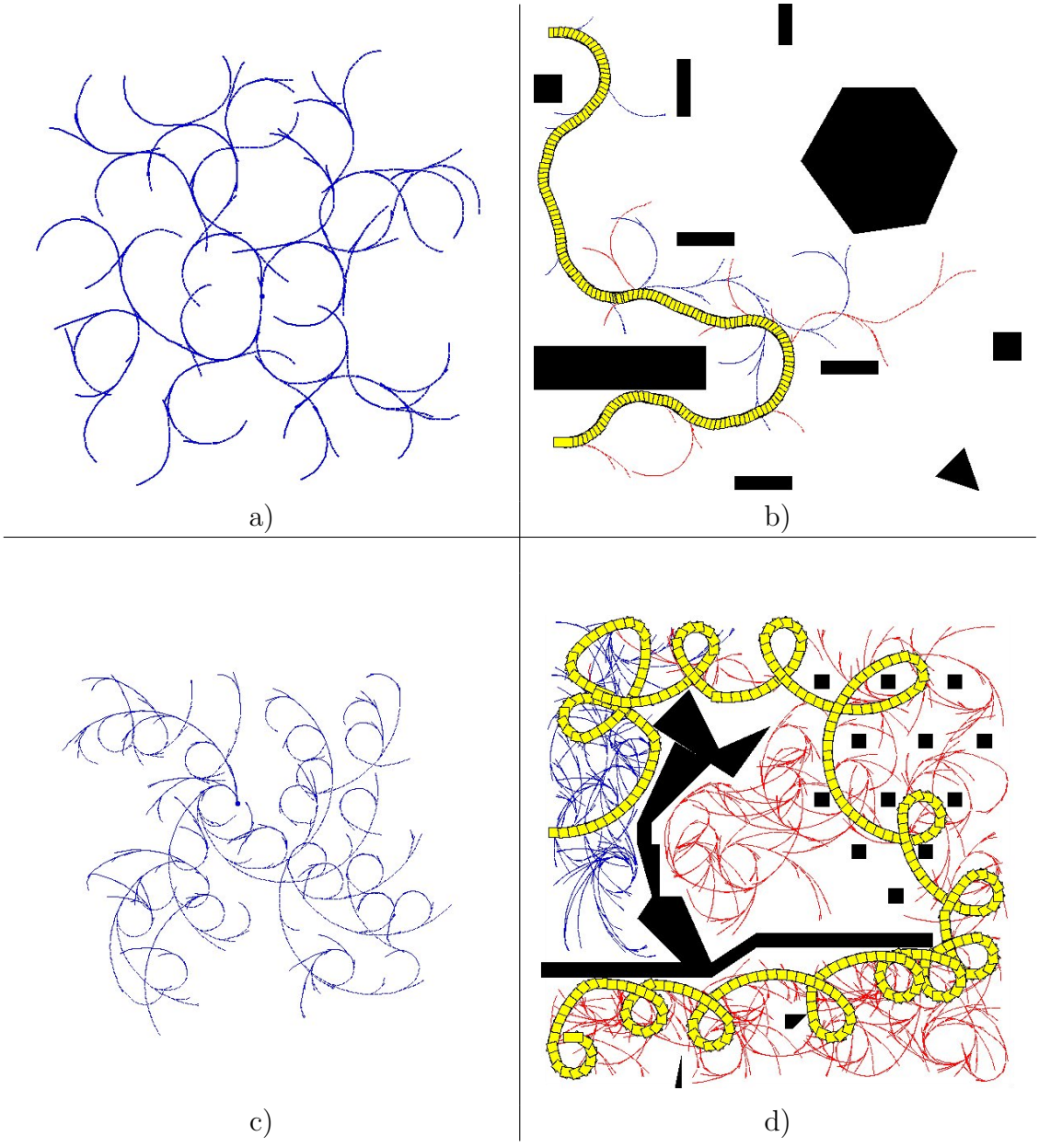


Figura 2.8: a) Árbol de exploración rápida para el modelo "smooth"del carro; b) Camino calculado usando un carro "smooth"; c) RRT para un carro Dubins, avanza sólo virando a la izquierda; d) Camino calculado con el carro de avance delantero y conducción solo a la izquierda. Los caminos se obtuvieron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

2.5. Resumen de las propiedades de los RRTs

Esta última sección presenta un resumen de las propiedades representativas de los RRTs así como algunas desventajas.

- La expansión de los RRT es fuertemente sesgada hacia las porciones no exploradas del espacio de estados.
- La distribución de vértices en un RRT se aproxima a la distribución de muestreo, permitiendo un desarrollo consistente.
- Un RRT es probabilisticamente completo bajo condiciones muy generales.
- El algoritmo RRT es relativamente simple, lo cual facilita su análisis (esta es también una característica de los roadmaps probabilísticos).
- Un RRT siempre permanece conectado, aún cuando el número de aristas sea mínimo.
- Un RRT puede considerarse como un modulo de planificación de caminos, el cual puede ser adaptado a una gran variedad de sistemas genéricos de planificación.
- Se pueden construir algoritmos completos de planificación de caminos sin requerir de la habilidad para dirigir el sistema entre dos estados prescritos, lo cual amplia grandemente la aplicación de los RRT.

Los principales inconvenientes de los métodos RRTs es la fuerte dependencia en la métrica y que la razón de convergencia se expresa en virtud de parámetros que no pueden medirse fácilmente.

Capítulo 3

Exploración en ambientes desconocidos

Hay dos formulaciones básicas del problema de la planificación de caminos y la navegación basadas en la disponibilidad del modelo del ambiente. En un ambiente conocido, el modelo se da como entrada, así el problema de planificación de movimientos se convierte en uno de programación geométrica. En un ambiente desconocido, la ausencia del modelo requiere que el robot obtenga información local del ambiente empleando un sistema de sensado. Una de las principales diferencias entre ambas formulaciones, radica en que, el camino para llegar de una ubicación origen a una ubicación destino puede ser preplaneado antes de ejecutar cualquier movimiento. En el último caso, el camino debe ser calculado incrementalmente por medio de la exploración de nuevas zonas del ambiente.

En un ambiente desconocido tenemos dos aspectos críticos: *a)* el cómputo se sustenta en información local (o parcial) y *b)* el sensado es parte integral de la navegación. A causa del primer aspecto, los algoritmos para ambientes desconocidos con frecuencia se les llama *algoritmos en línea* [22]. Como lo vislumbra el segundo aspecto, en un ambiente desconocido, el algortimo es necesario para catalogar las operaciones del sensor. En general, en ambientes desconocidos distintos y con robots navegadores equipados con

diferentes tipos de sensores se requieren diversos algoritmos.

En el presente capítulo, describiremos brevemente algunos métodos de exploración y planificación de movimientos en ambientes desconocidos que han sido parte fundamental en el desarrollo de nuestro proyecto de investigación. Algunos de estos métodos fueron creados para sistemas de robots manipuladores y otros para robots moviles, pero todos ellos contienen puntos claves para moldear nuevos métodos de exploración. Este capítulo también puede considerarse como una introducción, ya que posteriormente reportaremos otros métodos de exploración en ambientes desconocidos como resultado de nuestro trabajo.

3.1. SBIC-PRM

La investigación hecha por Yu y Gupta [31], [30] para la planificación de movimientos usando sensores, presenta un sistema real "Eye-in-hand"para un brazo manipulador con muchos grados de libertad. Ellos exploran la adaptabilidad del método PRM para este sistema, construyendo incrementalmente el roadmap con la información reportada por los sensores del espacio libre en el ambiente físico, este roadmap representa la conectividad del espacio libre conocido, dentro del cual, el robot puede moverse para detectar más espacio físico. A este método le llamaron SBIC-PRM (del inglés, sensor-based incremental construction of probabilistic roadmap).

3.1.1. Algoritmo SBIC-PRM

Dado un espacio parcialmente conocido, el espacio de configuraciones del robot implicitamente se divide en tres regiones. La región blanca (el espacio libre, C_{libre}), la región negra (los obstáculos, C_{obs}) y la región gris que corresponde al espacio desconocido. Un conjunto de configuraciones se genera aleatoriamente, de tal forma que se crean

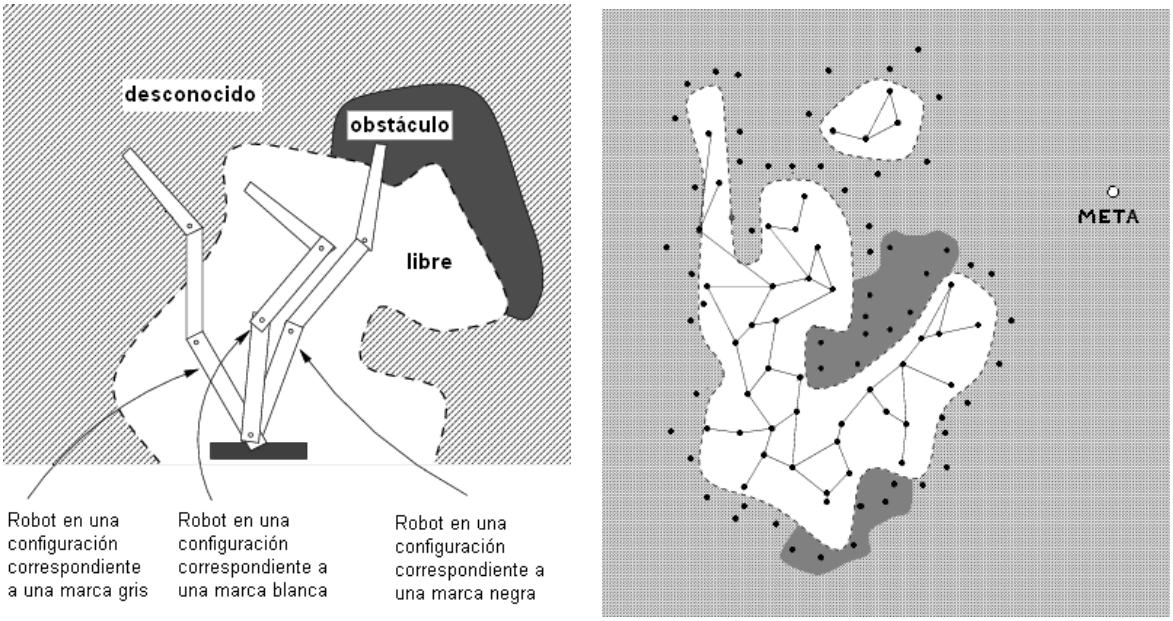


Figura 3.1: a) Ejemplo de marcas gris, blanca y negra en el espacio físico. b) Los tres tipos de marcas expuestas sobre el espacio de configuraciones. La región blanca es el C_{libre} conocido, las marcas sobre esta región son blancas. El roadmap es el grafo que se muestra dentro de esta región. La región obscura indica el espacio conocido de los obstáculos, C_{obs}, las marcas aqui son negras. La región gris es el C-espacio desconocido en ella las marcas son grises.

tres tipos de configuraciones (denominadas marcas) de acuerdo a la región, así tenemos marcas blancas, negras y grises, ver figura 3.1.

En principio, el robot se encuentra en la configuración inicial dentro de un espacio libre inicial. Existe un planificador que genera el correspondiente roadmap. De ahi, el paso clave del algoritmo es expandir incrementalmente el roadmap R que representa la conectividad del actual espacio libre. Para cualquier iteración dada, la i-ésima, el planificador, en primer lugar, intenta alcanzar la configuración meta ejecutando el planificador local desde todas las marcas en la región blanca. Si existe alguna que regrese éxito, el planificador habrá encontrado un camino libre de colisión. Si no sucede, el planificador toma un nuevo escaneo desde una marca blanca. El planificador mueve al

robot a este nodo de vista, para actualizar el espacio libre conocido, en sí no se calcula explicitamente el espacio físico libre, en vez de ello, se agregan más marcas de manera que el roadmap se expanda. El proceso completo puede pensarse como un frente de onda del espacio libre, representado por las marcas grises, dirigido hacia la configuración meta.Una vez actualizado el roadmap, el planificador nuevamente intenta alcanzar la meta desde todas las marcas blancas y el proceso entero se repite, hasta que la meta es alcanzada o el planificador realiza un número máximo de iteraciones.

3.2. Plan-N-Scan

El sistema presentado por Renton et al. [24] llamado Plan-N-Scan, es muy similar en hardware al utilizado posteriormente por Yu y Gupta, trabaja con un robot PUMA el cual tiene montado en la "muñeca" un sensor de alcance. El sistema incrementalmente sensa una secuencia de objetivos seleccionados de acuerdo a sensados previos. El resultado es un "voxel map", una malla tridimensional, conveniente para el chequeo de colisiones. La planificación de movimientos se apoya en un algoritmo A^*.

Plan-N-Scan mantiene una estructura de datos tipo pila donde almacena el objetivo final y objetivos intermedios (voxels que serán sensados). La pila se inicializa con el voxel meta y en cada iteración el algoritmo extrae un voxel sub-meta. El planificador primero selecciona una posición de vista para el voxel sub-meta y planifica un camino libre de colisión o determina si se requieren objetivos adicionales donde sensar y asi alcanzar la sub-meta. Las posiciones de vista tienen que satisfacer dos criterios: (1) la línea de visión del sensor al objetivo no puede estar obstruido y (2) el manipulador debe estar libre de colisión cuando se encuentre en la posición de vista, si ninguna posición cumple con los criterios entonces se puede utilizar la busqueda A^* o declarar a la sub-meta *no sensable*. Una vez que se tienen la posición de vista, se invoca una

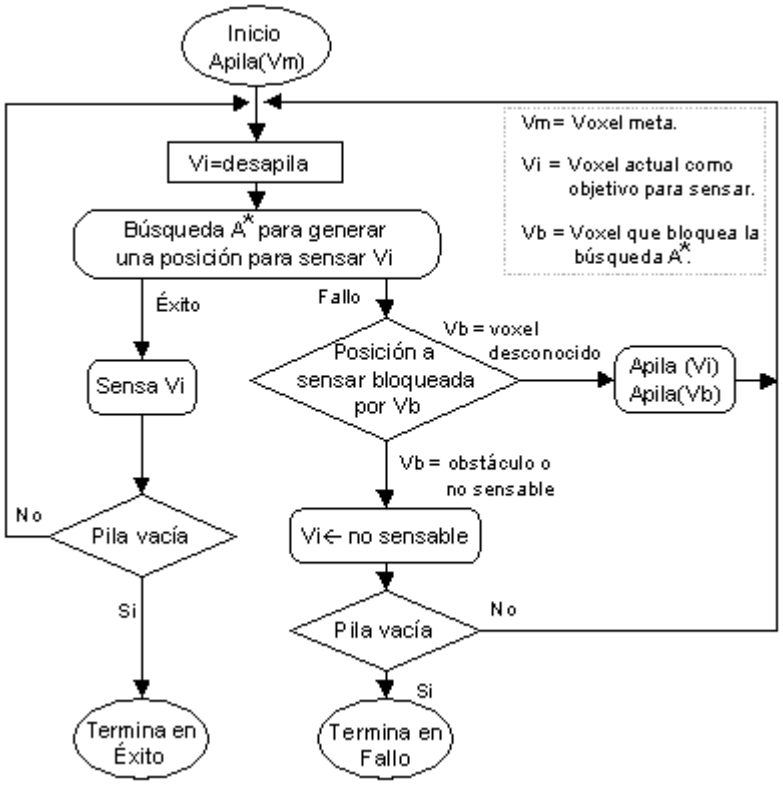


Figura 3.2: Diagrama de flujo del algoritmo Plan-N-Scan.

búsqueda A^* para encontrar un camino libre de colisión a la posición objetivo, si esto sucede entonces el voxel sub-meta es extraido de la pila, si la posición objetivo no puede alcanzarse debido a colisiones potenciales con el espacio desconocido, entonces se insertan en la pila objetivos intermedios. El algoritmo termina con éxito cuando la pila esta vacia y el sistema alcanzó el objetivo final. Podría suceder que la pila este vacia debido a que el voxel meta se haya declarado no sensable, por lo que el algoritmo termina en fallo. La figura 3.2 muestra el diagrama de flujo del planificador.

3.3. Algoritmo de exploración utilizando funciones de valoración

Kruse, Gutschet y Wahl describen un sistema similar al método anterior (sólo en simulación) [10]. Su algoritmo de planificación de caminos también construye un roadmap utilizando campos de potencial. Generalmente, no es necesaria la ejecución del algoritmo de planificación, ya que la configuración a la que se mueve el robot a partir de la configuración actual puede alcanzarse directamente a través de un camino lineal en el espacio de configuraciones. Después de cada sensado, el grafo sólo se extiende, en vez de recalcularse completamente. Heurísticamente examinan los alrededores de la configuración donde se efectuó el sensado, debido a que sólo utilizan cinemática directa es dificil localizar nuevas áreas libres en el C-espacio. Tras varias extensiones, el grafo tiende a alargarse y degenerar. Por lo tanto, si la planificación de caminos falla, se require recalcular completamente el grafo. Esto último no pasa con el método de SBIC-PRM.

Kruse et al. dan mayor atención al algoritmo de planificación de vista, es decir, planificar la siguiente configuración donde se realizará el próximo sensado. Para la planificación de vista utilizan restricciones sobre los movimientos del robot y funciones de valoración en el C-espacio. Las restricciones son: a) La siguiente configuración de vista esta libre de colisión; b) La siguiente configuración de vista puede alcanzarse por un camino libre de colisión; c) El sensado en la siguiente configuración de vista debería obtener mucha información nueva del área desconocida. Esta restricción se manipula introducciendo una función de valoración $R_{inf}(q)$, la cual es la fracción del volumen desconocido en el volumen total sensado. d) La siguiente configuración de vista no debería estar lejos de la actual. Esta restricción se observa a través de la función $R_{dist} = 1 - d(q, q_{act})/d_{max}$, donde q_{act} es la configuración actual del robot, $d(q, q_{act})$ es una

función que calcula la distancia entre las configuraciones y d_{max} es la distancia máxima en el C-espacio. La próxima configuración de vista se elige de modo que la función $R(q) = R_{inf}(q)(wR_{dist}(q) + 1 - w)$ se maximice sobre el C-espacio discreto, sujeta a las dos primeras restricciones, donde $w \in [0, 1]$ es un parámetro de peso. La restricción sobre R_{dist} es critica sólo cuando el tiempo para moverse a cada nueva configuración de sensado toma una gran porción en la iteración escaneo-planificación-movimiento. La tercera restricción, R_{inf} es razonble pero no es suficiente.

3.4. Roadmaps probabilísticos usando sensores para robots tipo carro

Sánchez y Zapata proponen un método para el problema de planificación de movimientos usando sensores para robots tipo carro [25]. Adaptan los métodos PRM y lazy-PRM a la exploración de ambientes desconocidos para aprovechar la información reportada por los sensores y así calcular un camino factible libre de colisión.

Para el proceso de planificación tanto la configuración inicial como la final son nodos en el roadmap. El robot comienza en la configuración inicial donde se genera un lazy-roadmap inicial intentando hacer un muestreo del espacio de trabajo. Para cualquier iteración dada i, el planificador intenta alcanzar el nodo más cercano a la configuración meta a través de un camino construido por un planificador local (caminos Reeds and Shepp). El paso clave es guiar la búsqueda con la información del sensor, usando una estrategia A^*, y actualizar el grafo eliminando aristas que llevarían a una configuración en colisión. Un proceso asegura que tanto el nodo como la trayectoria están en un área libre de colisión para que el robot se mueva sin riesgo a chocar con un obstáculo. Si hay éxito en calcular un camino factible el robot se mueve a la configuración y el

proceso se repite consecutivamente hasta alcanzar la configuración meta o el algoritmo termina con fallo al exceder un tiempo máximo determinado. Una caracteristica clave del algoritmo es la destreza dada al sistema para regresar a la configuración alcanzada si ya no es posible el avance hacia la configuración meta desde la configuración actual.

Las dos características más poderosas de este enfoque son las capacidades de escapar de mínimos locales, por medio del proceso de retroceso y salir con "fallo" sino se puede encontrar un camino.

3.5. GGV para la navegación de robots tipo carro usando sensores

Nagatani, Iwai y Tanaka desarrollaron un método para la navegación de robots tipo carro usando sensores [9] basándose en el grafo generalizado de Voronoi (GGV). Desde el punto de vista de la completitud y seguridad el GGV tiene la ventaja de describir caminos para la navegavición de robots móviles, sin embargo es imposible aplicarlos directamente debido a la limitación existente en el ángulo de conducción de tales robots. Para resolver este problema, proponen un planificador local de caminos suaves para deformar localmente el GGV usando una función de evaluación y asi poder seguir los caminos trazados por el GGV lo más fiel posible.

Para realizar la navegación del robot móvil utilizando el GGV siguen el siguiente procedimiento: (1) Por medio de los sensores se adquiere la información local del ambiente; (2) Empleando un algoritmo convencional calculan el GGV local; (3) Se construye el espacio de configuraciones (C-espacio); (4) Se calculan candidatos de caminos suaves; (5) Aplicando una función de evaluación a cada camino candidato en el C-espacio se elige el mejor; y (6) Ejecutar el camino planeado en el área local. Los pasos 1 al 6 se

repiten hasta que se construya completamente el GGV.

Usualmente, los movimientos de exploración del robot siguen el GGV. Cuando encuentra un punto de reunión (punto donde se encuentran tres o más aristas del GGV), el robot guarda la ubicación de este punto, y elige una arista inexplorada. Cuando encuentra un punto límite (punto donde dos o más obstáculos convexos son conectados), el robot regresa; si ya no hay aristas por explorar, la tarea de exploración termina. A pesar de que el algoritmo encuentra caminos factibles para los robots tipo carro, la solución no es simple y el costo computacional es mayor que el enfoque convencional del GGV.

En el capítulo siguiente se aborda el método STR, diseñado para la exploración de ambientes desconocidos usando sensores para robots móviles omnidireccionales. Se presenta una variante del método junto a los resultados obtenidos en simulación.

Capítulo 4

Método SRT para ambientes desconocidos

Oriolo, Vendittelli, Freda y Troso presentan en [20] 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), presentados en detalle en el capítulo dos. Pueden obtenerse diversas estrategias de exploración adaptando al método general, diferentes técnicas de percepción. En [20] se exponen y comparan dos técnicas; la primera, SRT-Ball, los autores la denominan una técnica conservadora y conveniente para usar sensores con ruido. La segunda técnica de percepción llamada SRT-Star es menos conservadora, es decir, confia más en la información reportada por los sensores. La estrategia desarrollada en esta tesis sigue la línea de SRT-Star, como se verá en las siguientes secciones.

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. Desde un punto de vista práctico, la exploración es una tarea central en aplicaciones tales como misiones planetarias, intervenciones en áreas hostiles, construcción automática de mapas, entre otras.

Una definición ampliamente aceptada sobre la exploración es la siguiente: "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 cómo seleccionar la siguiente acción. La exploración basada en fronteras se logra cuando el criterio es la maximización de la utilidad de las acciones. Sin embargo existe otra opción, es decir, usar un mecanismo de selección aleatoria (también llamado

camino aleatorio). Las ventajas de esta elección son (1) simplicidad y (2) el hecho de que cualquier secuencia de acciones se ejecutará eventualmente. La última propiedad mencionada lleva a la completitud: se encontrará una solución cuando esta exista. Por otra parte, la selección de una acción puramente aleatoria puede ser muy ineficiente.

El enfoque del problema emana de las técnicas de planificación de movimientos aleatorios (PMA), las cuales construyen roadmaps en el espacio de configuraciones libre usando muestreo aleatorio y verificando las colisiones. En PMA, el problema planteado es de aprendizaje activo de orden-libre: lo que se observa del ambiente depende sólo de la última acción (búsqueda aleatoria), ya que el mapa del ambiente esta disponible progresivamente y el robot se traslada para recopilar más información. Por lo cual, los planificadores aleatorios son considerados como estrategias de exploración orientadas a objetivos utilizando la selección aleatoria de acciones. Como ya se había mencionado, la completitud (probabilística) de estos planificadores es inherente a su naturaleza, además, se puede lograr una mayor eficiencia agregando algunas heurísticas al esquema aleatorio básico. El método RRT es un ejemplo típico.

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, con 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, (localizacion 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 efectua 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 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 $\alpha < 1$

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

```
1   q_act = q_ini;
2   para k=1 a K_max
3      S ← PERCEPCION(q_act);
4      AGREGA(T, (q_act, S));
5      i ← 0;
6      repetir
7         θ_rand ← DIR_ALEATORIA;
8         r ← RADIO(S, θ_rand);
9         q_cand ← DESPLAZAR(q_act, θ_rand, α · r);
10        i ← i + 1;
11     hasta que (VALIDA(q_cand, d_min, T) o i = I_max)
12     si VALIDA(q_cand, d_min, T) entonces
13        MOVER_A(q_cand);
14        q_act ← q_cand;
15     sino
16        MOVER_A(q_act.padre);
17        q_act ← q_act.padre;
18  Regresa T;
```

Figura 4.1: Algoritmo básico de construcción del SRT

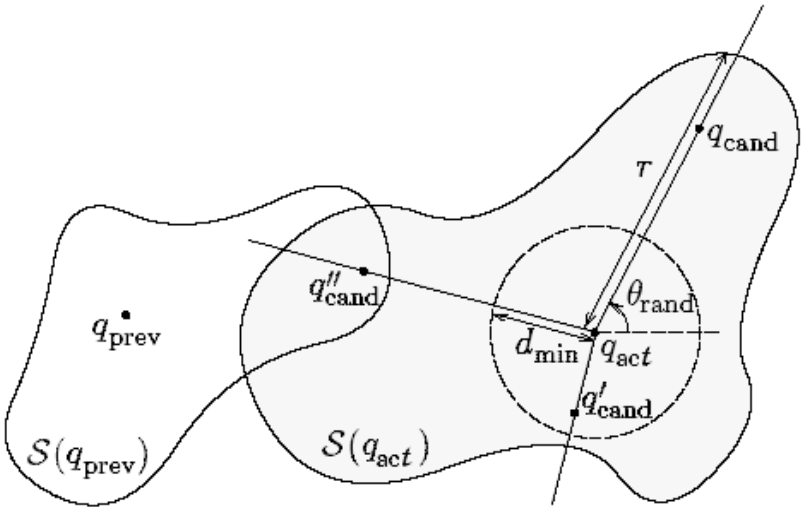


Figura 4.2: Generación de configuraciones candidatas con el método SRT. En este caso, q_{cand} es valida, mientras q' y q'' no lo son, la primera se encuentra a una distancia menor a d_{min} de q_{act} y q'' se ubica en la región segura local de otro nodo.

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 expansion 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 caracteristicas más 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 carácteristicas 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 [20] se presenta una variante del método llamada SRT-Star, la cual involucra una estrategia de percepción que toma completamente 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 4.4. Por lo tanto, en presencia de obstáculos la forma de S se deforma y para diferentes direcciones de exploración las longitudes de los radios varían.

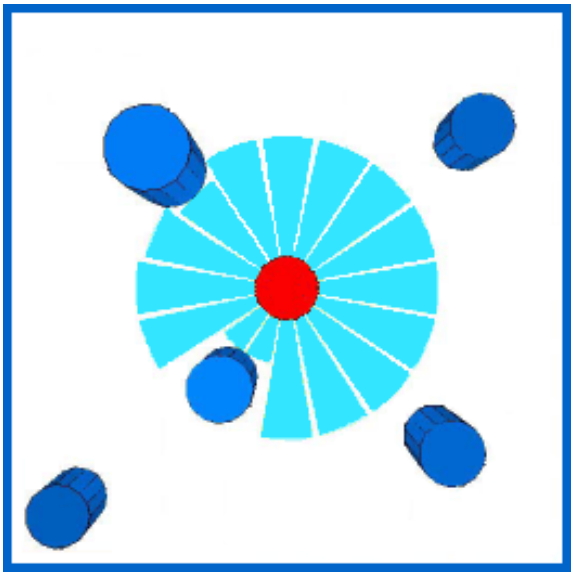


Figura 4.3: Región segura local S obtenida con la estrategia de percepción SRT-Star. Note que la extensión de S en algunos conos es reducida por el rango de alcance del sensor.

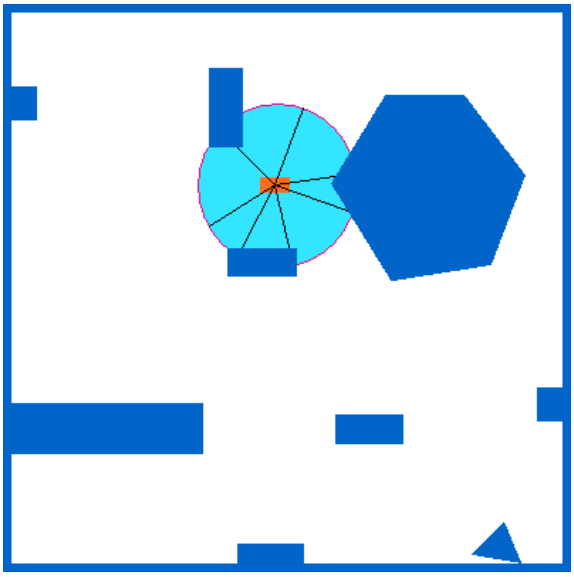


Figura 4.4: Diferentes radios obtenidos en la región segura local S con la estrategia de percepción SRT-Radial y la aplicación de la libreria GPC.

4.4. Experimentos y resultados

Para ilustrar el comportamiento de exploración de la estrategia SRT-Radial se presentan los resultados de dos variantes al método, SRT_Extensivo y SRT_Meta. Los algoritmos se desarrollaron en Visual C++ V.6.0. aprovechando la estructura de la librería MSL,[1] así como su interfaz gráfica para seleccionar los algoritmos, visualizar el ambiente de trabajo y la animación del camino calculado. Para simular el módulo de percepción del sistema de sensado se utilizó la librería GPC [2] . En los apéndices B y C se presentan con más detalles las librerías MSL y GPC.

4.4.1. Detalles de implementación

Las modificaciones hechas al método SRT radican principalmente en la condición de termino del algoritmo y el tipo de robot móvil considerado. Para efectuar las simulaciones suponemos la disponibilidad de un sensor telemétrico rotacional a 360^0 situado sobre el robot, en general, se puede fácilmente extender el sistema para soportar cualquier tipo y número de sensores. También es importante recordar que una consideración fundamental para el funcionamiento del método es la hipótesis de una perfecta localización.

En la primera variante, SRT_Extensivo, consideramos un robot móvil que se traslada en cualquier dirección (robot holonómico), como originalmente se planteó en el método SRT. El algoritmo SRT_Extensivo termina con éxito cuando el proceso automático de retroceso nos lleva a la configuración inicial, es decir, al punto de partida del robot. Esto significa, que el algoritmo agotará todas las opciones disponibles de acuerdo a la

[1] MSL (Motion Strategy Libray) es una librería para el desarrollo y prueba de algoritmos de planificación de movimientos, incluye varios planificadores que usan RRT, PRM y FDP; así como un ambiente gráfico en OpenGL para la descripción de la escena.

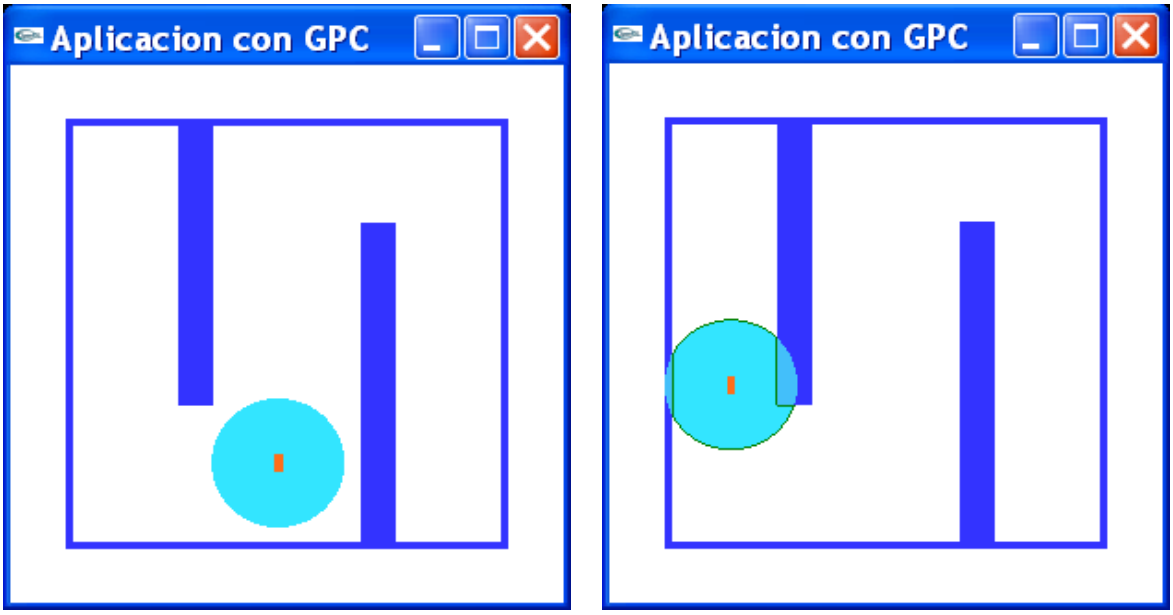
[2] GPC (General Polygon Clipping) es una librería de operaciones para el manejo de recorte de polígonos.

selección aleatoria de la dirección de exploración, dando al algoritmo la tarea principal de explorar el mayor porcentaje posible del ambiente y obtener su correspondiente roadmap. El algoritmo termina con "fallo"después de un número máximo de iteraciones.

En la segunda variante atacamos un problema híbrido de la planificación de movimientos, es decir, combinamos la tarea de exploración con aquella que requiere encontrar un objetivo a partir de una posición inicial, problema Inicio-Meta[3] . El algoritmo SRT_Meta, explora el ambiente terminando con éxito cuando la configuración meta se encuentra en la región segura local asociada a la configuración actual donde se lleva a cabo el sensado. En caso de no encontrar la configuración meta, el algoritmo realiza el proceso de retroceso hasta la configuración inicial. Por lo tanto, en SRT_Meta la tarea principal es encontrar el objetivo fijado, quedando en segundo término la exploración exhaustiva del ambiente. En SRT_Meta, el robot explorador no es omnidireccional, se aplica una restricción al ángulo de conducción, $|\phi| \leq \phi_{max} < \frac{\pi}{2}$.

El método SRT_Meta también se aplico a un problema de planificación de caminos, tomando en cuenta todas las consideraciones antes mencionadas. El planteamiento es el siguiente: suponemos que contamos con dos robots; el primer robot puede ser omnidireccional o tener presentar una restricción no-holonómica simple , como se mencionó en el párrafo anterior. Este robot tiene la tarea de explorar el ambiente y obtener la región segura que contenga las posiciones inicial y final. El segundo robot se considera no-holonómico, especificamente un robot de tipo carro, el cual se desplazará por un camino libre de colisión dentro de la región segura. Un planificador local calculará el camino de la configuración inicial a la configuración final, para tal efecto, se adaptó el método RRTExtExt para ejecutarse en la región segura con lo que se evita el proceso de detección de colisiones con los obstáculos. Se eligió el planificador RRTExtExt por ser conveniente para manejar las restricciones no-holonómicas de los robots tipo carro

[3] Ver [30], pp. 31.



a) b)

Figura 4.5: a)Zona de percepción del sensor (S←zona circular), ubicado sobre el robot (rectángulo), en ausencia de obstáculos. b)Zona percibida por el sensor en presencia de obstáculos (SR←área delimitada por la curva cerrada) obtenida con el uso de la libreria GPC.

y ser experimentalmente más rápido que los RRTs básicos.

En el proceso de simulación, el robot junto con el sistema de sensado se desenvuelven en un mundo 2D, donde los obstáculos son considerados estáticos, el único ente móvil es el robot. La descripción geométrica del robot, del espacio de trabajo y de los obstáculos es por medio de polígonos; de la misma forma se decidió modelar la zona de percepción del sensor y la región segura a través de polígonos. Esta representación facilita el uso de la librería GPC para la simulación del modulo de percepción; sea S la zona que el sensor puede percibir en ausencia de obstáculos y SR la zona percibida, el calculo del área de SR se obtiene empleando la operación diferencia de GPC entre S y los polígonos que representan los obstáculos, ver figura 4.5.

4.4.2. Resultados

Las pruebas se realizaron en una computadora personal bajo el sistema operativo Windows XP con un procesador Pentium 4 CPU 2.40GHz, 512MB de memoria RAM y 64MB de memoria de video.

SRT_Extensivo

El algoritmo SRT_Extensivo fue probado en diferentes ambientes tomando los siguientes valores $k_{max} = 250,\ I_{max} = 10,\ \alpha = 0{,}7,\ d_{min} = 3{,}5$. La figura 4.6 muestra dos ambientes. Una serie de experimentos revelaron que el algoritmo trabaja eficientemente explorando los ambientes casi en su totalidad.

a)Ambiente 1 b) Ambiente 2

Figura 4.6: Ambientes y posición inicial del robot para las pruebas del método SRT_Extensivo.

La tabla 4.1 resume los resultados obtenidos respecto al número de nodos del SRT y el tiempo de ejecución. El tiempo de ejecución proporcionado por los experimentos

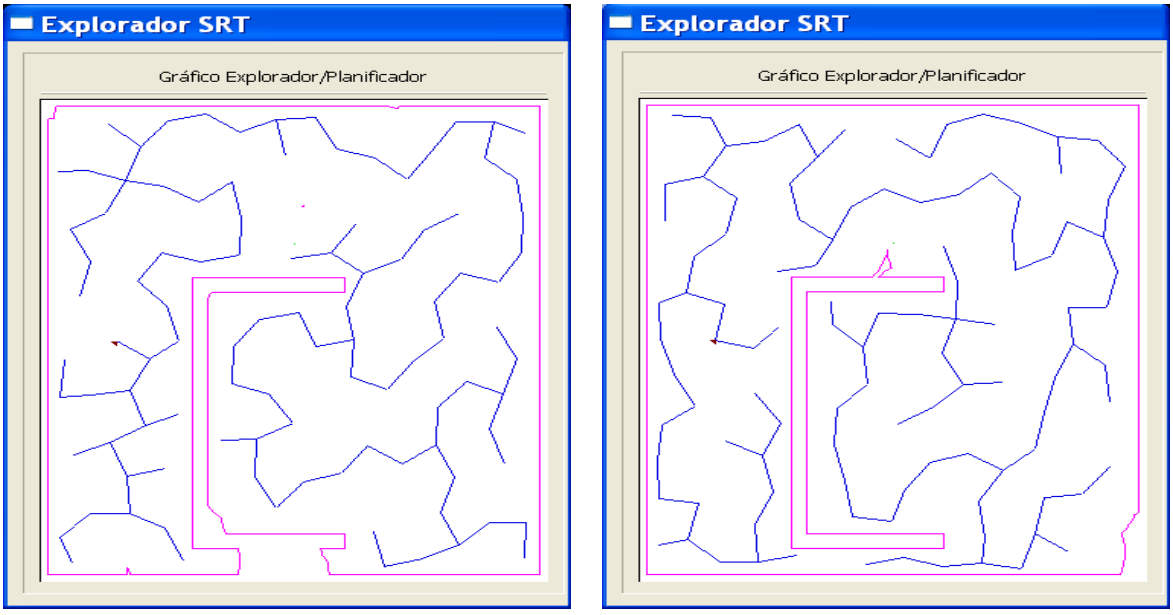
corresponde al tiempo total de exploración incluyendo el tiempo de percepción del sensor.

En las figuras 4.7 y 4.8 se visualizan los árboles aleatorios de exploración usando sensores (SRT) junto con los contornos de la región libre explorada por el robot para los ambientes 1 y 2. En las figuras 4.7.b y 4.8.a se observa como el robot puede explorar por completo el ambiente, cumpliendo con la tarea encomendada, tanto para un ambiente complejo lleno de obstáculos como para uno más simple que contiene un pasaje estrecho. En esta simulación se muestra la ventaja de la estrategía de percepción SRT-Radial debido a que aprovecha la información reportada por los sensores en todas direcciones se pueden generar y validar configuraciones candidatas a través de espacios reducidos.

Por la naturaleza aleatoria del algoritmo en la elección de la dirección de exploración, el algoritmo puede dejar pequeñas zonas del ambiente sin explorar, como en las figuras 4.7.a y 4.8.b.

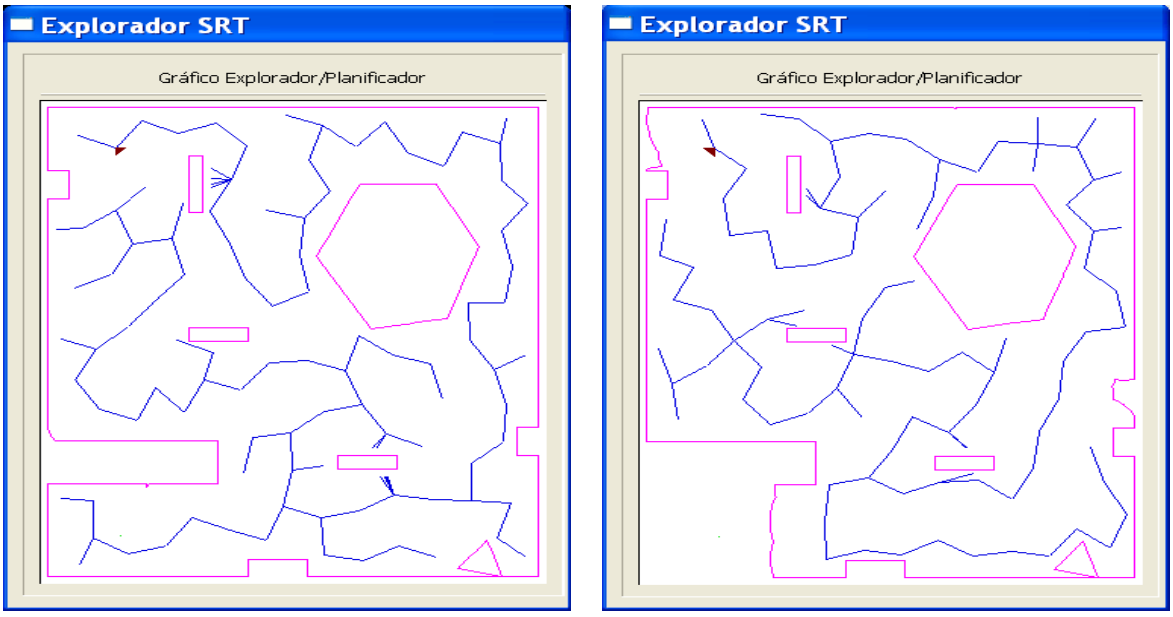
	Ambiente 1	Ambiente 2
Nodos (mín.)	92	98
Nodos (máx.)	111	154
Tiempo (mín.)	132.594 seg.	133.766 seg.
Tiempo(máx.)	200.406 seg.	193.859 seg.

Tabla 4.1: Resultados del método SRT_Extensivo



a) b)

Figura 4.7: SRT y región explorada por el robot en el ambiente 1

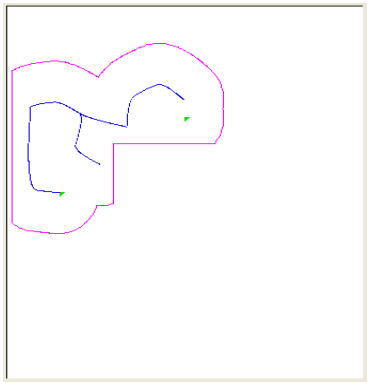


a) b)

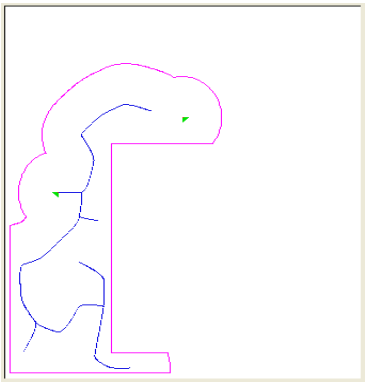
Figura 4.8: SRT y región explorada por el robot en el ambiente 2

SRT_Meta

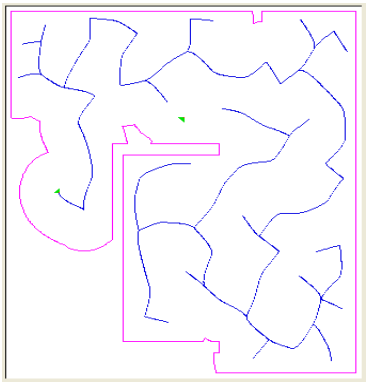
Como se mencionó el algoritmo SRT_Meta termina cuando la configuración meta se encuentra dentro de la región segura de la actual configuración o termina cuando regresa a la configuración inicial. La figura 4.9 muestra algunos SRT obtenidos en los ambientes 1 y 3. Los tiempos y número de nodos cambian, de acuerdo algoritmo elegido, a la selección aleatoria en la dirección de exploración y a la posición inicial y final del robot en el ambiente, marcados en las figuras con un pequeño triángulo.



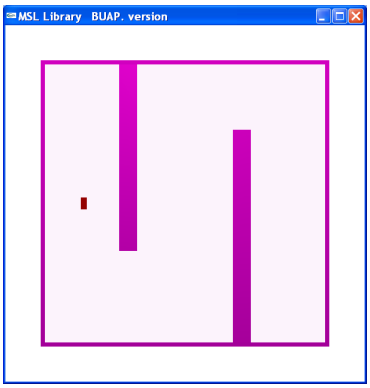
a) Tiempo: 13.812 seg.
Nodos: 13



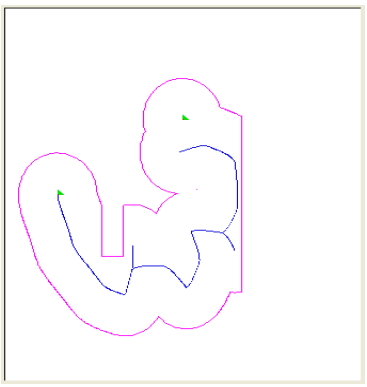
b) Tiempo: 30.531 seg.
Nodos: 24



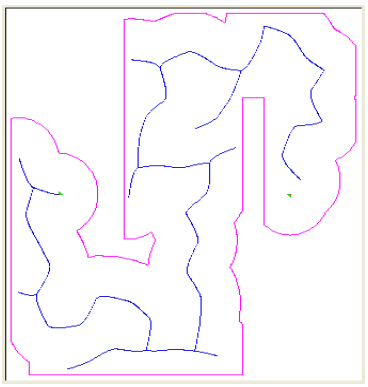
c) Tiempo: 138.812 seg.
Nodos: 83



d) Ambiente 3



e) Tiempo: 25.343 seg.
Nodos: 19

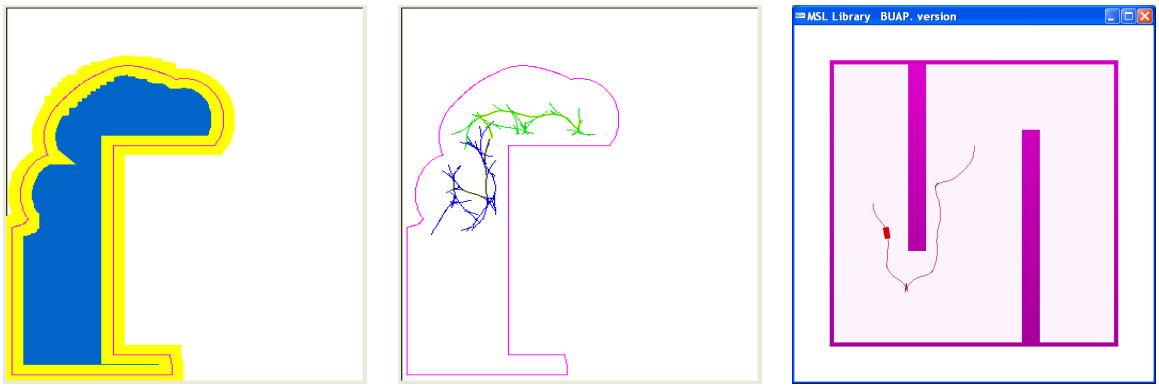


f) Tiempo: 79.171 seg.
Nodos: 41

Figura 4.9: SRT's y regiones exploradas por el robot en el ambiente 1 y 3 con el método SRT_Meta.

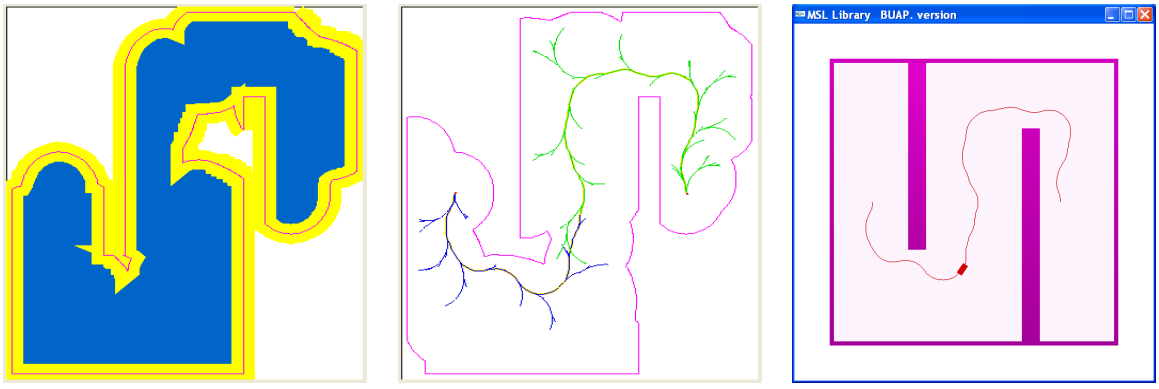
Cuando el algortimo SRT_Meta ha calculado la región segura que contiene la posición inicial y final, un segundo robot de tipo carro tiene la posibilidad de ejecutar nuevas tareas de planificación de caminos ejecutando localmente los planificadores RRTs. La región segura nos garantiza que el robot podrá moverse de manera confiable en el interior de esa área ya que no existe ningún obstáculo y un chequeo de colisiones por parte de los planificadores RRTs es innecesario. Pero no debemos olvidar la geometría del robot al ejecutar movimientos cerca de la frontera entre la región segura y el espacio desconocido, siempre hay posibilidad de encontrar un obstáculo con el cual colisionar. Por lo tanto, surge la necesidad de construir una banda de seguridad en el contorno de la región segura para proteger al robot de posibles colisiones y asegurar su movilidad. Las figuras 4.10, 4.11 y 4.12 muestran la banda de seguridad, el RRT calculado y el camino encontrado para algunos robots móviles con diferentes restricciones, en los ambientes 1, 2 y 3.

En el siguiente capitulo abordaremos el problema de explorar un ambiente desconocido con un robot tipo carro usando sensores, es decir, explorar el ambiente y planificar un camino en una sóla etapa con el mismo robot.



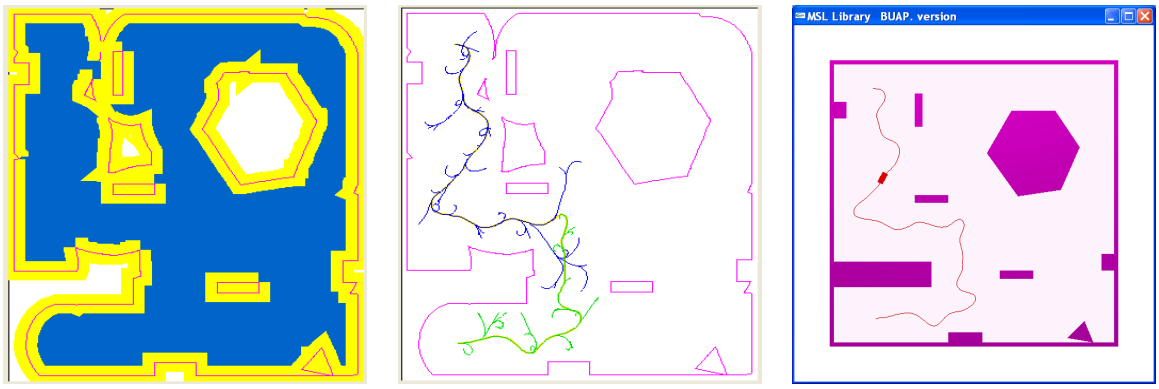
a) b) c)

Figura 4.10: a) Región segura y banda de seguridad, para el SRT de la fig. 4.9.b. b)RRT calculado con el planificador RRTExtExt, en un tiempo de 2.594 seg. con 535 nodos. c) Camino encontrado para un robot de tipo carro.



a) b) c)

Figura 4.11: a) Región segura y banda de seguridad, para el SRT de la fig. 4.9.f. b)RRT calculado con el planificador RRTExtExt, en un tiempo de 5.204 seg. con 593 nodos. c) Camino encontrado para un robot de tipo carro de avance delantero.



a) b) c)

Figura 4.12: a) Región segura y banda de seguridad para un SRT en el ambiente 2. b)RRT calculado con el planificador RRTExtExt, en un tiempo de 13.485 seg. con 840 nodos. c) Camino encontrado para un robot de tipo carro "smooth".

Capítulo 5

Exploración usando sensores con AAL

En este capitulo desarrollamos algunas extrategias de exploración de ambientes desconocidos basadas en el enfoque de los árboles aleatorios de exploración rápida (RRT). Los RRTs fueron desarrollados para la planificación de movimientos contando previamente con el modelo del ambiente. El objetivo del presente proyecto de tesis es aprovechar las bondades[1] de los RRTs, en la exploración de ambientes y el manejo de restricciones no-holonómicas, para la exploración de ambientes desconocidos usando sensores para robots de tipo carro. Se propone construir un árbol aleatorio local (AAL) que represente la conectividad del espacio de estados libre X_{libre} conocido asociado una región segura reportada por el sensor. El árbol se expandirá incrementalmente como el sensor tome nuevas percepciones del ambiente hasta que el estado meta sea un nodo del AAL, es decir, hasta encontrar un camino factible libre de colisiones, o hasta un número de iteraciones máximo.

El robot móvil equipado con un sensor, (sensor telemétrico rotacional a 360^0) que proporciona el área libre de obstáculos alrededor del robot, necesita planificar y eje-

[1] En el capito 2 se describen las características principales de los RRTs, asi como algunas variantes. Y en el apéndice A se presenta un análisis matemático sobre estas características.

cutar movimientos libres de colisión desde un estado inicial hasta un estado final, en un ambiente inicialmente desconocido. El robot debe construir una representación del ambiente de manera incremental con ayuda del sensor. Cada vez que se incrementa el conocimiento del ambiente se debe verificar la posibilidad de planificar un camino entre los estados inicial y final.

El proceso de planificación en ambientes desconocidos consiste escencialmente en dos etapas [1]: 1) Ejecutar el algoritmo de planificación de caminos en el componente conectado libre que contiene al estado inicial y determinar si el estado meta puede alcanzarse. 2) Si el estado meta no puede alcanzarse entonces los sensores incrementan el conocimiento del ambiente, por consecuencia la conectividad del espacio de estados. Se repiten los pasos 1) y 2) hasta encontrar un camino que conecte a los estados inicial y final o hasta que se determine que no es posible incrementar el conocimiento del ambiente.

5.1. Estructura de datos

Planteamos la contrucción de una estructura de datos tipo árbol, llamado árbol aleatorio local (AAL), donde cada nodo contiene un estado libre de colisión, una entrada de control y una descripción de la región segura local asociada al estado. Cada arista en el árbol representa que el robot puede moverse de un estado a otro bajo la entrada de control. Los nodos del árbol pueden ser de dos tipos: 1) Intermedio, utilizado para el movimiento del robot dentro de la RSL y 2) Terminal, nodo donde se puede efectuar un nuevo proceso de percepción del ambiente. Opcionalmente, el nodo puede incluir la referencia a una lista de estados candidatos a un nuevo proceso de percepción del ambiente. Esta lista generalmente ira referenciada en los nodos terminal. La descripción del robot, los obstáculos y la RSL es polígonal, como en el capitulo anterior.

5.2. Proceso de exploración

El sistema empieza en el estado inicial. En el estado actual se efectua un proceso de percepción del ambiente para determinar la región segura local asociada al estado, se verifica si el estado meta se encuentra en la RSL, sino un planificador RRT alcanza un estado candidato y el ciclo se repite, expandiendo el AAL e incrementando el ambiente explorado. Cuando no se tengan más candidatos el sistema regresará al estado previo, este proceso de retroceso puede llevar al robot hasta la posición inicial, en ese caso el algoritmo termina con fallo. También termina con fallo después de un número de iteraciones máximo. El algoritmo termina con éxito cuando el estado meta esta en la región segura local del estado actual y se encuentra un camino que une los estados inicial y final, ver el diagrama de la figura 5.1.

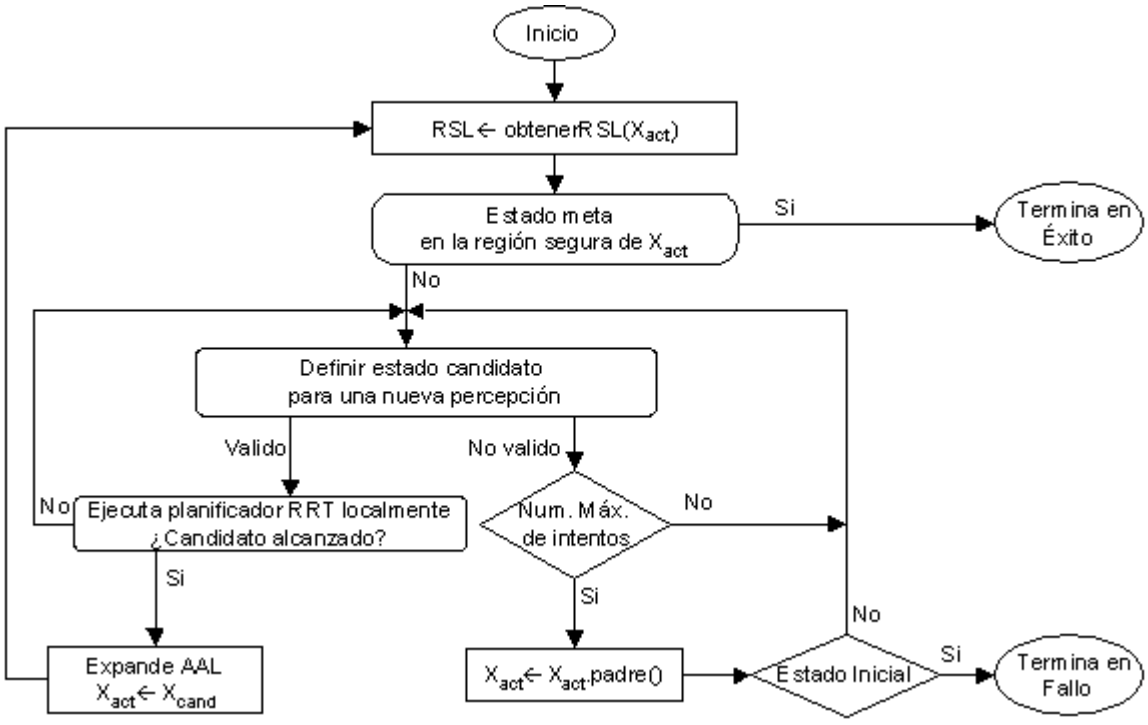


Figura 5.1: Flujo del proceso de exploración

5.3. Algoritmo

El algoritmo explora el área desconocida en forma similar el método SRT y adapta el enfoque RRT para la planificación local de caminos en el espacio de estados libre asociado a la región segura.

En cada iteración k del algoritmo, ver figura 5.2, se efectua un proceso de percepción para obtener la región segura local S circundante al robot en el estado actual, x_{act}. Se crea un nuevo nodo que contiene al estado junto con la descripción de la RSL y se agrega al árbol como un nodo terminal. El algoritmo verifica si el estado meta x_{meta} se encuentra en la región segura local de x_{act}, sino una rutina determina un estado candidato valido dentro de la región segura para efectuar un nuevo proceso de percepción del ambiente. Las restricciones diferenciales del robot de tipo carro se manipulan con un planificador RRT que se ejecuta localmente en el espacio de estados libre para encontrar un camino factible que una los estados actual y candidato. Cada nodo creado por el planificador local se agrega al árbol como un nodo intermedio. Si el proceso de planificación local tiene éxito, el nodo candidato se agrega al árbol como un nodo terminal y el ciclo se repite; en otro caso, el algoritmo determina otro estado candidato valido desde x_{act}, que el planificador pueda unir con un camino factible o hasta que un número de intentos máximo I_{max} se exceda. Si lo último ocurre, el robot regresa al nodo padre de x_{act}, donde empieza el ciclo otra vez. El proceso de retroceso puede llevar al robot hasta la posición inicial, en ese caso el algoritmo termina con fallo, determinando que no se puede adquirir mayor conocimiento del ambiente.

De este algoritmo general surgen diversas estrategias de exploración de acuerdo a la elección en los estados candidatos y en la construcción del árbol de exploración. En las siguientes secciones se describen las estrategias SRRT_Local y SRRT_LocalB desarrolladas como parte del proyecto de tesis, junto a las principales rutinas del algoritmo.

CONSTRUIR($x_{inic}, K_{max}, I_{max}$)

1 $x_{act} = x_{inic}$;
2 $k \leftarrow 0$;
3 **mientras** (NO(META_EN_RSL(x_{act})) o $NO(x_{act} = NULL)$ o $k < K_{max}$))
4 $S \leftarrow$ PERCEPCION(x_{act});
5 AGREGA($\mathcal{T}, (x_{act}, S)$);
6 $i \leftarrow 0$;
7 **repetir**
8 $x_{cand} \leftarrow$ EDO_CANDIDATO(x_{act});
9 $ExisteCamino \leftarrow$ PLANIFICADOR_RRT(x_{act}, x_{cand});
10 $i \leftarrow i + 1$;
11 **hasta que** ($ExisteCamino$ o $i = I_{max}$)
12 **si** $ExisteCamino$ **entonces**
13 MOVER_A(x_{cand});
14 $x_{act} \leftarrow x_{cand}$;
15 **sino**
16 MOVER_A($x_{act}.padre$);
17 $x_{act} \leftarrow x_{act}.padre$;
18 $k \leftarrow k + 1$;

19 **si** (META_EN_RSL(x_{act})) **entonces**
20 $ExisteCamino \leftarrow$ PLANIFICADOR_RRT(x_{act}, x_{meta});
21 **si** $ExisteCamino$ **entonces**
22 **Regresa** EXITO;
23 **si** (x_{act}=NULL)) **entonces**
24 *"Robot en la posición inicial ";*
25 **Regresa** FALLO;

Figura 5.2: Algoritmo básico

5.4. Estrategia SRRT_Local

Existen dos variantes de SRRT_Local, denominada así porque adapta la ejecución local de un planificador RRT en el proceso de exploración usando sensores. Las variantes se diferencian en la forma de construir el árbol de exploración.

EDO_CANDIDATO(x_{act})

Los estados candidatos para un nuevo proceso de percepción del ambiente se generan con la rutina EDO_CANDIDATO(x_{act}). En SRRT_Local la elección de candidatos es simple y sigue los siguientes pasos:

1. Se elige una dirección aleatoria para el primer estado candidato, x_{cand1}. x_{cand1} debe estar dentro de la región segura local dejando un margen en la frontera para asegurar la movilidad del robot, como en el algoritmo 4.1.

2. Se valida el estado, 1) debe estar alejado a una distancia mayor a una distancia mínima establecida y 2) no debe situarse en la región segura local de un nodo terminal.

3. Los estados candidatos siguientes también pueden tomarse en direcciones aleatorias o utilizar un método más determinista, por ejemplo, ubicarlos en direcciones equidistantes apartir de la dirección del primer estado. Y validarse según el paso 2.

Los estados candidatos validos son almacenados en una lista referenciada en el nodo actual, nodo terminal donde se toma el sensado del ambiente. La descripción de la región segura local puede almacenarse sólo en los nodos terminal entonces los nodos intermedios únicamente servirán para darle movilidad al robot respetando las restricciones

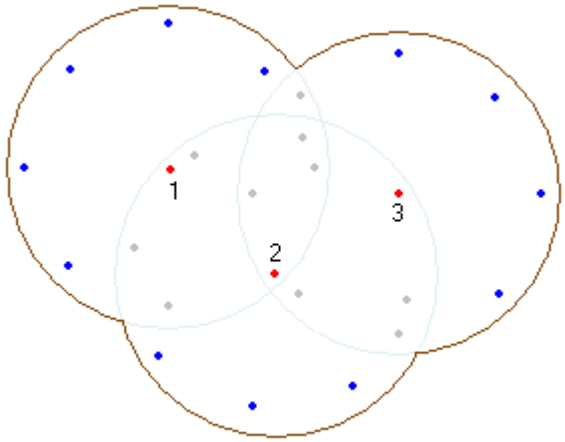


Figura 5.3: Estados candidatos validos

diferenciales. Cuando el robot móvil se desplaza de x_{act} a x_{cand} puede ir enriqueciendo la información del espacio libre agregando la descripción de las RSLs a sus correspondientes nodos intermedios. En este caso, el hecho de examinar sólo los nodos terminal reduce el esfuerzo de computo y mantiene la capacidad de exploración del algoritmo.

El número de nodos candidatos fue elegido experimentalmente, dando buenos resultados. Los ocho candidatos seleccionados estan distribuidos equidistantes uno del otro cerca de la frontera de la región segura local con el espacio desconocido. La figura 5.3 muestra tres percepciones del ambiente en ausencia de obstáculos; al tomar la primera percepción la lista de candidatos de x_1 es de ocho, después el sistema se mueve a x_2 invalidando candidatos de x_1 y del propio x_2, el tercer proceso de percepción también invalida candidatos. De esta forma sólo van quedando los candidatos cerca de la frontera del espacio de estados libre con el espacio desconocido distribuidos homogénamente en todas direcciones para evitar zonas sin explorar. Además, se agregó una heurística simple para guiar el proceso de exploración hacia el estado meta, los estados candidatos se toman ordenados de acuerdo a la distancia euclidiana que los separa del estado meta,

eligiendo primero aquellos más cercanos. Una vez elegido el estado candidato se marca como visitado para invalidarlo. Entonces, el procedimiento de elección y validación de estados para las siguientes percepciones del ambiente hacen que el número de intentos máximo I_{max} del algoritmo 5.2 sea variable dependiendo del número de candidatos validos disponibles localmente.

PLANIFICADOR_RRT

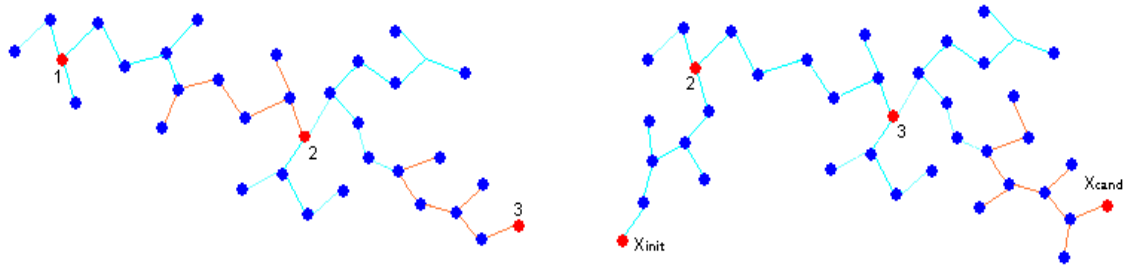
Las variantes de la estrategia SRRT_Local adaptan un planificador RRT_Bidireccional[2] local. En la primera variante, $\mathcal{T}_a$ tiene como raíz a x_{act} y $\mathcal{T}_b$ toma como raíz a x_{cand}, usualmente el planificador conecta ambos árboles en la región segura local de x_{act}. Por tanto, el árbol de exploración que representa la conectividad del espacio de estados libre esta conformado por pequeños árboles bidireccionales locales, actuando los nodos terminal, donde se realiza el sensado, como nodos de unión.

La diferencia de la segunda variante radica en la forma de construir el árbol de exploración. En cada iteración del algoritmo, el árbol de exploración, que también funciona como el árbol de planificación $\mathcal{T}_a$ con raíz en x_{ini}, crece al agregarle los nodos del árbol local $\mathcal{T}_b$, el cual tiene como raíz a x_{cand}. La figura 5.4 muestra los árboles aleatorios de exploración creados con ambas variantes.

5.5. Estrategia SRRT_LocalB

La estrategia de exploración SRRT_LocalB adapta el planificador RRT_GoalBias[3] (de ahí la B en el nombre) para ejecutarse localmente en la región segura. Al contrario de la estrategia anterior, SRRT_LocalB primero ejecuta el planificador local y después

[2] Capitulo 2, sección 2.3.2
[3] Capitulo 2, sección 2.3.1



a) b)

Figura 5.4: a) Árbol aleatorio de exploración con tres nodos terminal, el árbol crece al unir los dos árboles bidireccionales pequeños, en el nodo de percepción 2. b) Árbol aleatorio de exploración con cuatro nodos terminal, el árbol crece al agregarle el árbol local con raíz en x_{cand}.

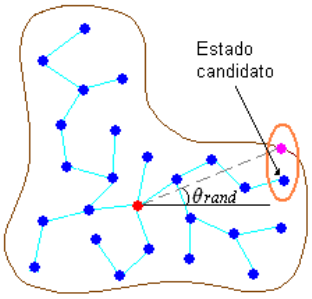
determina los estados candidatos. La característica que diferencia a las variantes de SRRT_LocalB es la forma de elegir los estados candidatos para un nuevo proceso de percepción.

Una vez realizado el sensado del ambiente desde al estado actual, se ejecuta el planificador RRT_GoalBias para construir el árbol aleatorio a partir de x_{act} en la región segura. De RRT_GoalBias obtenemos el primer estado candidato siendo el estado en el árbol más cercano a x_{meta}, según la métrica del algoritmo RRT. No utilizamos esta métrica para guiar el proceso de exploración completo debido a la influencia que tiene el parámetro del ángulo de conducción del robot móvil y que puede acarrear problemas de mínimos locales. Los siguientes candidatos se toman con la rutina EDO_CANDIDATO(x_{act}).

EDO_CANDIDATO(x_{act})

En la primera variante de SRRT_LocalB si el estado candidato no resultara valido según el paso 2 de la rutina EDO_CANDIDATO(x) de la sección anterior, el algortimo elige otro estado candidato de la siguiente manera:

- Se elige una dirección aleatoria θ_{rand}.
- Se toma el estado en la frontera de la región segura en la dirección θ_{rand}.
- El estado candidato será el vecino más próximo en el árbol al estado en la frontera.
- Se valida el estado de acuerdo al paso 2 de EDO_CANDIDATO(x) de la sección anterior.



El proceso se repite hasta encontrar un candidato valido o hasta un número máximo de iteraciones, como originalmente se plantea.

En la segunda variante de SRRT_LocalB la rutina EDO_CANDIDATO(x_{act}) aprovecha la distribución de las ramas del árbol local construido a partir del x_{act}, ver apendice A. Como los nodos hojas del árbol son los nodos más cercanos a la frontera con el espacio desconocido, EDO_CANDIDATO(x_{act}) los selecciona como estados candidatos, ver figura 5.5. Nuevamente los estados candidatos validos son almacenados en una lista. Se sigue la misma heurística del SRRT_Local para guiar el proceso de exploración, moviendose primero a los estados más cercanos a x_{act}.

Las funciones restantes del algoritmo 5.2 son iguales para las estrategias de exploración y sus variantes.

META_EN_RSL(x_{act})

La función META_EN_RSL(x_{act}) regresa verdadero si el estado meta se encuentra en la región segura local del estado actual. Esto se puede validar de dos formas, con la ayuda de la librería GPC utilizando la función de intersección de polígonos, o calculando la distancia entre la posición actual y la posición meta y comparandola con la longitud del rayo que une la posición actual con un punto en la frontera de la RSL en la dirección de la posición meta.

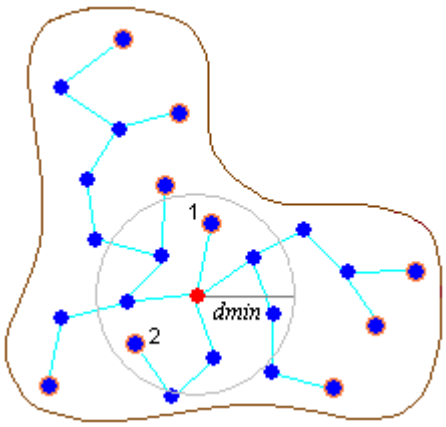


Figura 5.5: La mayoría de los nodos hojas contienen estados candidatos validos, solo los nodos 1 y 2 no lo son por estar a una distancia menor a d_{min}.

MOVER_A(x)

La rutina MOVER_A(x) simplemente sigue las entradas de control generadas por el planificador local para llegar al estado x. Opcionalmente en cada movimiento actualiza la región segura con nuevas percepciones del ambiente.

PERCEPCION(x_{act})

El proceso de percepción del ambiente se realiza de la forma explicada en el capitulo anterior.

5.6. Resultados experimentales

Las pruebas se hicieron en una computadora personal bajo el sistema operativo Windows XP con un procesador Pentium 4 a 2.40GHz, 512MB de memoria RAM y 64MB de memoria de video.

Los ambientes utilizados para la simulación de las estrategias de exploración se muestran en la figura 5.6.

a)Ambiente 1 b) Ambiente 2

Figura 5.6: Ambientes y posición inicial del robot

Las variantes de las estrategias de exploración simplemente son referidas como variante 1 o variante 2 de la estrategia SRRT_Local o de la estrategia SRRT_LocalB.

SRRT_Local

En este caso primero mostraremos algunos ejemplos de los árboles aleatorios de exploración obtenidos, junto con los caminos encontrados por los algoritmos, véase la figuras 5.7 y 5.8.

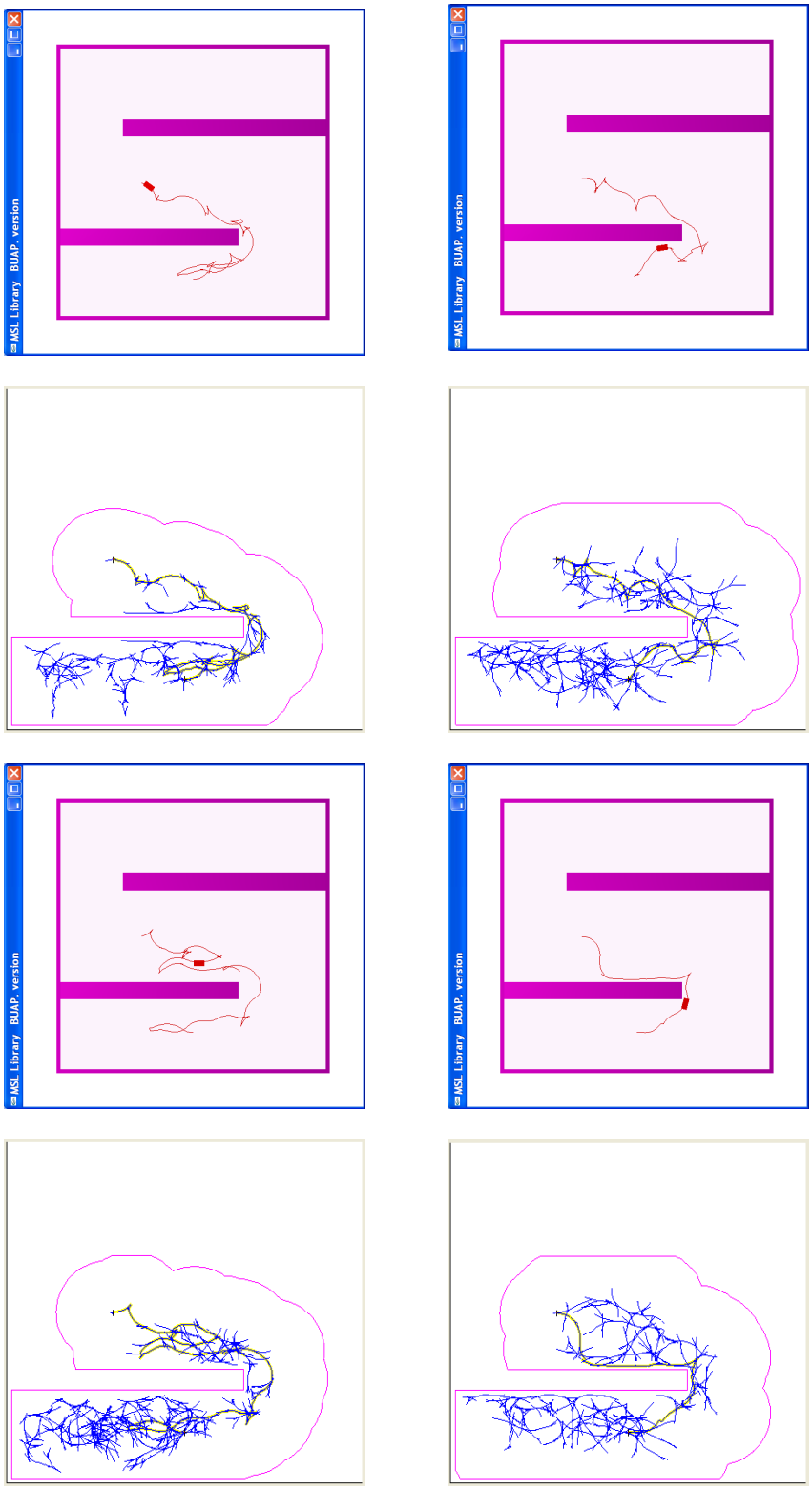


Figura 5.7: Estrategia SRRT_Local sobre el ambiente 1, variante 1 arriba y variante 2 abajo.

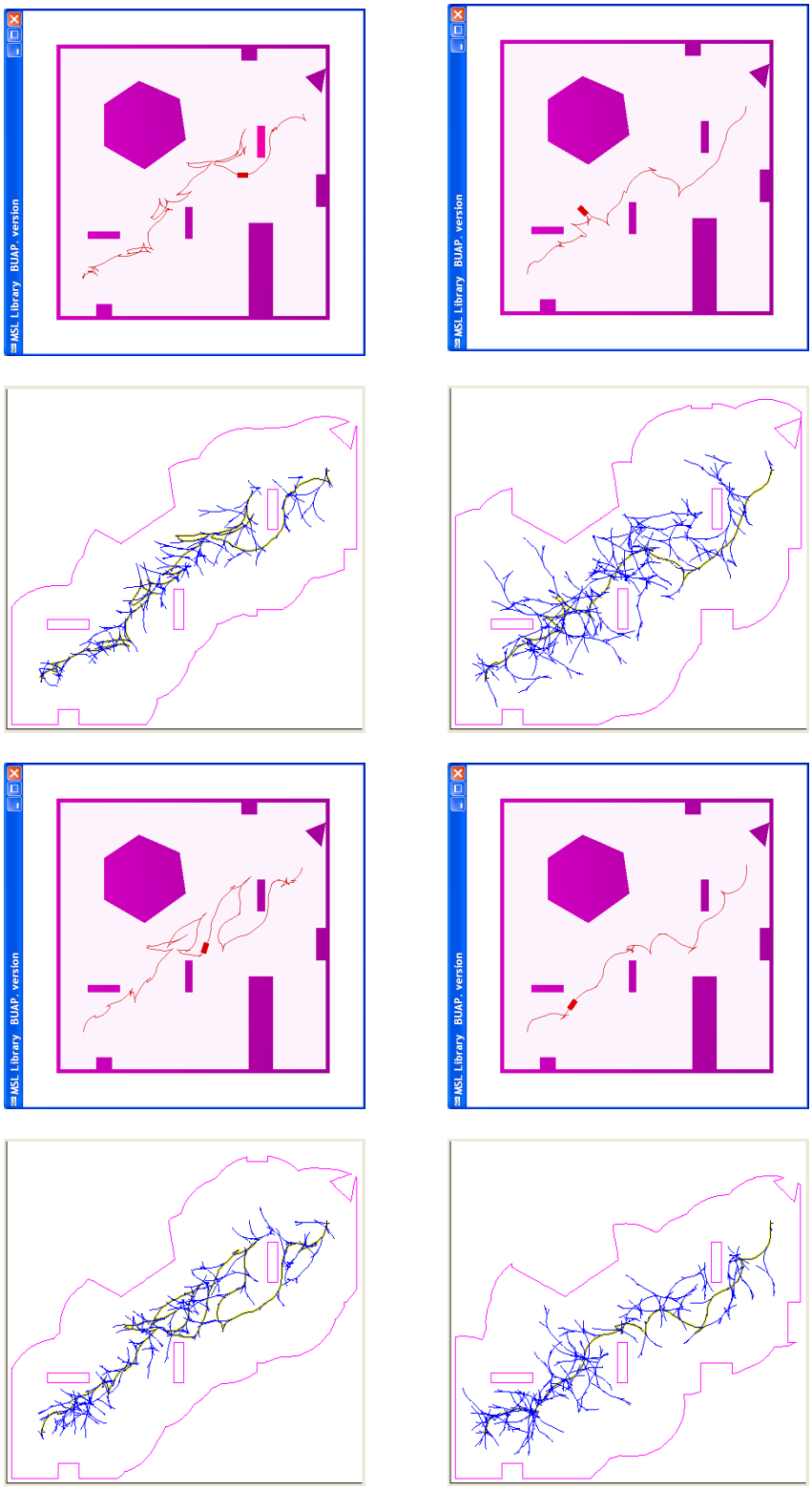


Figura 5.8: Estrategia SRRT_Local sobre el ambiente 2, variante 1 arriba y variante 2 abajo.

Los caminos planificados con la variante uno de la estrategia de exploración SRRT_Local, son factibles para los movimientos del robot móvil de tipo carro, pero son dificiles de seguir por que requieren muchas maniobras de conducción por parte del robot. Esto sucede por la unión de pequeños arboles bidireccionales y la elección aleatoria del siguiente estado para un nuevo sensado, ambas caracteristicas obligan al algoritmo a encontrar un camino factible en un espacio reducido que se traduce en muchas maniobras de conducción. La segunda variante de SRRT_Local surgió motivada por este problema, ¿comó obtener caminos que cumplan con las restricciones globales del ambiente y del robot y que además sean un poco menos complicados de seguir?.

El hecho de que la variante dos utilice al árbol aleatorio de exploración como árbol de planificación local da mayores oportunidades al planificador local de encontrar un nodo que conecte al estado candidato, sin obligarlo a construir caminos factibles en espacios reducidos, sino de aprovechar caminos ya construidos a lo largo de la región segura. En ocasiones el algoritmo ocupa un porcentaje de tiempo ligeramente mayor que la variante uno, en su búsqueda del nodo más cercano al cual conectar el nodo aleatorio producido por el planificador local. Esta sencilla modificación produce buenos resultados, como se ve en las figuras 5.7 y 5.8.

Las tablas 5.1 y 5.2 resumen la comparación de las variantes de la estrategia SRRT_Local, respecto al número de nodos y tiempos de ejcución promedios tomados de una serie de ejecuciones.

SRRT_LocalB

La ventaja que presentan las variantes de la estrategia SRRT_LocalB sobre la estrategia SRRT_Local es que no arrastran con el problema original de los planificadores RRT–bidireccionales, que necesitan conectar varios vértices para unir un árbol al otro.[4]

SRRT_Local		
Ambiente 1	Variante 1	Variante 2
Tiempo de planificación local prom.	1.83–3.42 seg.	2.90–3.83 seg.
Nodos de planificación local prom.	96–162	123–165
Tiempo de exploración prom.	43.31 seg.	50.61 seg.
Procesos de planificación local prom.	17	15

Tabla 5.1: Resultados de la estrategia SRRT_Local en el ambiente 1

SRRT_Local		
Ambiente 2	Variante 1	Variante 2
Tiempo de planificación local prom.	7.74–11.89 seg.	6.44–11.69 seg.
Nodos de planificación local prom.	129-221	146–208
Tiempo de exploración prom.	108.57 seg.	88.66 seg.
Procesos de planificación local prom.	11	11

Tabla 5.2: Resultados de SRRT_Local en el ambiente 2

En SRRT_LocalB la expansión del árbol de exploración se da al construir un nuevo árbol local a partir de un nodo ya existente en el árbol de exploración y de conectar los nodos aleatorios que produce el planificador local al árbol de exploración con raíz en el estado meta o al árbol local con raíz en el estado actual. Las figuras 5.9 y 5.10 muestran los árboles aleatorios de exploración y los caminos encontrados. Las tablas 5.3 y 5.4 resumen la comparación entre las variantes uno y dos de SRRT_LocalB, respecto al número de nodos y tiempo de ejecución.

[4] Ver capitulo 2, sección 2.3.2

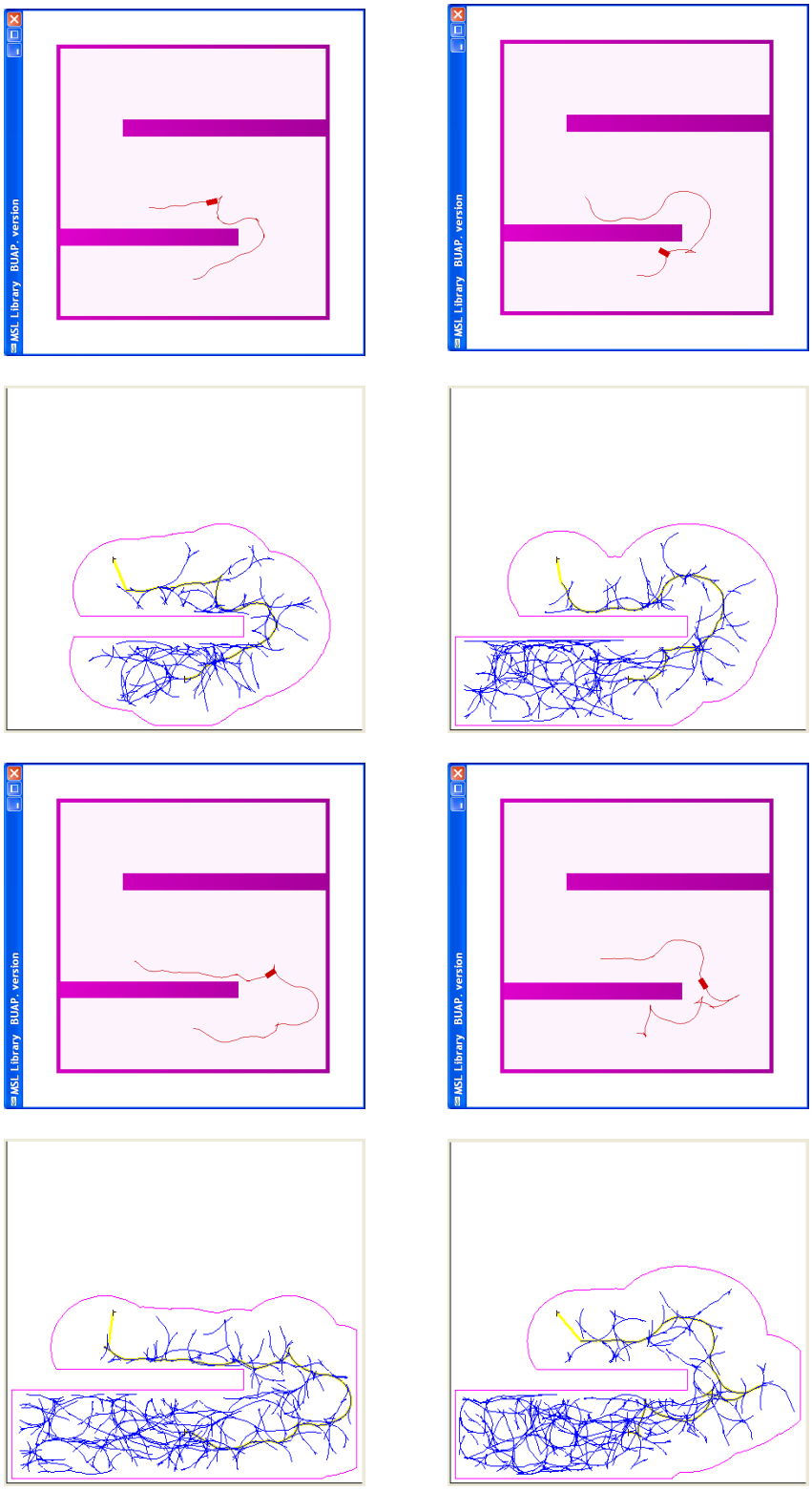


Figura 5.9: Estrategia SRRT_LocalB sobre el ambiente 1, variante 1 arriba y variante 2 abajo.

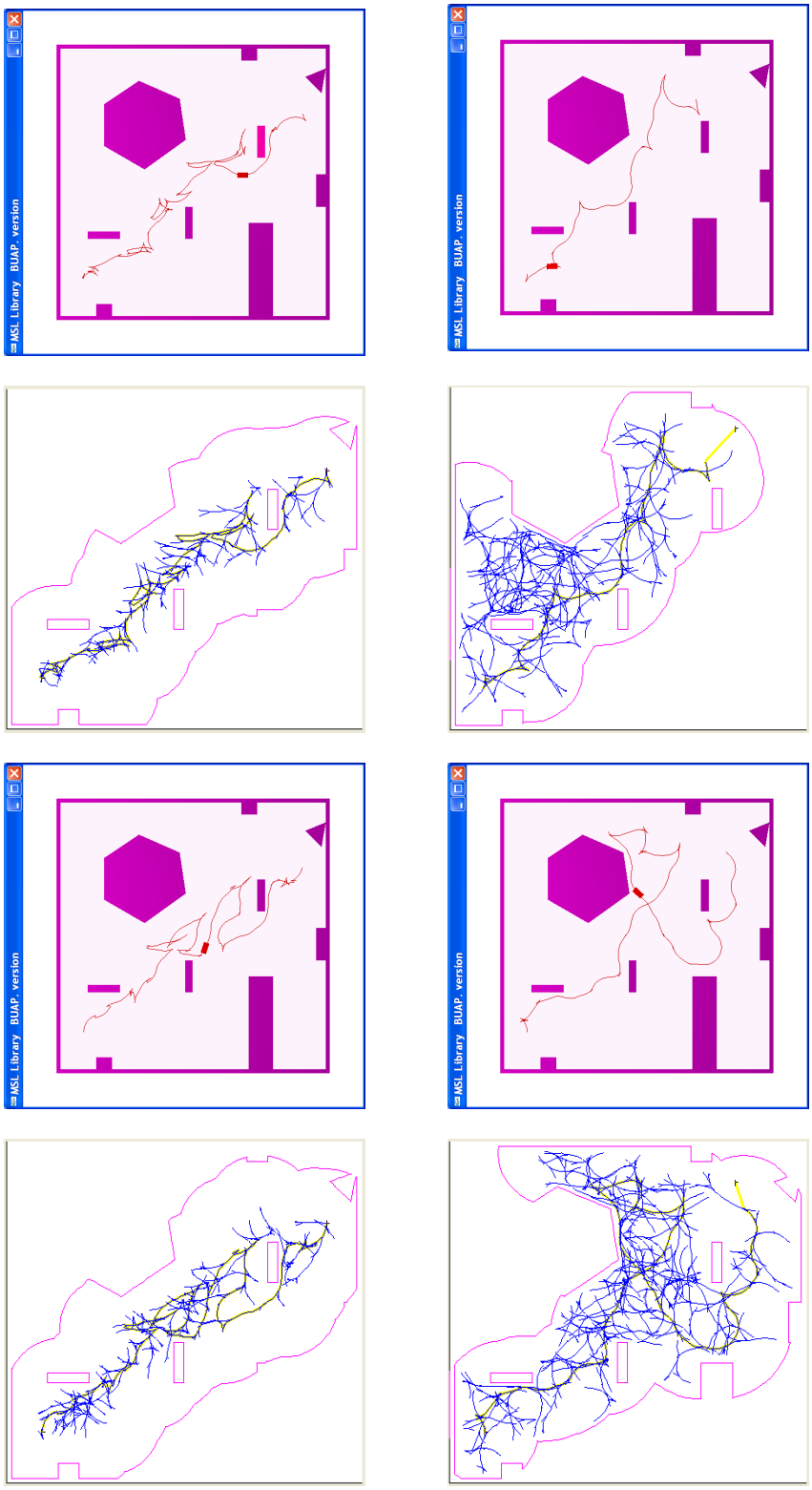


Figura 5.10: Estrategia SRRT_LocalB sobre el ambiente 2, variante 1 arriba y variante 2 abajo.

SRRT_LocalB		
Ambiente 1	Variante 1	Variante 2
Tiempo de planificación local prom.	2.29–2.75 seg.	2.21–2.8 seg.
Nodos de planificación local prom.	131–140	125–134
Tiempo de exploración prom.	54.56 seg.	44.97 seg.
Procesos de planificación local prom.	20	18

Tabla 5.3: Resultados de la estrategia SRRT_LocalB en el ambiente 1

SRRT_LocalB		
Ambiente 2	Variante 1	Variante 2
Tiempo de planificación local prom.	4.71–5.71 seg.	3.20–4.58 seg.
Nodos de planificación local prom.	168-175	127–241
Tiempo de exploración prom.	70.43 seg.	104.55 seg.
Procesos de planificación local prom.	13	24

Tabla 5.4: Resultados de SRRT_LocalB en el ambiente 2

Las estrategias de exploración conservan la habilidad de escapar de mínimos locales al realizar el proceso de retroceso hacia el estado inical, al excederse el número máximo de intentos para encontrar un nodo valido o después de evaluar todos los nodos en la lista de estados candidatos validos para el siguiente proceso de percepción del ambiente. De igual forma, los algoritmos mantienen la caracteristica de terminar con fallo después de un tiempo máximo, e implementan otra forma de conClusión derivada del proceso de retroceso. El algoritmo termina cuando el robot regresa al estado inicial, asumiendo que no es posible incrementar la información del espacio libre a través de los sensores.

Capítulo 6

Conclusiones y trabajos futuros

Presentamos la implementación, en primer lugar, de un método de exploración de ambientes desconocidos usando sensores para un robot móvil omnidireccional, el método SRT. Adaptamos al método, la estrategia de percepción SRT_Radial, para aprovechar lo más posible la información proporcionada por los sensores. En seguida aplicamos y adaptamos el método para un robot móvil que tiene una restricción no-holonómica simple en el ángulo de conducción. Después aplicamos el método a un problema híbrido de exploración de movimientos, donde se plantea contar con dos robots, uno omnidireccional o con una restricción no-holonómica simple que realiza la tarea de exploración del ambiente y un segundo robot, un robot móvil de tipo carro, que realiza la tarea de encontrar un camino factible dentro de la región segura encontrada por el robot explorador, que une la posición inicial con la posición final. Se muestran los resultados de la implementación a través de simulaciones.

En segundo lugar, desarrollamos algunas estrategias de exploración para ambientes desconocidos usando sensores para un robot móvil de tipo carro basadas en árboles aleatorios de exploración. Las estrategias de exploración SRRT_Local y SRRT_LocalB, con dos variantes cada una de ellas, adaptan los planificadores RRT para ejecutarse localmente en el área segura construida por los sensores. Las estrategias construyen

incrementalmente un árbol aleatorio de exploración como el sensor reporta el espacio libre del ambiente. El árbol refleja la conectividad del espacio libre conocido donde el robot móvil se mueve para reconocer mayores porciones del espacio físico. Para explorar el ambiente las estrategias utilizan una simple elección de estados candidatos para un nuevo proceso de sensado, dependiendo de la estrategía, guiados con una heurística sencilla sobre la distancia euclidiana que separa al estado canditato del estado meta. En el árbol los nodos son distinguidos por dos clases, nodos intermedios, aquellos donde se lleva a cabo un proceso de percepción del ambiente; y nodos terminales, nodos que sirven para darle movilidad al robot que opcionalmente tienen una descripción de la región segura local y principalmente contienen la entrada de control.

Los resultados obtenidos a través de simulaciones muestran que las estrategías resuelven el principal problema de la planificación de movimientos usando sensores, encontrar un camino factible de un estado inicial a un estado meta considerando las restricciones globales del ambiente y del robot usando la información proporcionada por los sensores en ambientes simples y complejos. La eficiencia de las estrategias varia en tiempo y en la complejidad del camino obtenido.

Trabajos futuros

Los trabajos futuros que pueden derivarse a partir de los resultados obtenidos de nuestro trabajo, principalmente son los siguientes:

- Sería ideal probar las estrategias propuestas con robots reales.
- Probar otras estrategias para la determinación del estado candidato a un nuevo proceso de percepción, por ejemplo adaptar el enfoque basado en entropía para robots manipuladores como se propuso en [30]. Actualmente, el problema de la planificación del siguiente punto de vista es un tópico abierto para investigación.

- Adaptar las estrategias a ambientes 3D.

- Debido a que el método adapta los planificadores RRT, los cuales son convenientes para manejar muchos grados de libertad y espacios de altas dimensiones, un trabajo futuro es ampliar las estrategias a otros tipos de robots, por ejemplo robots móviles manipuladores.

Por último, queda abierto el problema de necesidad y suficiencia de los algoritmos SRRT_Local y SRRT_LocalB, nuestra conjetura, que necesita ser probada, es que conservan la completitud probabilística de los planificadores RRT.

Apéndice A

Análisis sobre los RRTs

Este apéndice provee algunos análisis hechos a los RRTs, y presenta varios problemas abiertos para futuras investigaciones [14].

Un resultado clave mostrado hasta este momento es que la distribución de vértices de los RRTs converge a la distribución de muestreo, la cual usualmente es uniforme. Esto actualmente se ha mostrado para planificación holonómica en un espacio de estados no-convexo, siendo verificado a través de simulaciones y pruebas chi-cuadrada.

Sea $D_k(x)$ una variable aleatoria cuyo valor es la distancia de x al vértice más cercano en G, donde k es el número de vértices en un RRT. Sea d_k el valor de D_k. Sea ϵ la distancia incremental recorrida en el procedimiento EXTENDER (el tamaño de paso del RRT).

Consideremos el caso de un problema de planificación holonómica, en el cual $\dot{x} = u$ (el simulador incremental permite el movimiento en cualquier dirección). El primer lema establece que el RRT se acercaría arbitrariamente a cualquier punto en el espacio convexo.

Lema A.1. *Supongamos que X_{free} es un subconjunto convexo, acotado, abierto, n-dimensional de un espacio de estados n-dimensional. Para cualquier $x \in X_{free}$ y una*

constante positiva $\epsilon > 0$, $\lim_{k\to\infty} P\left[d_k\left(x\right) < \epsilon\right] = 1$.

Bosquejo de la prueba: Sea x cualquier estado en X_{free}, y sea x_0 un vértice inicial. Sea $Q(x)$ un "balón"de radio ϵ, centrado en x. Sea $Q'(x) = Q(x) \cap X_{free}$. Note que $\mu(Q'(x)) > 0$, en donde μ denota el volumen (o medida) de un conjunto. Inicialmente, $d_1(x) = \rho(x, x_0)$. En cada iteración, la probabilidad que un estado elegido aleatoriamente se situará en $Q'(x)$ es estrictamente positivo. Por lo tanto, si todos los vértices en el RRT se sitúan en el exterior de $Q(x)$, entonces $E\left[D_k\right] - E\left[D_{k+1}\right] > b$ para algún número real positivo $b > 0$. Esto implica que $\lim_{k\to\infty} P\left[d_k\left(x\right) < \epsilon\right] = 1$. □

El siguiente lema extiende el resultado de espacios convexos a espacios no-convexos.

Lema A.2. *Supongamos* X_{free} *un componente conectado, no-convexo, acotado, abierto, n-dimensional de un espacio de estados n-dimensional. Para cualquier* $x \in X_{free}$ *y un número real positivo* $\epsilon > 0$, *entonces* $\lim_{n\to\infty} P\left[d_n\left(x\right) < \epsilon\right] = 1$.

Bosquejo de la prueba: Sea x_0 el vértice inicial en un RRT. Sea $Q(x)$ un área en forma circular de radio ϵ con centro en x. Si x_0 y x están en el mismo componente conectado de un conjunto abierto acotado, entonces existe una secuencia de estados, $x_1, x_2, \ldots, x_k$, tal que, puede construirse una secuencia de $Q = Q_1(x_1), \ldots, Q_k(x_k)$, con $Q_i \cap Q_{i+1} \neq 0$ para cada $i \in \{1, \ldots, n-1\}$, $x_0 \in Q_1$ y $x \in Q_k$. Sea $C_i = Q_i \cap Q_{i+1}$. Note que Q se construye de forma que cada C_i es abierto, lo cual implica que $\mu(C_i) > 0$. El lema A.1 se puede aplicar inductivamente para cada C_i y concluir que $\lim_{n\to\infty} P\left[d_n\left(x_i\right) < \epsilon\right] = 1$ para un $x_i \in C_i$. En cada caso, ϵ puede seleccionarse para garantizar que hay un vértice del RRT en C_i. Eventualmente, la probabilidad de que un vértice del RRT caerá en Q_k se aproxima a uno. Una aplicación final del lema A.1. implica que $P\left[d_n\left(x\right) < \epsilon\right] = 1$. □

Para la planificación holonómica de caminos esto inmediatamente implica lo siguiente:

Teorema A.3. *Supongamos que x_{init} y x_{goal} se sitúan dentro del mismo componente conectado de un componente conectado no-convexo, acotado, abierto, n-dimensional de un espacio de estados n-dimensional. La probabilidad de que un RRT construido a partir de x_{init} encontrará un camino a x_{goal} se acerca a uno como el número de vértices del RRT se aproxima al infinito.*

Esto establece completez probabilística, como se considera en [8], del RRT básico.

Sea X un vector de valores para una variable aleatoria que representa el proceso de muestreo usado para construir un RRT. Esto refleja la distribución de muestreos que es regresada por la función ESTADO_ALEATORIO en el algoritmo EXTENDER. Usualmente , X es caracterizado por una función de densidad de probabilidad uniforme sobre X_{free}; sin embargo, permitiremos a X ser caracterizado por cualquier función de densidad de probabilidad continua. Sea X_k un vector de valores para una variable aleatoria que representa la distribución de los vértices del RRT después de k iteraciones.

Teorema A.4. *X_k converge a X en probabilidad.*

Bosquejo de la prueba: Consideremos el conjunto $Y_k = \{x \in X_{free} \mid \rho(x, v) > \epsilon \ \forall v \in V_k\}$ en donde V_k es el conjunto de todos los vértices del RRT después de la iteración k. Intuitivamente, esto representa la porción descubierta de X_{free}. Del lema A.2, se sigue que $Y_{k+1} \subseteq Y_k$ y $\mu(Y_k)$ se acerca a cero como k se aproxima al infinito. Recuerdemos que el algoritmo de construcción del RRT agrega un vértice a V si el muestreo se encuentra en ϵ de otro vértice en V (ϵ es el tamaño de paso del RRT). Cada vez que esto ocurre, el nuevo vértice sigue la misma densidad de probabilidad como X. Debido a que $\mu(Y_k)$

se aproxima a cero, la función de densidad de probabilidad de X y X_k difieren sólo en algún conjunto $Z_k \subseteq Y_k$. Desde que $\mu(Y_k)$ se acerca a cero como k se aproxima a infinito, $\mu(Z_k)$ también se aproxima a cero. Desde que $\mu(Z_k)$ se aproxima a cero y la función de densidad de probabilidad de X es suave (smooth), X_k converge a X en probabilidad. □

Ahora consideramos el caso más general. Supongamos que los movimientos obtenidos del simulador incremental son localmente restringidos. Por ejemplo, podrían originarse integrando $\dot{x} = f(x, u)$ sobre algún tiempo Δt. Supongamos además que el número de entradas al simulador incremental es finito, Δt es constante, ninguno de dos vértices se sitúa en un $\epsilon > 0$ especificado uno del otro de acuerdo a ρ y que EXTENDER elige la entrada aleatoriamente. Eventualmente, es posible eliminar algunas de estas restricciones. Si x_{init} y x_{goal} se encontran en el mismo componente conectado de un componente conectado no-convexo, acotado, abierto, n-dimensional de un espacio de estados n-dimensional. Además, existe una secuencia de entradas, $u_1, u_2, \ldots, u_k$, cuando es aplicado a x_{init} produce una secuencia de estados, $x_{init} = x_0, x_1, x_2, \ldots, x_k+1 = x_{goal}$. Todos estos estados se encuentran en el mismo componente conectado de X_{free}.

Lo siguiente establece la completez probabilística de un planificador no-holonómico.

Teorema A.5. *La probabilidad de que un RRT inicializado en x_{init} contenga a x_{goal} como un vértice se aproxima a uno como el número de vértices se aproxima al infinito.*

Visión de la prueba: El razonamiento procede por inducción en i. Supongamos, que el RRT contiene a x_i como un vértice después de un número de iteraciones finitas. Consideremos el diagrama de Voronoi asociado con los vértices del RRT. Existe un número real positivo, c_1 tal que $\mu(Vor(x_i)) > c_1$ en donde $Vor(x_i)$ denota la región de Voronoi asociada con x_i. Si un muestro aleatorio cae en $Vor(x_i)$, el vértice

será seleccionado para expansión, y se aplica una entrada aleatoria; entonces, x_i tiene probabilidad $\mu\left(Vor\left(x_i\right)\right)/\mu\left(X_{free}\right)$ de ser seleccionado. Existe un segundo número positivo real, c_2 (el cual depende de c_1), tal que la probabilidad de que la correcta entrada, u_i, es seleccionada en al menos c_2. Si ambos x_i y u_i tienen probabilidad de al menos c_2 de ser seleccionados en cada iteración, entonces la probabilidad de que el siguiente paso en la trayectoria solución será construido tiende a uno. Este razonamiento es aplicado inductivamente de x_1 a x_2, hasta que el estado final $x_{goal} = x_{k+1}$ es alcanzado. □

Razón de convergencia. El teorema 6 y 7 expresan la razón de convergencia del planificador. Estos resultados representan un importante primer paso hacia un completo entendimiento del comportamiento de los algoritmos de planificación que usan RRT; sin embargo la razón de convergencia desafortunadamente es expresada en términos de parámetros que no pueden medirse fácilmente. Permanece abierto el problema de caracterizar la razón de convergencia en términos de parámetros simples que se puedan calcular para un problema en particular. Esta dificultad general existe incluso en el análisis de planificadores aleatorios para el problema de la planificación holonómica de caminos.

Por simplicidad, asuminos que el planificador consiste de un RRT simple. El planificador bidireccional es solo ligeramente mejor en términos del análisis, y un RRT simple es más fácil de analizar. Además, asuminos que el tamaño de paso es lo suficientemente largo para que el planificador intente conectar x_{near} a x_{rand}.

Sea $A = \{A_0, A_1, \ldots, A_k\}$ una secuencia de subconjuntos de X, referidos como una *secuencia de atracción*. Los conjuntos restantes deben ser elegidos bajo las siguientes reglas. Para cada A_i en A, existe un "basin" $B_i \subseteq X$ tal que:

1. Para todo $x \in A_{i-1}$, $y \in A_i$ y $z \in X \setminus Bi$, la métrica, ρ, permite $\rho(x, y) < \rho(x, z)$.

2. Para todo $x \in B_i$, existe un m tal que la secuencia de entradas $\{u_1, u_2, \ldots, u_m\}$ selecionadas por el algoritmo EXTENDER llevarán al estado a $A_i \subseteq B_i$.

Finalmente, se asume que $A_k = X_{goal}$.

Cada "basin" B_i puede intuitivamente considerarse como una zona protegida que asegura que un elemento de B_i será seleccionado por la búsqueda del vecino más cercano, y una fuente de potencial que atrae al estado a A_i. Una secuencia de atracción sería elegida con cada A_i tan larga como sea posible y con k tan pequeño como sea permisible. Si el espacio contiene pasajes estrechos, entonces la secuencia de atracción será más larga y cada A_i será más pequeño. El análisis indica que el rendimiento degradará significativamente en este caso, lo cual es consistente con los resultados obtenidos del análisis para planificadores holonómicos [7]. Note que para la planificación kinodynamic, la elección de la métrica, ρ, también afecta importantemente a la secuencia de atracción, y finalmente el rendimiento del algoritmo.

Usando μ para representar una unidad de medida, p se define como

$$p = \min_i \{\mu(A_i) / \mu(X_{free})\},$$

lo cual corresponde a una cota inferior en la probabilidad de que un estado aleatorio se situará en un A_i particular.

El siguiente teorema caracteriza el número esperado de iteraciones requeridas para encontrar una solución.

Teorema A.6. *Si existe una secuencia de conexión de longitud k, entonces el número esperado de iteraciones requeridas para conectar q_{init} a q_{goal} no es mayor que k/p.*

Prueba: Si un vértice del RRT se encuentra en A_{i-1} y un muestreo aleatorio, x, cae en A_i, entonces el RRT será conectado a x. Esto es cierto porque usando la primera

propiedad en la definición de un basin, se sigue que uno de los vértices en B_i debe ser seleccionado por expansión. Usando la segunda propiedad del basin, se eligirán entradas que finalmente generarán un vértice en A_i.

En cada iteración, la probabilidad de que el muestreo aleatorio se sitúe en A_i es por lo menos p; por lo tanto, si A_{i-1} contiene un vértice del RRT, entonces A_i contendrá un vértice con probalididad de al menos p. En el peor caso, las iteraciones pueden considerarse como pruebas Bernoulli en las cuales p es la probabilidad de un resultado exitoso. Un problema de planificación de caminos se resuelve después de que k resultados exitosos son obtenidos debido a que cada éxito extiende el progreso del RRT de A_{i-1} a A_i.

Sean $C_1, C_2, \ldots, C_n$ variables aleatorias independientes e idénticamente distribuidas según la distribución de Bernoulli con parámetro p. La variable aleatoria $C = C_1 + C_2 + \cdots + C_n$ denota el número de éxitos después de n iteraciones. Desde que cada C_i tiene una distribución de Bernoulli, C tendrá una distribución binomial,

$$\left(\begin{array}{c} n \\ k \end{array} \right) h^k \left(1-h\right)^{n-k},$$

donde k es el número de éxitos. La esperanza de la distribución binomial es n/p, la cual también representa una cota superior en la probabilidad esperada de encontrar exitosamente un camino. □

El siguiente teorema establece que la probabilidad de éxitos aumenta exponencialmente con el número de iteraciones.

Teorema A.7. *Si existe una secuencia de atracción de longitud k, para una constante $\delta \in (0, 1]$, la probabilidad que el RRT encuentre un camino después de n iteraciones es al menos $1 - \exp\left(-np\delta^2/2\right)$, donde $\delta = 1 - k/\left(np\right)$.*

Prueba: La variable aleatoria C de la prueba del teorema 6 tiene una distribución binomial, lo cual habilita la aplicación de un límite tipo Chernoff en su cola de probabilidad. El siguiente teorema[1] es directamente aplicable para establecer la prueba. Si C es distribuido binomialmente, $\delta \in (0, 1]$, y $\mu = E[C]$, entonces $P[C \leq (1 - \delta)\mu] < \exp(\mu\delta^2/2)$. $\square$

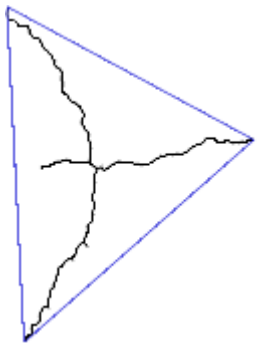


Figura A.1: Envolvente convexa de un RRT en un disco "infinitamente"grande.

Un RRT en un gran disco. En el límite cuando el número de iteraciones se aproxima al infinito, el RRT llega ser uniformemente distribuido, pero qué pasa cuando el RRT es colocado en un espacio grande?. Intuitivamente, parece que la mejor estrategia sería crecer el árbol lejos del estado inicial tan rápido como sea posible. Para determinar si esto ocurre, se desarrollaron muchos experimentos de simulación [14] (cada uno con cientos de miles de iteraciones) para carecterizar como un RRT crece en el caso límite en un disco con un radio que se aproxime al infinito. Consideremos el caso de un espacio de estados 2D y la planificación holonómica. La figura A.1 muestra un resultado típico, en el cual el RRT tiene tres grandes ramas separadas una de la otra aproximadamente 120 grados. Este comportamiento se observa repetidamente para el caso 2D, aunque la orientación de las ramas es aleatoria. En altas dimensiones se ha observado que el

[1] Teorema referido en el articulo de R. Motwani y P. Raghavan, *Randomized Algorithms*.

RRT hace $n + 1$ ramas en un espacio n-dimensional. Las ramas también tienen igual separación unas de otras. Esto da evidencia experimental en el sentido esperado, el RRT crece hacia fuera del origen en una razón líneal al número de iteraciones y decrementa moderadamente con el número de dimensiones. Permanece abierta la pregunta para confirmar estas observaciones probando el número y direcciones de estas ramas.

Apéndice B

Librería MSL

La librería Motion Strategy Library, MSL[1] , desarrollada por Steven LaValle y su grupo de colaboradores, permite el 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).

B.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 B.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 siguiente estado del sistema,

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

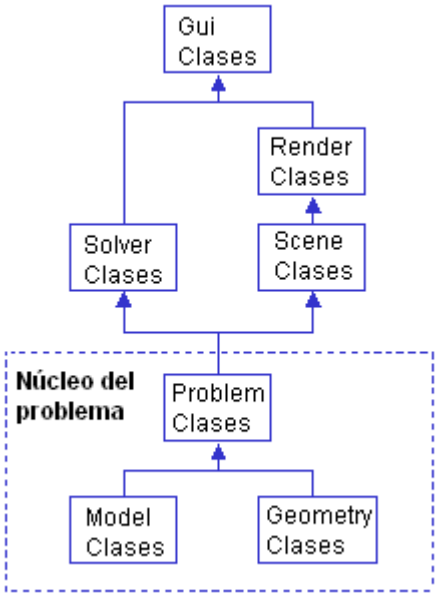


Figura B.1: Clases que constituyen la librería MSL.

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.

B.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.

B.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.

B.4. Jerarquía de clases

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

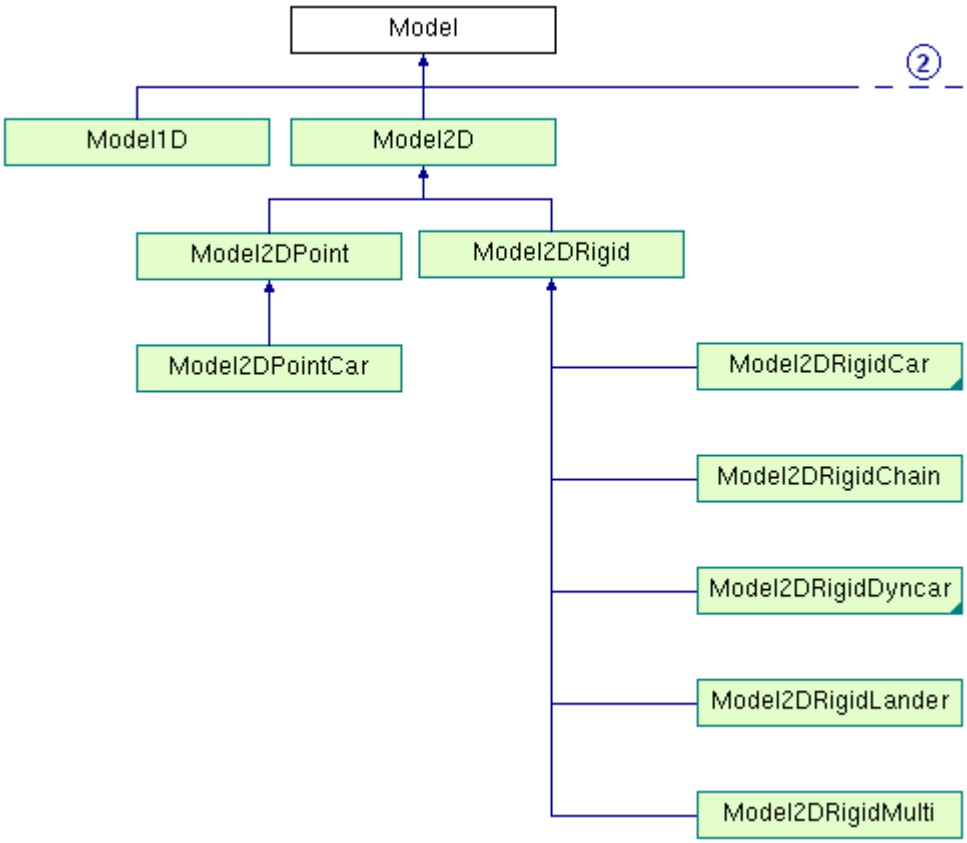


Figura B.2: Jerarquía clase Model

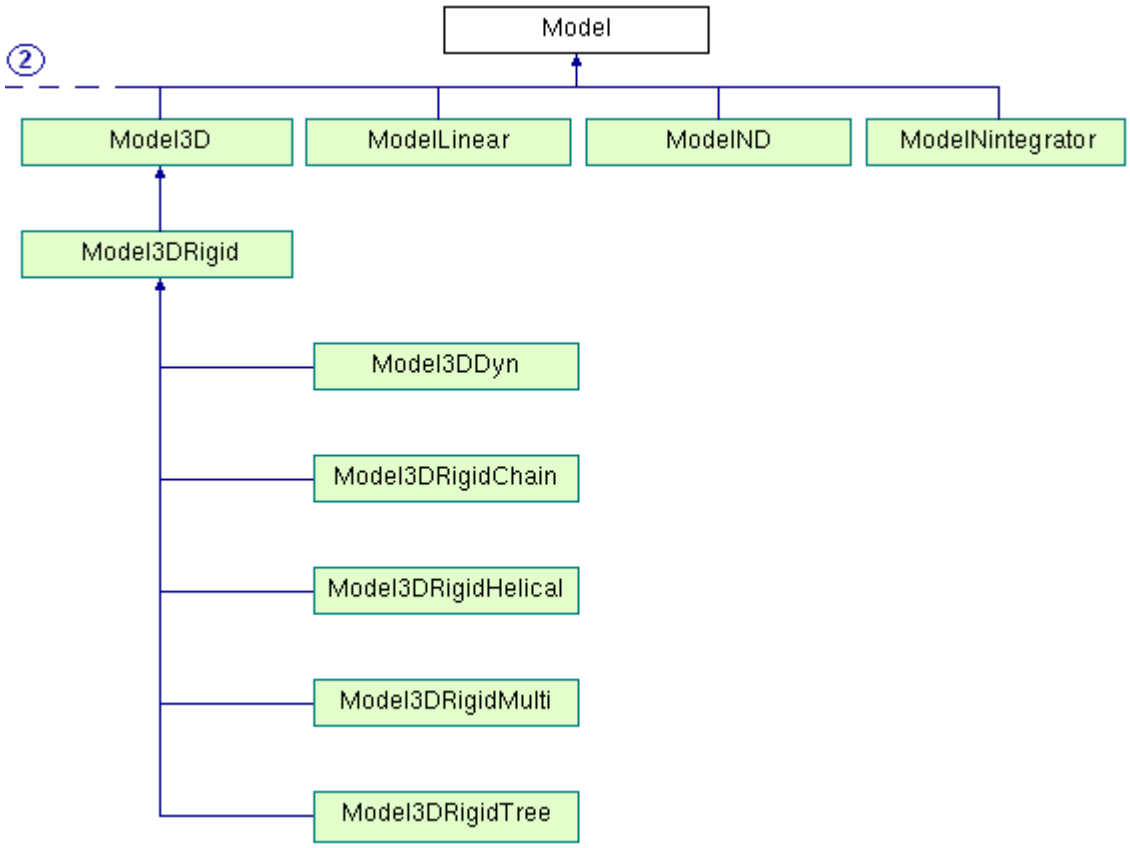


Figura B.3: Jerarquía clase Model

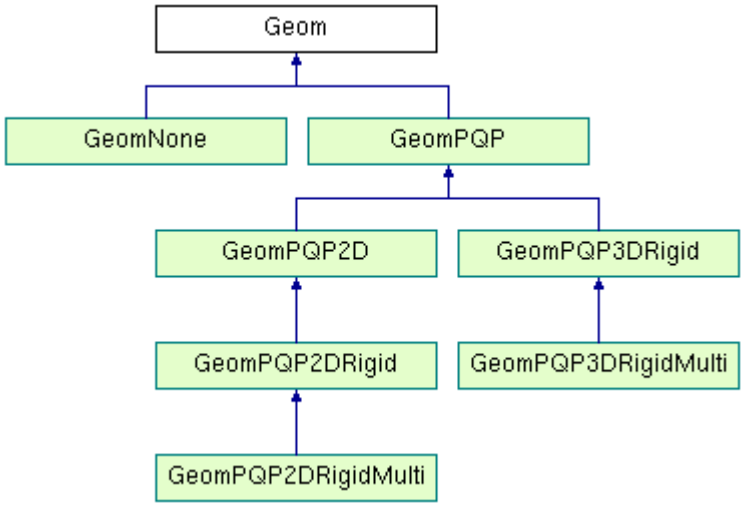


Figura B.4: Jerarquía clase Geom

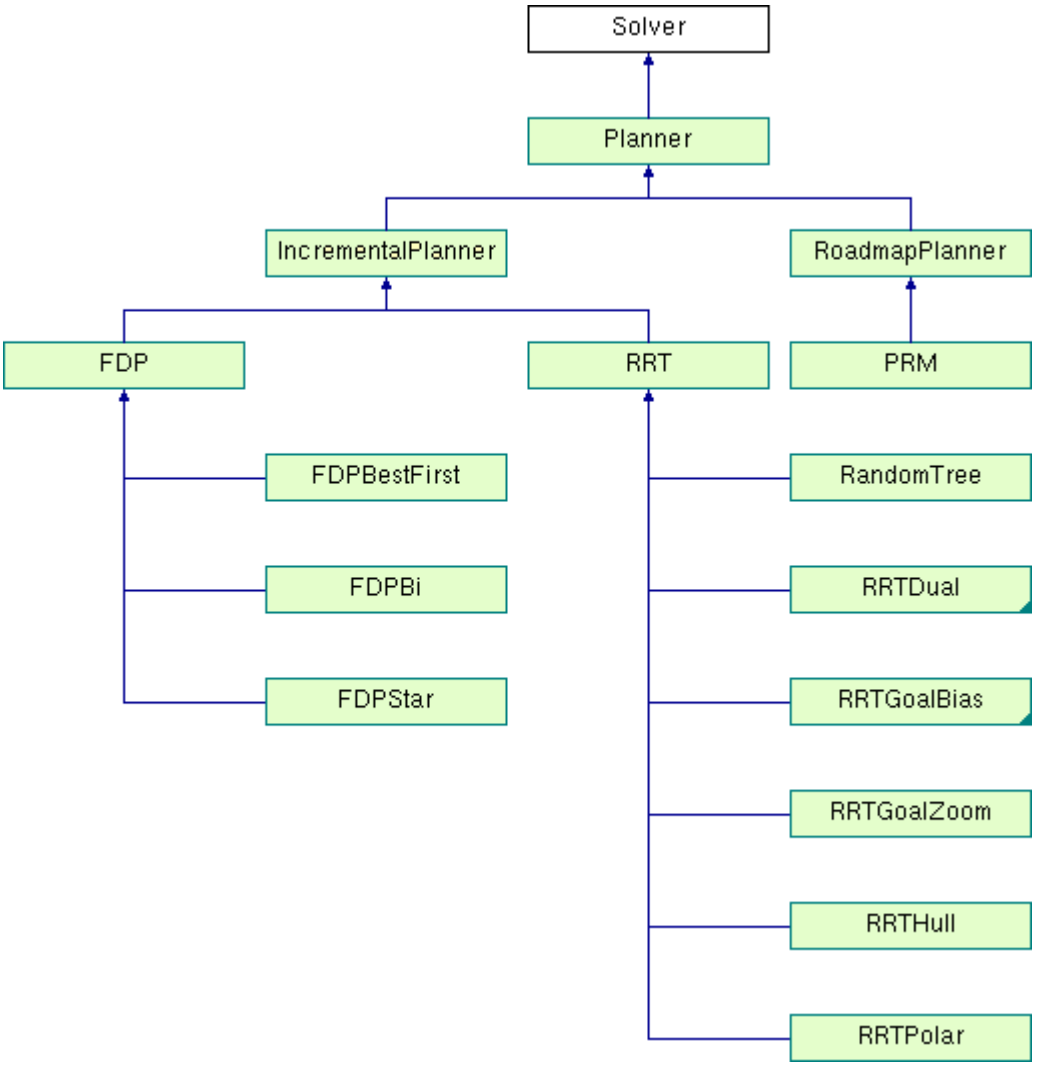


Figura B.5: Jerarquía clase Solver

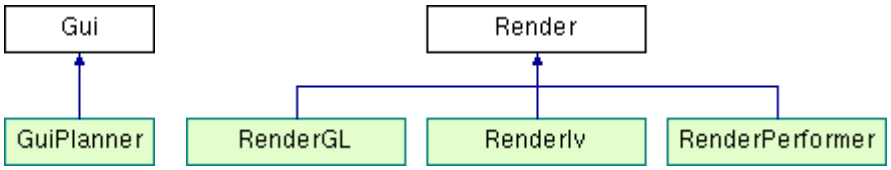


Figura B.6: Jerarquía clase Gui y Render.

Apéndice C

Librería 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 [29]. 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).

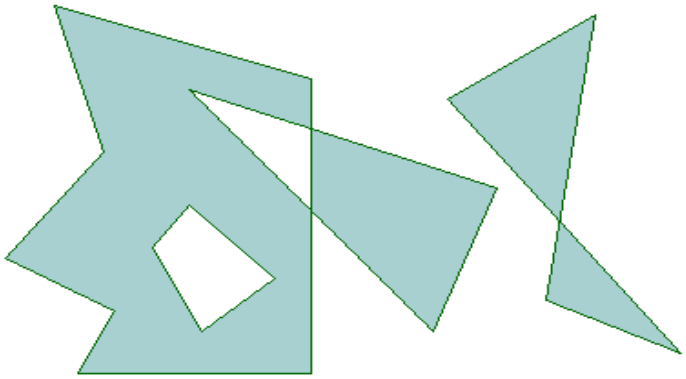
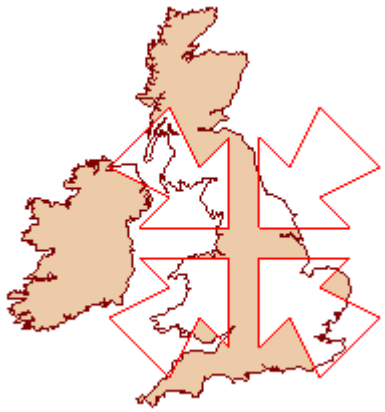


Figura C.1: Polígono genérico con cuatro contornos.

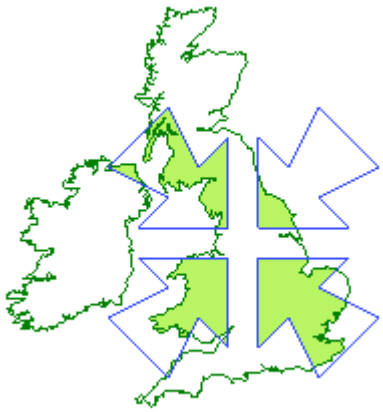
C.1. Descripción

Un polígono genérico (o un 'conjunto poligono') consiste de cero o más limites disjuntos de una composición arbitraria. Cada limite se le denomina "contorno", y puede ser como ya se menciono, convexo, concavo o interceptarse a sí mismo. Los huecos internos pueden formarse debido a los contornos. Ver la figura C.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.

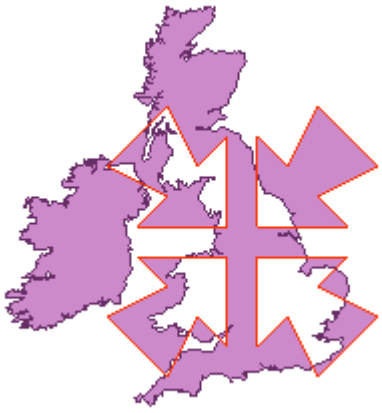
La librería soporta cuatro tipos de operaciones de recorte: la diferencia, intercepción, or-exclusivo o unión de dos polígonos genéricos. La figura C.2 muestra los tipos de operación, en cada caso el polígono resultante es resaltado con un color de relleno.



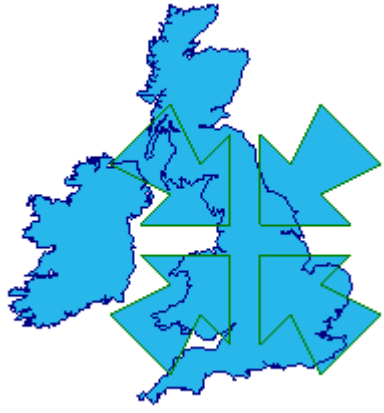
a)Diferencia



b)Intercepción



c)Or-exclusivo



d)Unión

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

C.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);
```

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 C.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

```
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
```

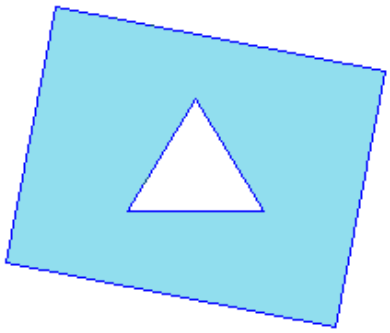


Figura C.3: Polígono y su correspondiente archivo ascii

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);
```

C.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 C.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.

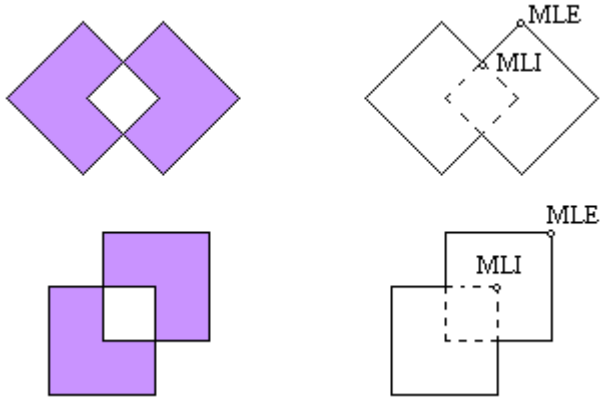
Los ejemplos en la figura C.4.b), muestran figuras similares que se intersectan a sí mismas, sin embargo, cada una crea dos contornos externos tocandose en el punto de intersección. En resumen, los contornos generados por el recortador no sólo son afectados por la forma de los polígonos entrantes sino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

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

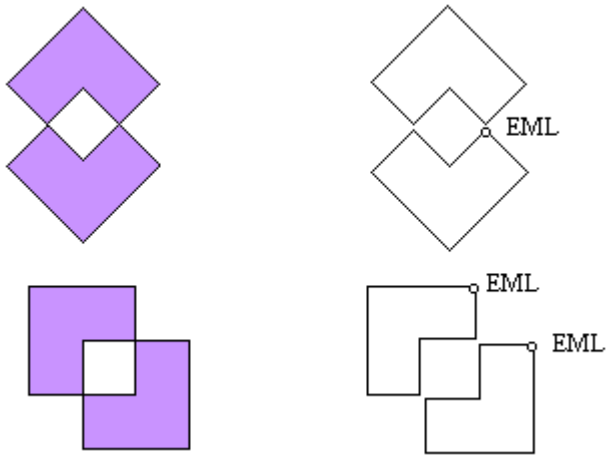
La actual versión simplemente utiliza las banderas para identificar cuales contornos pueden ser considerados huecos y cuales externos. Descubriendo cuales huecos se sitúan en que contornos externos toma un poco más de trabajo por parte del usuario, Una forma de asociar los huecos $H_1, H_2, \ldots, H_n$ 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.

C.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)



b)

Figura C.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.

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] 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.

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

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

[6] 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*, Vol. 79, pp.497–516, 1957.

[7] 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*, pp. 141–154, 1998.

[8] 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*, Vol. 12, No. 4, pp. 566–580, 1996.

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

[10] 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*, pp. 1067–1072, 1996.

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

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

[13] S. M. LaValle and Z. S. Peng Cheng. RRT-based trajectory desing for autonomous automobiles and spacecraft. *Archives of control sciences*, Vol. 11, No. 3, pp. 51–78, 2001.

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

[15] 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*, pp. 995–1001, 2000.

[16] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Algorithmic and Computational Robotics: New Directions*, Vol. 20, No. 5, pp. 378–400, 2001.

[17] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, Vol. 32, No. 2, pp. 108–120, 1983.

[18] V. Lumelsky and A. Stepanov. Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, Vol. 3, No. 4, pp. 403–430, 1987.

[19] E. Mazer, J. M Ahuactzin, and P. Bessiére. The ariadne's clew algorithm. *Journal of Artificial Intelligence Research*, No. 9, pp. 295–316, 1998.

[20] 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.

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

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

[23] J. A. Reeds and L. A. Shepp. Optimal pahts for car that goes both forwards and backwards. *Pacific J. Math*, Vol. 145, No. 2, pp. 367–393, 1990.

[24] 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*, No. 24, pp. 207–234, 1999.

[25] A. Sanchez and R. Zapata. Sensor-based probabilistic roadmaps for car-like robots. *Fifth Mexican International Conference in Computer Science, IEEE Computer Society*, pp. 282-288, 2004.

[26] S. Sekhavat, P. Svestka, J. P. Laumond, and M. H. Overmars. Multilevel path planning for nonholonomic robots using semiholonomic subsystems. *Int. J. Robot. Res.*, No. 17, pp. 840–857, 1998.

[27] 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 Robotics and Automation*, 1995.

[28] D. Vallejo, C. Jones, and N. Amato. An adaptive framework "for single shot"motion planning. *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1722–1727, 2000.

[29] B. R. Vatti. A generic solution to polygon clipping. *Communications of the ACM*, Vol. 3, No. 7, pp. 56–63, 1992.

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

[31] 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.