Benemérita

Universidad Autónoma de Puebla.

Facultad de Ciencias de la Computación.

“SRT-BUAP: Una herramienta

para la exploración y navegación de robots móviles”.

Tesis profesional.

Para obtener el titulo de:

Ingeniería en Ciencias de la Computación.

Presenta:

Yazmin Rojas García.

Asesor:

Dr. Abraham Sánchez López.

Puebla, Pue. Primavera 2009.

Tesis de ingeniería sustentada por:

Yazmin Rojas García.

Como requisito parcial para obtener el titulo de

Ingeniería en Ciencias de la Computación.

Aceptada por:

_____

Dr. Abraham Sánchez López.

Asesor.

MC. Yalú Galicia Hernández	MC. José Esteban Torres León.
Presidente	Secretario

Primavera 2009

Agradecimientos.

Primero que nada quiero agradecer a Dios por haberme brindado esta hermosa experiencia de obtener un titulo universitario.

También agradezco a mi asesor de tesis el Dr. Abraham Sánchez López por su ayuda y enseñanza al realizar este proyecto de tesis poniéndome retos y haciéndome ver que no todo es imposible.

Agradezco a todos mis profesores por apoyarme, sacarme de mis dudas y abrirme más caminos.

Tengo un agradecimiento especial a mis padres Ma. Cristina A. García Jiménez y Humberto Rojas Castro por que siempre me han apoyado escuchándome y dándome ánimos para lograr esta meta.

Gracias a mi hermano Tonalli Rojas García por tenerme paciencia y aunque no es tu área siempre alguna duda me puedes aclarar.

Gracias a la Dra. Ma. Alberta García Jiménez por orientarme que carrera tomar y siempre apoyándome en lo que podía.

Mis tíos y primos (Alejandro, Ángeles, Marcela, Berenice, Marbella, Miguel Ángel, Alejandra, por mencionar a algunos) también les agradezco mucho por su apoyo tanto moral como económico.

Y finalmente le quiero agradecer a mis amigos (Rogelio, Andrés, Gerardo, Daniel, etc.) por estar a lo largo de mi carrera al pendiente de mi apoyándonos mutuamente y no dejarnos vencer por tal difícil que a veces parecía la situación.

Soy en verdad un viajero solitario

y los ideales que han iluminado

mi camino y han proporcionado

una y otra vez nuevo valor para

afrontar la vida han sido:

la belleza, la bondad y la verdad.

Albert Einstein.

Índice.

Capítulo I. 8

1. Introducción. 8

1.1 Antecedentes del proyecto. 8

1.2 Objetivo General. 9

1.3 Objetivos Específicos. 9

1.4. Bosquejo. 9

Capítulo II. 11

2.Árboles aleatorios de exploración rápida. 11

2.1 Formulación del problema. 11

2.2 Árboles aleatorios de exploración rápida. 12

2.3 Diseño de planificadores de caminos. 13

2.4 Aplicación en robots tipo carro. 17

Capítulo III. 19

3. Métodos SRT para ambientes desconocidos. 19

3.1 Método SRT. 19

3.2 Exploración de ambientes desconocidos con el método SRT. 20

3.3 Exploración con SRT-Radial. 23

Capítulo IV. 25

4. Exploración usando sensores con AAL. 25

4.1 Estructura de datos. 25

4.2 Proceso de exploración. 26

4.3 Algoritmo. 27

4.4 Estrategia SRRT_Local. 28

4.5 Estrategia SRRT_LocalB. 30

Capítulo V. 33

5. Resultados y experimentos. 33

5.1 Comportamiento SRT_Extensive con sensado radial. 33

5.2 Comportamiento SRT_Extensive con sensado star. 34

5.3 Comportamiento SRT_Goal con sensado radial. 36

5.4 Comportamiento SRT_Goal con sensado star. 37

5.5 Comportamiento SRRT_local_v1 con sensado radial. 38

5.6 Comportamiento SRRT_local_v1 con sensado star. 39

5.7 Comportamiento SRRT_local_v2 con sensado radial. 40

5.8 Comportamiento SRRT_local_v2 con sensado star. 41

5.9 Comportamiento SRRT_localB_v1 con sensado radial. 42

5.10 Comportamiento SRRT_localB_v1 con sensado star. 43

5. 11Comportamiento SRRT_localB_v2 con sensado radial. 44

5.12 Comportamiento SRRT_localB_v2 con sensado star. 46

Capitulo VI. 47

Conclusiones. 47

Apéndice A. 48

Librería MFC. 48

A.1. Descripción general. 48

A.2 Jerarquía de MFC. 49

Apéndice B. 50

Librería MSL. 50

B.1 Descripción general. 50

B.2. Librerías utilizadas por MSL. 52

B.3. Crear un nuevo problema. 52

B.4. Jerarquía de clases. 54

Apéndice C. 57

Librería GPC. 57

C.1 Descripción. 57

C.2 Funciones. 59

C.3 Huecos y contornos externos. 61

C.4 Asociación de los huecos con el contorno externo. 62

C.5 Lados coincidentes y casi-coincidentes. 62

Bibliografía. 64

Capítulo I

1. Introducción.

Los robots móviles o vehículos auto guiados, son estructuras móviles que pueden desplazarse en ambientes estructurados o no estructurados ya sea parcialmente o totalmente conocidos.

Gracias a estos robots móviles que hacen su contribución en la industria realizando diversas actividades en ambientes nocivos, o utilizar elementos peligrosos que son nocivos a la salud e integridad física. La inclinación actual se dirige a independizar al máximo al robot móvil del operador, cumpliendo sólo con la función de vigilar que su comportamiento sea normal y dando un mínimo de instrucciones, lo que significa que el robot móvil debe de ser capaz de identificar, por si solo, el entorno en el que va a laborar, es decir, debe de ser autónomo (del griego auto "uno mismo" y nomos "norma", en términos generales es la capacidad de tomar decisiones sin ayuda de otro elemento).

1.1 Antecedentes del proyecto.

Desde siempre el hombre ha tenido el deseo de poseer el conocimiento del propio ser, de su comportamiento y el de otras especies, a fin de crear agentes con capacidad autónoma con los cuales pueda compartir la inteligencia, para encargarle la realización de tareas que a él le desagradan por ser monótonas, complicadas o peligrosas.

Para el estudio y modelado de estos comportamientos a fin de reproducirlos en un robot, se ha hecho necesaria la cooperación de diversas ramas de la ciencia. Con el avance de la ciencia, el desarrollo de las nuevas tecnologías y de las diversas disciplinas se ha podido construir robots que nos facilitan algunas tareas.

Los robots móviles tienen un gran rango de aplicaciones, que con frecuencia son de naturaleza no industrial.

Los robots móviles se utilizan en situaciones peligrosas o ambientes hostiles como: debajo del agua, en zonas contaminadas radioactivamente, biológicamente o en el espacio.

La característica principal de los robots móviles es la movilidad y la independencia de un sistema físico de guía, como puede ser rieles, cables, medios magnéticos, ópticos, etc. Por lo que los robots móviles requieren de un sistema de navegación que les permita en todo momento conocer su posición dentro del ambiente, con la finalidad de poder definir un plan de trayectoria a seguir para poder llegar a su objetivo evitando obstáculos que se les puedan presentar en el camino, para esto, el robot debe poseer un sistema sensorial que le permita percibir y representar el mundo que lo rodea.

El laboratorio MOVIS de la FCC-BUAP ha propuesto y desarrollado en los últimos años algoritmos y estrategias para la navegación segura de robots móviles. La estrategias están orientadas a la exploración de robots móviles en ambientes desconocidos, dichas estrategias también son de mucha utilidad para la planificación de movimientos basada en sensores. También se han propuesto algoritmos para la solución del problema SLAM (Localización y mapeo simultáneo).

1.2 Objetivo General.

Integrar en una sola plataforma de software, estrategias de exploración y navegación para resolver diversos problemas en robótica móvil.

El presente proyecto tiene como objetivo integrar en una herramienta que denominaremos SRT-BUAP las estrategias para la exploración y la navegación de robots móviles en ambientes desconocidos, conocidos, parcialmente estructurados o estructurados que han desarrollado en los últimos años de investigación.

Los desarrollos se han propuesto y probado en software implantado con diferentes lenguajes de programación, se busca contar con una plataforma común.

1.3 Objetivos Específicos.

Analizar métodos eficientes para el reconocimiento e interpretación de patrones geométricos.

Integrar las diferentes estrategias propuestas en los diferentes proyectos de robótica móvil de nuestro grupo de trabajo.

Desarrollar un sistema robusto para la planificación de movimientos por medio de la recopilación de información de los sensores internos y/o externos.

Construir un sistema en MFC para probar el funcionamiento de los algoritmos propuestos y así obtener una correcta visualización del ambiente desconocido con base a la información recaudada por el robot.

1.4. Bosquejo.

Para conocer un poco más acerca de este proyecto de tesis con el problema de planificación de movimientos usando sensores, se muestra un bosquejo general.

En el capitulo I se presenta una introducción al tema de la robótica móvil y la importancia de su estudio en la actualidad.

En el capítulo II se detallan los aspectos teóricos y de aplicación de los árboles aleatorios de exploración rápida, los cuales son la base del presente trabajo de tesis.

En el capitulo III se describe el método de exploración de ambientes desconocidos que se basa en la generación de una estructura de datos incremental también conocida como árbol aleatorio de exploración usando sensores (SRT); inspirado en los RRTs.

En el capitulo IV se desarrollan estrategias de exploración para ambientes desconocidos donde se propone construir un árbol aleatorio local (AAL).

En el capitulo V presentan los resultados experimentales generados por la presente tesis haciendo una comparación con los diferentes ambientes y las diferentes técnicas que se han implantado en SRT-BUAP.

En el capitulo VI finalmente se menciona las conclusiones de la presente tesis.

Capítulo II.

2.Árboles aleatorios de exploración rápida.

Presentaremos un breve análisis y diseño de los procesos actuales de los 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 Illinous, E.U., [1,2,3,4,5]. La base de estos métodos es la construcción incremental de árboles de búsqueda que intentan explorar rápida y uniformemente el espacio de estados.

Los RRTs son, convenientes para resolver problemas con restricciones diferenciales.

2.1 Formulación del problema.

El tipo de problemas considerado en el enfoque RRT está formulado en términos de seis componentes:

Espacio de estados. Un espacio topológico X.

Valores límite. $x_{ini} \in$ X y $X_{meta} \subset$ X.

Detector de colisión. Una función $D: X \to \{verdadero, falso\}$,que determina si las restricciones globales son satisfechas desde el estado *x*.

Entradas. Un conjunto *U*, que especifica el conjunto de controles o acciones que puedan afectar el estado.

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

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

La planificación de caminos es como una búsqueda en el espacio de estados, X, para un camino continuo desde un espacio inicial, x_{ini} a una región meta $x_{meta} \in$ X.

2.1.1 Planificación de caminos básicas (holonómica).

La planificación de caminos, es una búsqueda en el espacio de configuraciones, C, donde cada configuración $q \in C$ La tarea de la planificación de caminos es calcular un camino continuo desde una configuración inicial q_{ini} a una configuración meta q_{meta} con una representación explicita de X_{libre} donde se detecta a través de un algoritmo de

detección de colisiones. 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.

Este planificador maneja problemas que involucran restricciones no integrables en el estado de velocidades, donde se tienen restricciones diferenciales que generalmente aparecen en forma implícita, $h_i(\dot{q}, q) = 0$ para algún $i = 1 \ldots k < N$ (N es la dimensión de C). Por el teorema de la función implícita, las restricciones se pueden expresar en la forma de control teórico, $\dot{q} = f(q, u)$, u es una entrada elegida de un conjunto de entradas, U. Usando una notación más general, sería $\dot{x} = f(x, u)$. A esta forma se le denomina *la ecuación de transición de estados o ecuación de movimientos.* Al simulador incremental lo podemos construir utilizando esta ecuación de estado por integración numérica, (utilizando, por ejemplo, las técnicas de Runge-Kutta).

2.2 Árboles aleatorios de exploración rápida.

El árbol aleatorio de exploración rápida (RRT), fue presentado en [1], como un algoritmo de planificación para búsquedas rápidas en espacios de altas dimensiones que tienen tanto restricciones algebraicas (proveniente de los obstáculos) como restricciones diferenciales (originadas por la no-holonomía y la dinámica). La idea es dirigir la exploración hacia regiones no exploradas del espacio. Se han propuesto dos técnicas aleatorias de planificación de caminos (planificación holonómica): el algoritmo del Hilo de Ariadna [6] y los planificadores de [7], que intentan "empujar" la búsqueda lejos de los vértices previamente construidos.

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, como se ilustra en la figura 2.2, selecciona del árbol vértice más cercano a un estado dado. Este vértice se elige de acuerdo a una métrica, ρ. La función NUEVO_ESTADO hace un movimiento hacia x aplicando una entrada $u \in$ U para algún incremento Δt. Esta entrada puede seleccionarse aleatoriamente o probando para producir un nuevo estado.

NUEVO_ESTADO utiliza implícitamente una función de detección de colisiones para determinar si el nuevo estado satisface las restricciones globales. Si NUEVO_ESTADO se cumple pueden ocurrir tres situaciones: *Alcanzado,* el nuevo vértice alcanza el estado muestreado x, *Avanzado* un nuevo vértice $x_{nuevo} \neq x$, y *Atrapado*, NUEVO_ESTADO, falla en producir un nuevo estado que se encuentre en X_{libre}.

Construir_RRT(x_{ini})
1. ***T***.ini(x_{ini});
2. **Para** *k = 1* **a** K
3. X_{aleat} ← ESTADO_ALEATORIO();
4. EXTENDER(***T***, x_{aleat});
5. Regresar ***T***

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

Figura 2.1 Algoritmo básico de construcción del RRT.

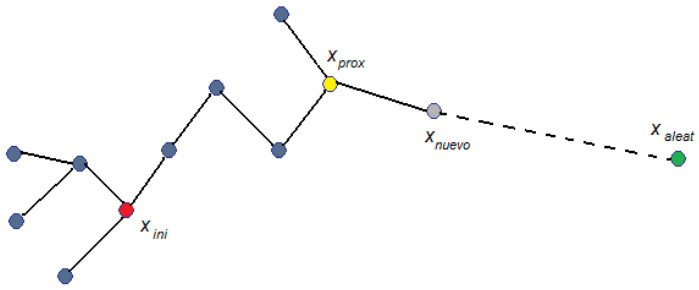


Figura 2.2 Operación de la función EXTENDER.

2.3 Diseño de planificadores de caminos.

Generalmente el algoritmo RRT es un planificador eficiente. Hay varias alternativas de planificadores que utilizan 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 bidireccionales.

Inspirados en las técnicas clásicas de búsqueda bidireccional, de la Inteligencia Artificial es razonable esperar un mejor desempeño al crecer dos árboles de

exploración, uno desde x_{ini} y el otro a partir de x_{meta}; se obtiene una solución cuando los dos árboles se encuentran. La construcción del RRT debe guiarse para asegurar que ambos árboles se encuentren antes de cubrir el espacio entero y permitir una unión eficaz.

La figura 2.3 muestra el algoritmo RRT_BIDIRECCIONAL que divide el tiempo de cómputo entre dos procesos:

(1) Explorar el espacio de estados. (2) Intentar crecer los árboles uno hacia el otro.

```
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_meta) = Alcanzado) entonces
6.                        Regresa CAMINO(T_a, T_b);
7.        INTERCAMBIAR(T_a, T_b);
8. Regresa Fallo;
```

Figura 2.3 Planificador bidireccional usando un RRT.

2.3.2 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 la otra que elija, con cierta probabilidad, entre x_{meta} o un estado tomado de cualquier parte del espacio de estados. Incluso con una probabilidad pequeña (por ejemplo 0.05) de elegir x_{meta}, RRT_GoalBias usualmente converge al objetivo mucho más rápido que el RRT básico. Pero, si se designa un sesgo muy grande hacia el objetivo entonces el planificador empieza a comportarse como un planificador aleatorio de campos de potencial, ya que pueden presentarse mínimos locales. Una mejora es RRT_GoalZoom el cual reemplaza la función ESTADO_ALEATORIO con una decisión, basada en cierta probabilidad, de optar por un estado aleatorio dentro de una región circunvecina al objetivo o por uno elegido del espacio de estados completo. El tamaño de la región alrededor del objetivo lo controla el vértice del árbol más cercano al objetivo. El efecto es que el foco de muestreo converge al objetivo como el RRT se acerca a él. Este planificador funciona bastante bien en la práctica, sin embargo, su rendimiento aún puede degradarse debido a mínimos locales. En general, parece mejor sustituir la función ESTADO_ALEATORIO con un esquema de muestreo que extraiga los estados desde una función de densidad de probabilidad no uniforme con un sesgo

gradual hacia el objetivo. La figura 2.4 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 con céntricos.

Un problema más a considerarse, es el tamaño del paso usado en la construcción del RRT. Este podría elegirse dinámicamente durante la ejecución, acorde con la función que calcula la distancia en la detección de colisión. Si los cuerpos están lejos de colisionar, se puede tomar un paso grande. Además se seguir esta idea ¿qué tan lejos debe estar x_{nuevo} de x_{prox}?. 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 2.5. Al sustituir EXTENDER por CONECTAR el árbol crece muy rápido, si lo permiten las restricciones de detección de colisión y las restricciones diferenciales. Una de las ventajas clave de la función CONECTAR es que construiremos 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.

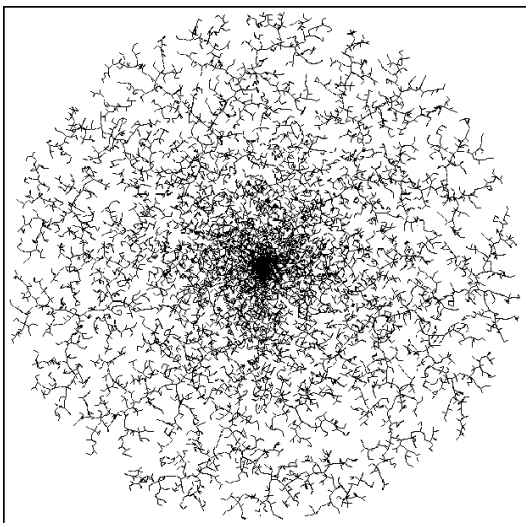


Figura 2.4 Un RRT en 2D construido usando un muestreo sesgado.

```
CONECTAR(T,x)
1. repetir
2.        S←EXTENDER(T,x);
3. hasta que no(S = Avanzado)
5. Regresa S
```

Figura 2.5 Función CONECTAR.

Los planificadores bidireccionales están 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 a partir 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. En la figura 2.3 muestra el algoritmo RRT_BIDIRECCIONAL, puede compararse con el algoritmo CONTRIR_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 T_a y 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 [4] para planificación kinodynamic, en ese enfoque, en cada iteración ambos árboles crecía 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 considera 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 de denominamos RRT_ExtCon y al algoritmo original como RRT_ExtExt. Entre las variantes discutidas hasta el momento, encontraremos a RRT_ExtCon más eficiente para problemas holonómicos 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 kynodynamic 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.

Si un enfoque dual ofrece ventajas sobre un árbol simple, entonces es natural peguntar si el crecimiento de tres o más árboles ofrecería mayores ventajas. Estos árboles adicionales, iniciarían en estados aleatorios. Por supuesto, los problemas de conexión en el caso de la planificación no-holonómica sería aun más difícil 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 limite, 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 2.5 a cada RRT. Para mejorar el rendimiento podemos considerar los vértices que están a una distancia fija de x_{aleat} de acuerdo con la métrica. Si la conexión tiene éxito entonces los dos árboles se unen en un único grafo.

2.4 Aplicación 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 para diseñar prototipos virtuales como un "conductor virtual de prueba".

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

Las restricciones diferenciales son cinemáticas. Por lo que el espacio de estados se reduce al espacio de configuraciones. El modelo es tri-dimensional, como en la figura 2.6, cualquier estado se representa por medio de (x,y,θ) y su ecuación de transición de estado es:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} \equiv \begin{pmatrix} s\cos\theta \\ ssen\theta \\ \frac{s}{L}\tan\phi \end{pmatrix}$$

(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, $|\phi| \leq \phi_{\max} < {}^{\pi}/_{2}$. El vector de entrada es (s, ϕ). Si el carro viaja sólo hacia adelante, fijamos a $s = 1$, obteniendo así el carro Dubins [8]. Si el carro de mueve hacia atrás y hacia adelante $s = -1$ ó $s = 1$, obtenemos un carro Reeds-Shepp [9].

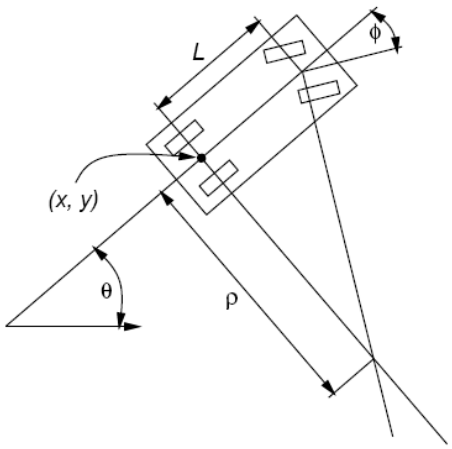


Figura 2.6 Modelo cinemático para un robot tipo carro.

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 y la velocidad angular, ω, del ángulo de conducción.

El resultado de 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:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{\theta} \end{pmatrix} \equiv \begin{pmatrix} s\cos\theta \\ ssen\theta \\ \omega \\ \frac{s}{L}\tan\phi \end{pmatrix}$$

(2.2)

Siempre existen dos árboles $\boldsymbol{T}_a$ y $\boldsymbol{T}_b$, hasta que se enlazan y se encuentra una solución. El crecimiento de dos RRTs también fue propuesto en [4], para planificación kinodynamic, considerando algunas variaciones se puede reemplazar cualquier ocurrencia de EXTENDER con CONECTAR en el RRT_BIDIRECCIONAL.

Una limitación al utilizar un RRT bidireccional en problemas de planificación no-holonómicas y kinodynamic es que necesitamos conectar varios vértices para unir un RRT al otro.

Capítulo III.

3. Métodos SRT para ambientes desconocidos.

Oriolo, Vendittelli, Freda y Troso presentan en [10] 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). Pueden obtenerse diversas estrategias de exploración adaptando al método general, diferentes técnicas de percepción. En [10] 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, confía más en la información reportada por los sensores. La estrategia desarrollada es esta tesis sigue las dos líneas, además de la propuesta que se denomina SRT_Radial.

3.1 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 limites (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 un caso de aprendizaje activo de orden-sensitivo, por que el flujo de datos es resultado de todas las acciones pasadas del robot. El problema central de la exploración es como seleccionar la siguiente acción. La exploración basada en fronteras se logra cuando el criterio es la maximización de la utilidad de las acciones. Sin

embargo, existe otra opción, es decir, usar un mecanismo de selección aleatoria (también llamado camino aleatorio). Las ventajas de esta elección son:

Simplicidad. El hecho de que cualquier secuencia de acciones se ejecutará eventualmente.

La última propiedad mencionada lleva a la completitud, es decir, 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 está disponible progresivamente y el robot se traslada para recopilar más información. Por lo cual, los planificadores aleatorios son considerados como estrategias de exploración orientados a objetivos utilizando la selección aleatoria de acciones. 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.

3.2 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 el punto de vista general, es decir, independiente de una estrategia de percepción particular. Finalmente, se presenta la variante adoptada y los resultados obtenidos por el medio de herramienta de simulación desarrollada.

3.2.1 Hipótesis de trabajo.

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

El espacio de trabajo es plano, es decir, R^2 o un subconjunto de R^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 (para evitar colisiones).

El robot siempre conoce su configuración q, (localización perfecta).

El robot está 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.

Una pequeña región del espacio físico circundante al robot y al sistema de sensado en la configuración inicial es libre, este requerimiento es importante por que de lo contrario ningún movimiento puede ejecutarse después del primer sensado.

3.2.2 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 libres. 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) esté contenida en la región segura local. El logaritmo que implementa el método SRT se describe en la figura 3.1.

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

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 3.2. Una nueva configuración candidata q_{cand} se determina

tomando un paso de longitud $\alpha \cdot r$ en dirección a θ_{rand}. La constante $\alpha < 1$ garantiza que q_{cand} se encuentre 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.

```
CONSTRUIR_SRT( q_ini, k_max, l_max, α, d_dim)
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 ← DESPLAZA(q_act, θ_rand, α · r);
10.             i ← i+1;
11.      hasta que (VALIDA(q_cand, d_min, T) o i = l_max)
12.      si VALIDA(q_cand, d_min, T) entonces
13.             MOVER_A(q_act);
14.             q_act ← q_cand;
15.      sino
16.             MOVER_A(q_act, padre);
17.      q_act ← q_act . padre;
18. Regresa T;
```

3.1 Algoritmo básico de construcción del SRT.

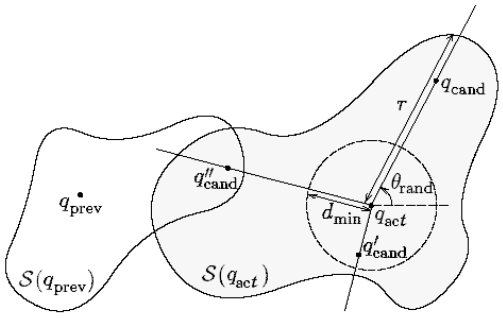


Figura 3.2 Generación de configuraciones candidatas con el método SRT. En este caso, q_{cand}, es valida, mientras q_{cand}' y q'_{cand}' no lo son, la primera se encuentra a una distancia menor a d_{min} de q_{act} y q''_{cand} se ubica en la región segura local de otro nodo.

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 3.2, q_{cand}

(i) Debe estar alejada de q_{cand} a una distancia mayor a una distancia mínima prefijada d_{min}

(ii) No debe situarse en la región segura local de otra configuración previa en 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 retroceso a la configuración inicial.

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

En comparación con el RRT, la estructura de SRT es un árbol con arcos 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 pequeños en virtud de objetos a su paso. También, no es necesario un chequeo de colisiones ya que las configuraciones candidatas son generadas dentro del área segura.

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

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

3.3 Exploración con SRT-Radial.

Como se menciono, la forma de la región segura local S refleja las características del sensor, así como la técnica de percepción adoptada. A su vez, la estrategia de exploración estará fuertemente afectada por la forma de S. En [10] se presenta una variante del método llamada SRT-Star, la cual involucra una estrategia de percepción que toma completamente 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. El radio del i-ésimo como 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 la 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}. 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.

En el capitulo 5 se presentan los resultados del método SRT, se compararan ambas estrategias (SRT_Star y SRT_Radial).

Capítulo IV.

4. Exploración usando sensores con AAL.

En este capítulo desarrollamos algunas estrategias 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 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 libres X_{libre} conocido asociado a una región segura reportada por el sensor. El árbol se expandirá incrementalmente cuando el sensor tome nuevas percepciones del ambiente hasta que el estado meta sea un nodo del AAL, es decir, hasta encontrar un cambio factible libre de colisiones, o hasta un número de iteraciones máximo.

El robot móvil equipado con un sensor, (sensor telemétrico rotacional a 360º ó un sensor laser) que proporciona el área libre de obstáculos alrededor del robot, necesita planificar y ejecutar movimientos libre 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 esencialmente en dos etapas [11]:

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.

4.1 Estructura de datos.

Planteamos la construcción de una estructura de datos tipo árbol, llamado árbol aleatorio local (ALL), 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.

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 terminales. La descripción del robot, los obstáculos y la RSL es poligonal.

4.2 Proceso de exploración.

El sistema empieza en el estado inicial. En el estado actual se efectúa 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 al AAL e incrementando el ambiente explorado. Cuando no se tenga 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 en la figura 4.1.

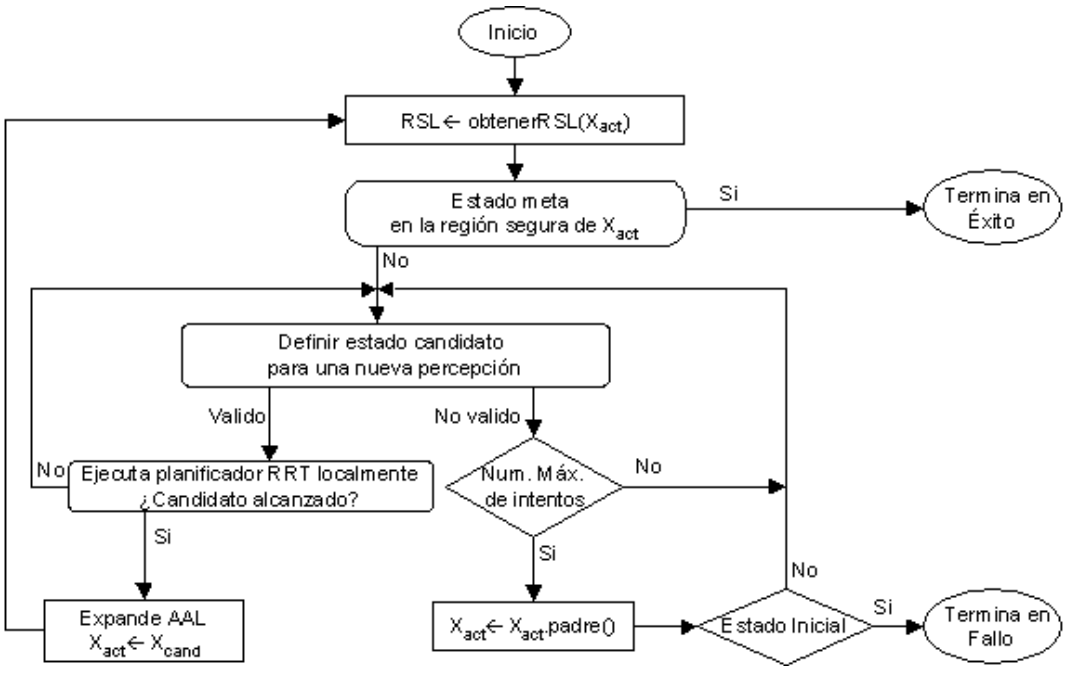


Figura 4,1 Flujo del proceso de exploración usando sensores.

4.3 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.

```
CONSTRUIR( x_inic, k_max, l_max)
1. x_act = x_inic;
2. k ←0;
3. mientras (NO(META_EN_RSL(x_act)) o NO(x_act = NULL) o k < k_max)
4.        S ← PERCEPCION(X_act);
5.        AGREGA(T,(x_act, S));
6.        i←0;
7.        repetir
8.                 x_cand ←EDO_CANDIDATO(x_act);
9.                 ExisteCamino ← PLANIFICADOR_RRT(x_act,x_cand);
10.                i ←i+1;
11.       hasta que (ExisteCamino o i = l_max)
12.                si ExisteCamino entonces
13.                         MOVER_A(x_cand);
14.                         x_act ← x_cand;
15.                sino
16.                         MOVER_A(x_act . padre);
17.                         x_act ← x_act . padre;
18.       k ← k+1;

19. si (META_EN_RSL(x_act))entonces
20.       ExisteCamino ← PLANIFICADOR_RRT(x_act, x_meta);
21.       si ExisteCamino entonces
22.                Regresa ÉXITO;
23. si  ExisteCamino entonces
24.       "robot en posición inicial";
25. Regresa FALLO;
```

Figura 4.2 Algoritmo básico.

En cada iteración k del algoritmo, ver figura 4.2, se efectúa 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 determinada 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 el 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 junto a las principales rutinas del algoritmo.

4.4 Estrategia SRRT_Local.

Existen dos variantes de SRRT_Local, denominada así por que 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.

4.4.1 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_{cand} . 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 3.1.

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 a partir 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 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 el nodo terminal reduce el esfuerzo de cómputo y mantiene la capacidad de exploración del algoritmo.

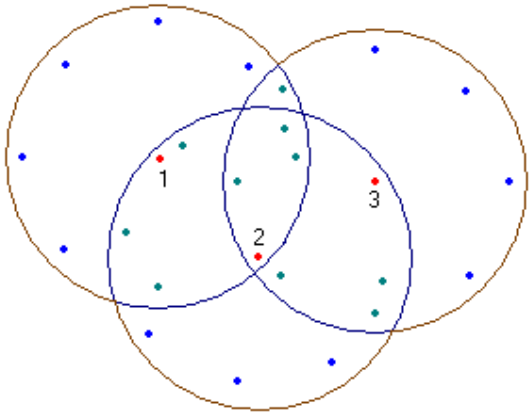


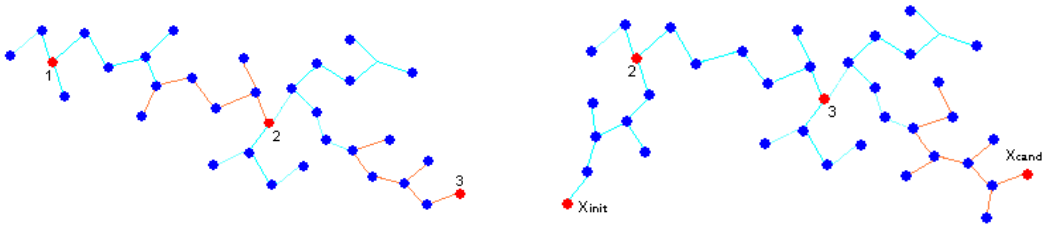
Figura 4.3 Estados candidatos validos.

El número de nodos candidatos fue elegido experimentalmente, dando buenos resultados. Los ocho candidatos seleccionados están distribuidos equidistantes uno del otro cerca de la frontera de la región segura con el espacio desconocido. La figura 4.3 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 homogéneamente 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 4.2 sea variable dependiendo del número de candidatos validos disponibles localmente.

4.4.2 Planificador RRT.

Las variantes de la estrategia SRRT_Local adapta un planificador RRT_Bidireccional local. En la primera variante, T_a tiene como raíz a x_{act} y 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 está conformado por pequeños árboles bidireccionales locales, actuando los nodos terminal, donde se realiza el sensado, como nodos de unión.



a) b)

Figura 4.4 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}.

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 T_a con raíz en x_{ini}, crece al agregarle los nodos del árbol local T_b el cual tiene como raíz a x_{cand}. La figura 4.4 muestra los árboles aleatorios de exploración creados con ambas variantes.

4.5 Estrategia SRRT_LocalB.

La estrategia de exploración SRRT_LocalB adapta el planificador RRT_GoalBias (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 determina los estados candidatos. Las características 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}).

4.5.1 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 algoritmo 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 constituido a partir del x_{act}. 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 la figura 4.5. Nuevamente los estados candidatos validos son almacenados en una lista. Se sigue la misma heurística del SRRT_Local para guiar el proceso de exploración, moviéndose primero a los estados más cercanos a x_{act}.

Las funciones restantes del algoritmo 4.2 son iguales para las estrategias de exploración y sus variantes.

4.5.2 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 inicial y la posición meta y comparándola 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.

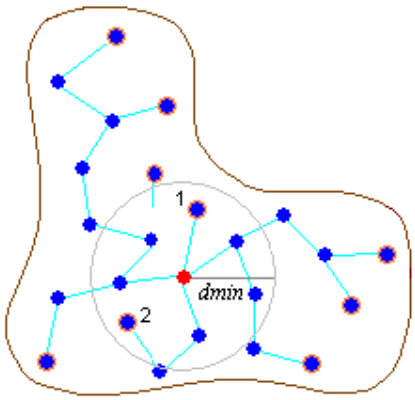


Figura 4.5: La mayoría de los nodos hojas contienen estados candidatos válidos, sólo los nodos 1 y 2 no lo son por estar a una distancia menor a d_{min}.

4.5.3 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.

4.5.4 PERCEPCION(x_{act}).

El proceso de percepción del ambiente como su nombre lo indica: obtiene información a través de nuevas percepciones como mención de forma explicada en el capitulo anterior.

Capítulo V.

5. *Resultados y experimentos.*

Para ilustrar el comportamiento de exploración de las dos estrategias mencionadas en este proyecto (SRT_Radial y SRT_star) a continuación se presentan los resultados de las variantes a los diferentes métodos.

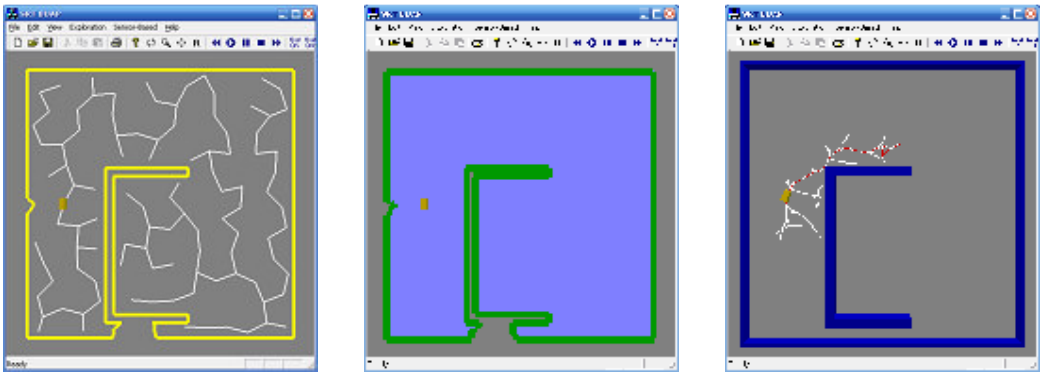
Los algoritmos se desarrollaron en Visual C++ v. 6.0 aprovechando las librerías MFC y MSL, así como su interfaz gráfica para poder visualizar el ambiente de trabajo y la animación del camino calculado. Para simular el módulo de percepción del sistema de sensado se utilizó la librería GPC. Los apéndices A, B y C detallan dichas librerías.

Los resultados obtenidos fueron realizados en una computadora personal bajo el sistema operativo Windows XP con un procesador AMD Athlon (TM) 64 X2 Dual-Core TK- 57 1.90 Ghz, 1G de memoria RAM.

En general los algoritmos han sido probados en diferentes ambientes tomando en cuenta los siguientes valores: $k_{max} = 250$, $I_{max} = 10$, $\alpha = 0.7$, $d_{min} = 3.5$.

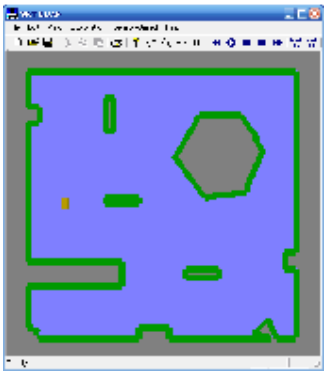
5.1 Comportamiento SRT_Extensive con sensado radial.

Las figuras muestran las áreas exploradas con el método SRT_Extensive y la estrategia SRT_Radial, así como la planificación con la ayuda del algoritmo RRT_ExtExt generado en el área explorada.

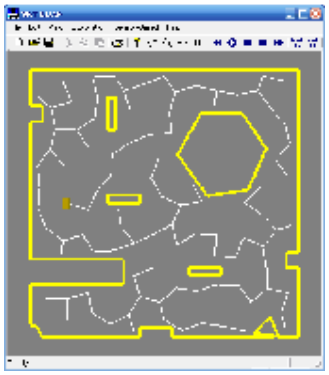


SRT | Mapa obtenido | Un ejemplo de planificación

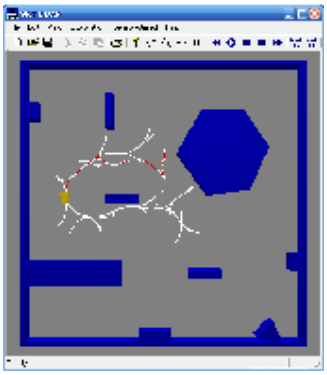
Figura 5.1 Ambiente 1.



Mapa obtenido



SRT



Ejemplo de planificación

Figura 5.2 Ambiente 3.

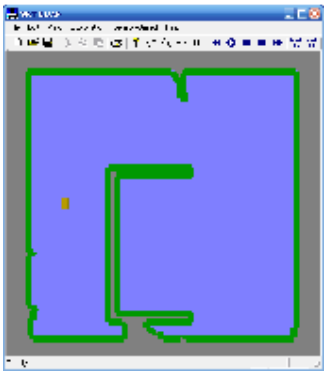
Ambiente 1	Exploración Extensive
Número de nodos máximo	111
Número de nodos mínimo	97
Tiempo de planificación máximo	62.219 seg.
Tiempo de planificación mínimo	50.968 seg.

Tabla 5.1 Resultados de las estrategias de exploración extensiva con tiempos máximos y mínimos en el ambiente 1 con el sensor Radial.

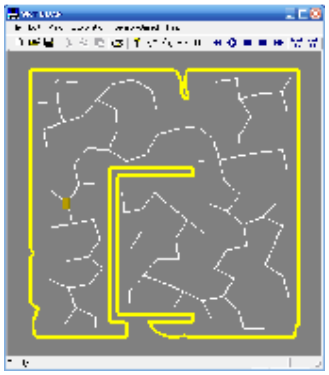
Como se puede apreciar tanto en las figuras como en la tabla el número de nodos siempre varia dependiendo del comportamiento del algoritmo, así como el tiempo de ejecución.

5.2 Comportamiento SRT_Extensive con sensado star.

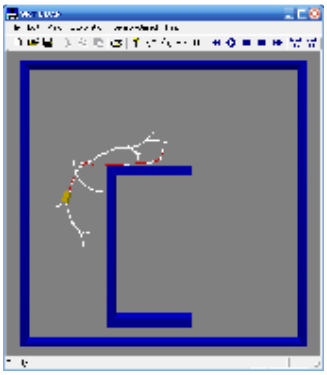
Las figuras muestran las áreas exploradas con el método SRT_Extensive y la estrategia SRT_star, así como la planificación con la ayuda del algoritmo RRT_ExtExt generado en el área explorada.



Mapa obtenido

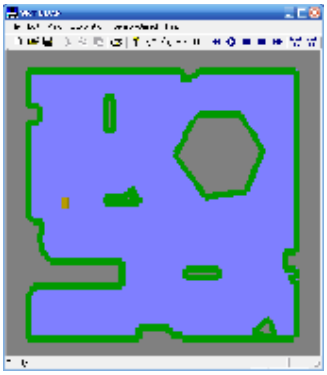


SRT

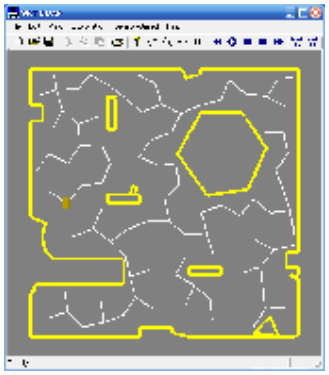


Ejemplo de planificación

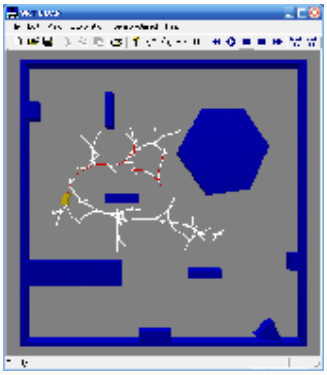
Figura 5.3 Ambiente 1.



Mapa obtenido



SRT



Ejemplo de planificación

Figura 5.4 Ambiente 3.

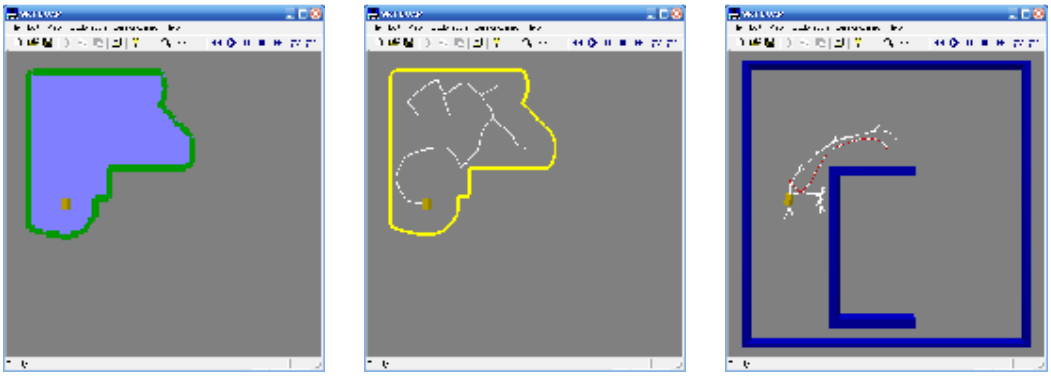
Ambiente 1	Exploración Extensive
Número de nodos máximo	98
Número de nodos mínimo	46
Tiempo de planificación máximo	65.43 seg.
Tiempo de planificación mínimo	16.06 seg.

Tabla 5.2 Resultados de las estrategias de exploración extensiva con tiempos máximos y mínimos en el ambiente 1 con el sensor Star.

En la exploración extensiva con un sensor star, el algoritmo puede tener un menor número de nodos y hacer una exploración completa, pues depende más de los datos proporcionados por el sensor que el ambiente en el que se encuentra.

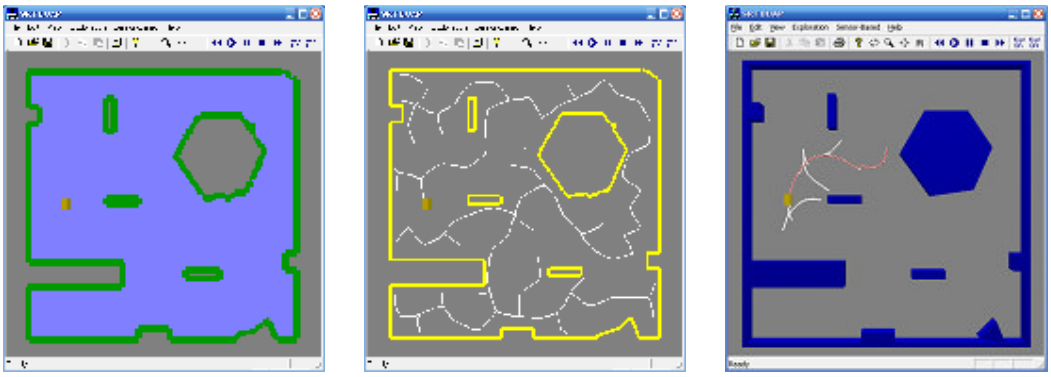
5.3 Comportamiento SRT_Goal con sensado radial.

Las figuras muestran las áreas exploradas con el método SRT_Goal y la estrategia SRT_Radial, así como la planificación con la ayuda del algoritmo RRT_ExtExt generado en el área explorada.



Mapa obtenido SRT Ejemplo de planificación

Figura 5.5. Ambiente 1.



Mapa obtenido SRT Ejemplo de planificación

Figura 5.6 Ambiente 3.

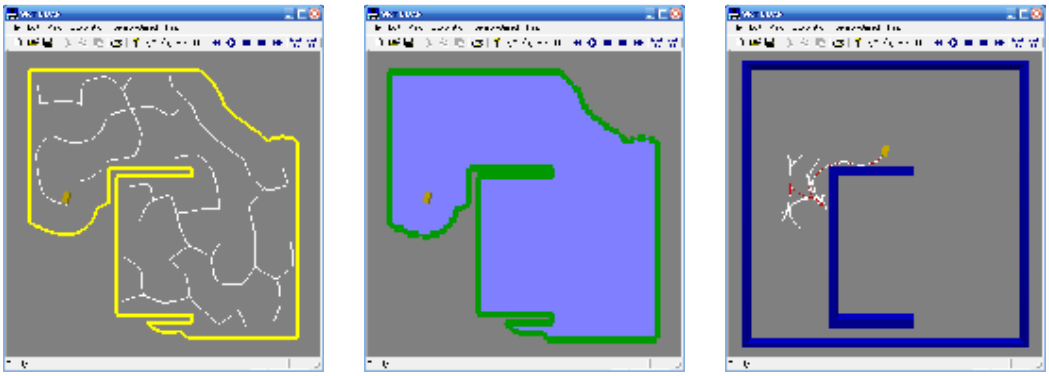
Ambiente 1	Exploración Goal
Número de nodos máximo	126
Número de nodos mínimo	25
Tiempo de planificación máximo	225.141 seg.
Tiempo de planificación mínimo	4 seg.

Tabla 5.3 Resultados de las estrategias de exploración meta con tiempos máximos y mínimos en el ambiente 1 con el sensor Radial.

Como se muestra en la figura 5.5 no necesariamente necesita analizar todo el ambiente para poder decir si ha encontrado su objetivo, por lo que el número de nodos como se muestra en la tabla 5.3 puede tomar el ambiente su mínimo de nodos al igual que el tiempo de ejecución puede ser muy rápida.

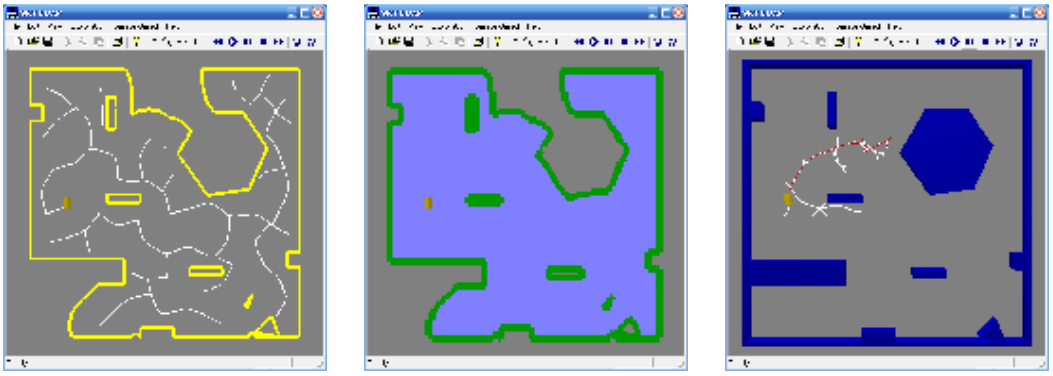
5.4 Comportamiento SRT_Goal con sensado star.

Las figuras muestran las áreas exploradas con el método SRT_star y la estrategia SRT_star, así como la planificación con la ayuda del algoritmo RRT_ExtExt generado en el área explorada.



SRT Mapa obtenido Un ejemplo de planificación

Figura 5.7 Ambiente 1.



SRT Mapa obtenido Un ejemplo de planificación

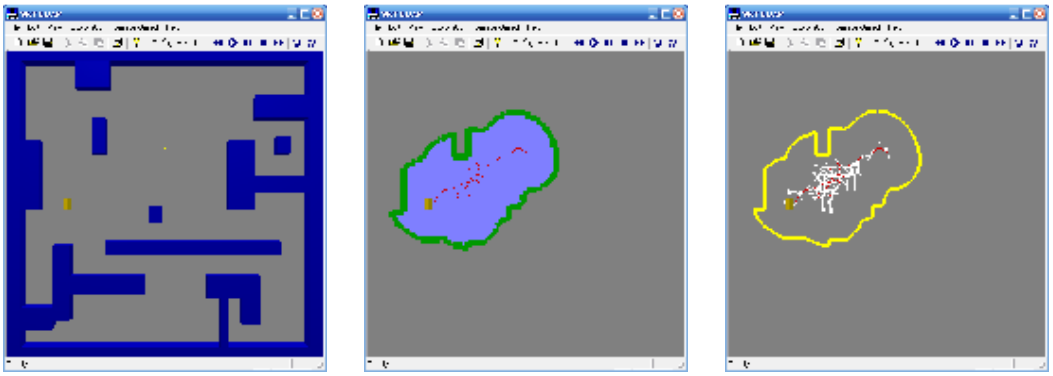
Figura 5.8 Ambiente 3.

Ambiente 1	Exploración Goal
Número de nodos máximo	82
Número de nodos mínimo	5
Tiempo de planificación máximo	170.53 seg.
Tiempo de planificación mínimo	3.34 seg.

Tabla 5.4 Resultados de las estrategias de exploración meta con tiempos máximos y mínimos en el ambiente 1 con el sensor Star.

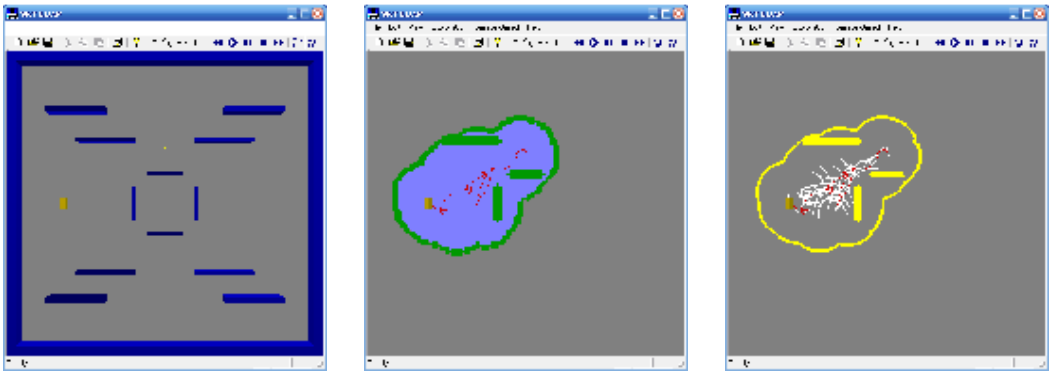
5.5 Comportamiento SRRT_local_v1 con sensado radial.

En las siguientes figuras mostramos algunos ejemplos de los árboles aleatorios de exploración obtenidos, junto con los caminos encontrados por el algoritmo SRRT_local_v1 con la estrategia sensor_radial.



Ambiente original Mapa obtenido Árbol obtenido

Figura 5.9 Ambiente 5.



Ambiente original | Mapa obtenido | Árbol obtenido

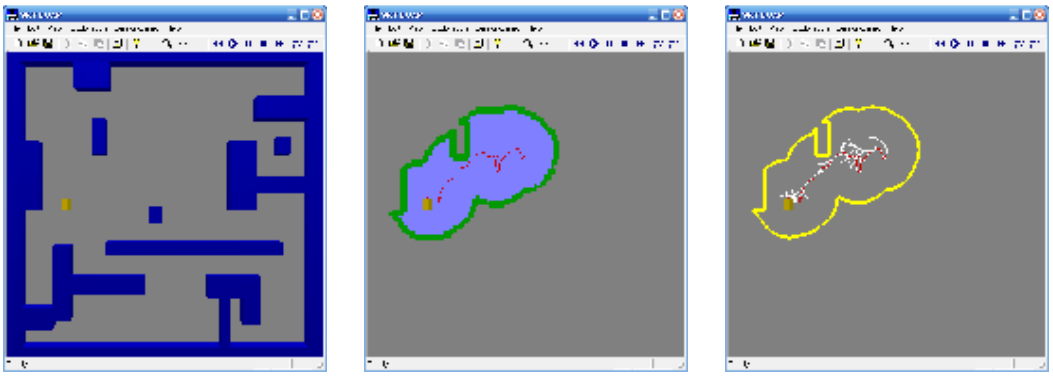
Figura 5.10 Ambiente 6.

Ambiente 5	SRRT_Local_v1
Número de nodos máximo	245
Número de nodos mínimo	35
Tiempo de planificación máximo	20.6399 seg.
Tiempo de planificación mínimo	3.28101 seg.

Tabla 5.5 Resultados de las estrategias de SRRT_Local_v1 con tiempos máximos y mínimos en el ambiente 5 con el sensor Radial.

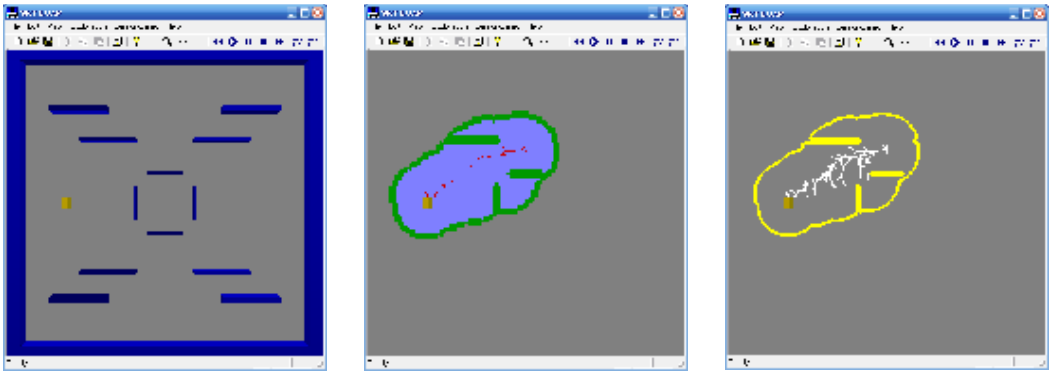
5.6 Comportamiento SRRT_local_v1 con sensado star.

En las siguientes figuras mostramos algunos ejemplos de los árboles aleatorios de exploración obtenidos, junto con los caminos encontrados por el algoritmo SRRT_local_v1 con la estrategia sensor_star.



Ambiente original Mapa obtenido Árbol obtenido

Figura 5.11 Ambiente 5.



Ambiente original Mapa obtenido Árbol obtenido

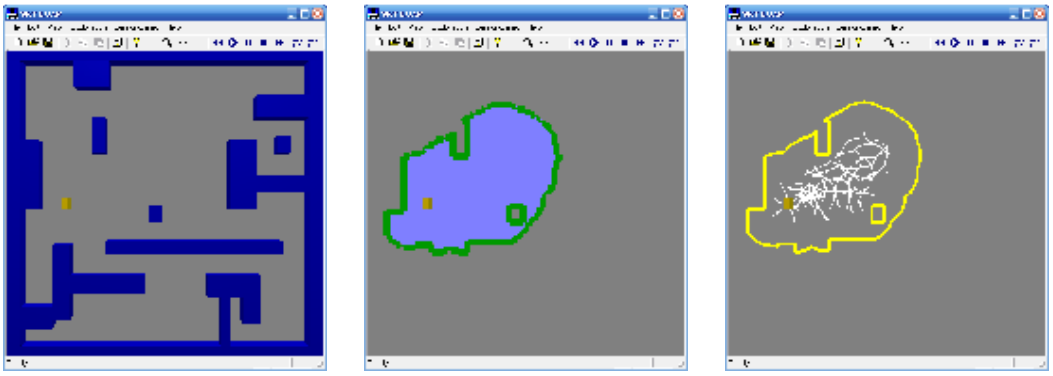
Figura 5.12 Ambiente 6.

Ambiente 5	SRRT_Local_v1
Número de nodos máximo	116
Número de nodos mínimo	6
Tiempo de planificación máximo	18.95 seg.
Tiempo de planificación mínimo	8.37seg.

Tabla 5.6 Resultados de las estrategias de SRRT_Local_v1 con tiempos máximos y mínimos en el ambiente 5 con el sensor Star.

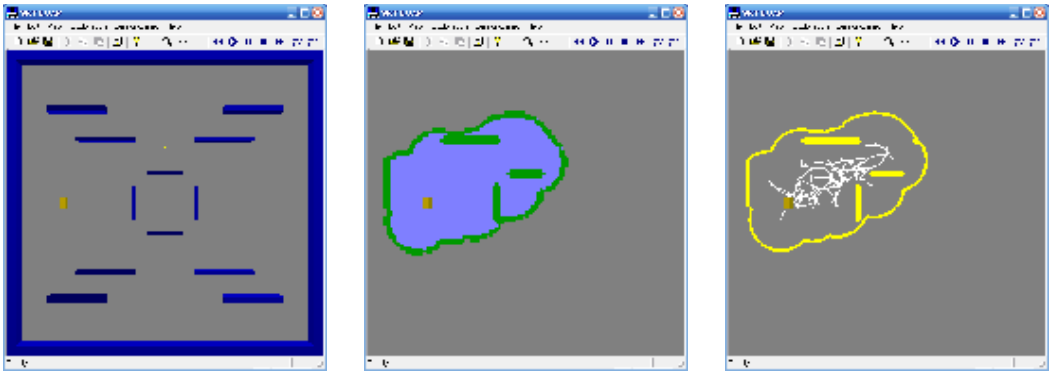
5.7 Comportamiento SRRT_local_v2 con sensado radial.

En las siguientes figuras mostramos algunos ejemplos de los árboles aleatorios de exploración obtenidos, junto con los caminos encontrados por el algoritmo SRRT_local_v2 con la estrategia sensor_radial.



Ambiente original | Mapa obtenido | Árbol obtenido

Figura 5.13 Ambiente 5.



Ambiente original | Mapa obtenido | Árbol obtenido

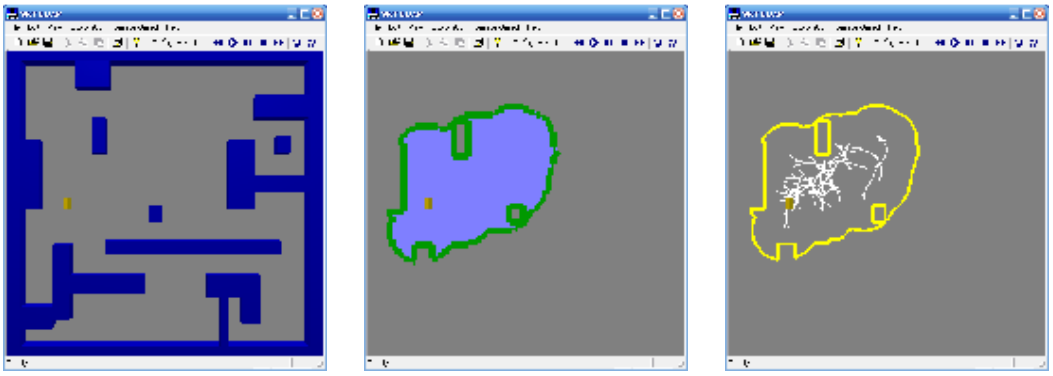
Figura 5.14 Ambiente 6.

Ambiente 5	SRRT_Local_v2
Número de nodos máximo	1202
Número de nodos mínimo	470
Tiempo de planificación máximo	14.92 seg.
Tiempo de planificación mínimo	8.64 seg.

Tabla 5.7 Resultados de las estrategias de SRRT_Local_v2 con tiempos máximos y mínimos en el ambiente 5 con el sensor Radial.

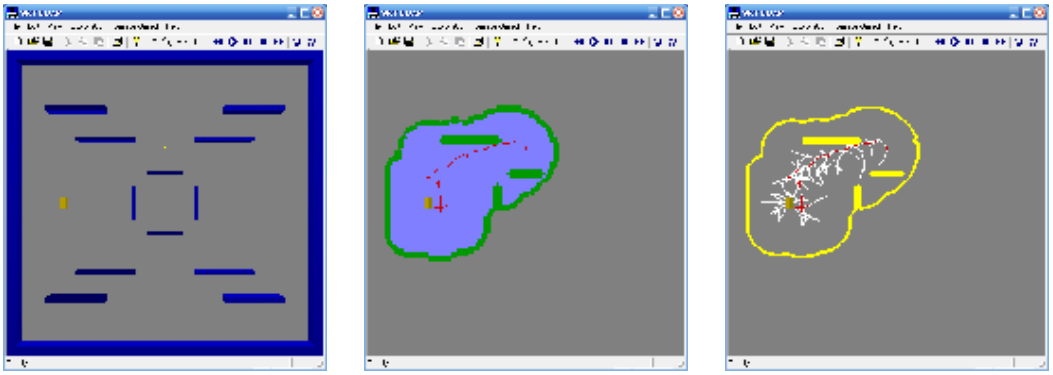
5.8 Comportamiento SRRT_local_v2 con sensado star.

En las siguientes figuras mostramos algunos ejemplos de los árboles aleatorios de exploración obtenidos, junto con los caminos encontrados por el algoritmo SRRT_local_v2 con la estrategia sensor_star.



Ambiente original | Mapa obtenido | Árbol obtenido

Figura 5.14 Ambiente 5.



Ambiente original | Mapa obtenido | Árbol obtenido

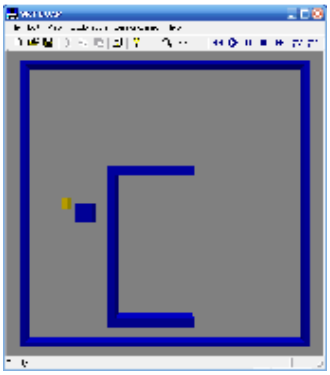
Figura 5.15 Ambiente 6.

Ambiente 5	SRRT_Local_v2
Número de nodos máximo	1309
Número de nodos mínimo	786
Tiempo de planificación máximo	37.82 seg.
Tiempo de planificación mínimo	19.18 seg.

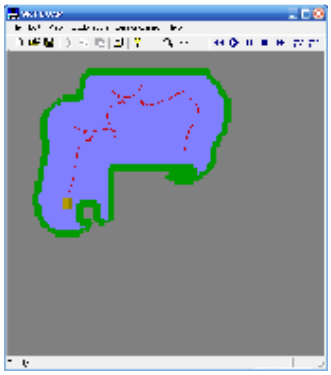
Tabla 5.8 Resultados de las estrategias de SRRT_Local_v2 con tiempos máximos y mínimos en el ambiente 5 con el sensor Star.

5.9 Comportamiento SRRT_localB_v1 con sensado radial.

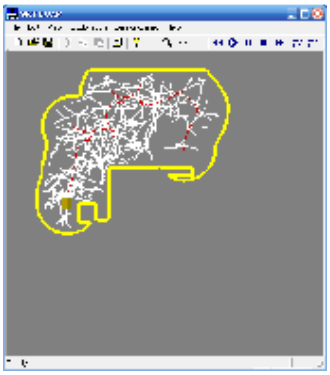
En las siguientes figuras mostramos algunos ejemplos de los árboles aleatorios de exploración obtenidos, junto con los caminos encontrados por el algoritmo SRRT_localB_v1 con la estrategia sensor_radial.



Ambiente original

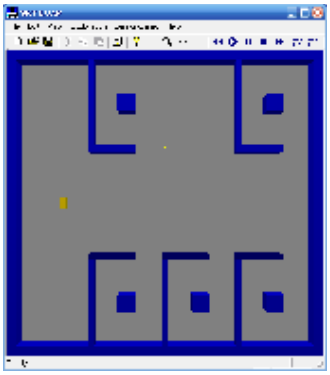


Mapa obtenido

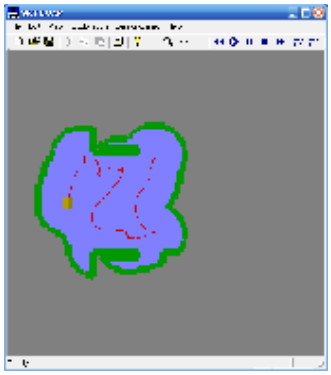


Árbol obtenido

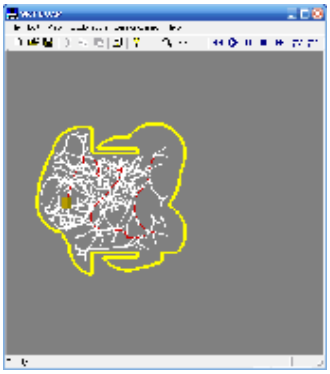
Figura 5.16 Ambiente 7.



Ambiente original



Mapa obtenido



Árbol obtenido

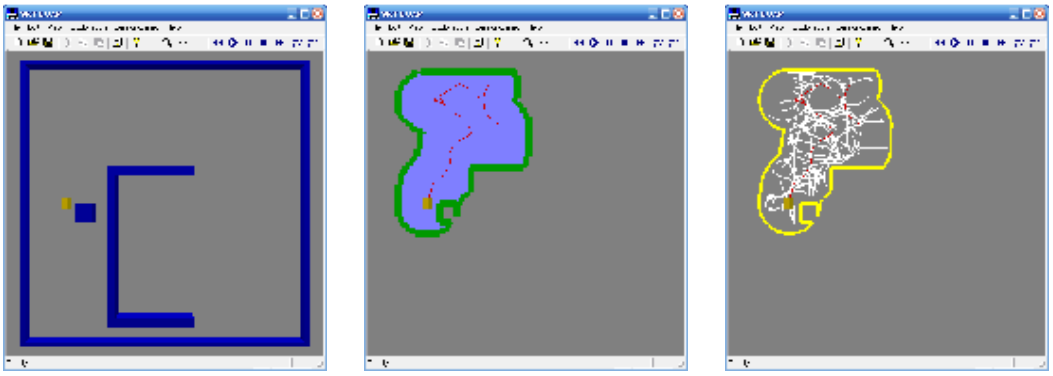
Figura 5.17. Ambiente 8.

Ambiente 7	SRRT_LocalB_v1
Número de nodos máximo	3051
Número de nodos mínimo	887
Tiempo de planificación máximo	63.76 seg.
Tiempo de planificación mínimo	14.34 seg.

Tabla 5.9 Resultados de las estrategias de SRRT_LocalB_v1 con tiempos máximos y mínimos en el ambiente 7 con el sensor Radial.

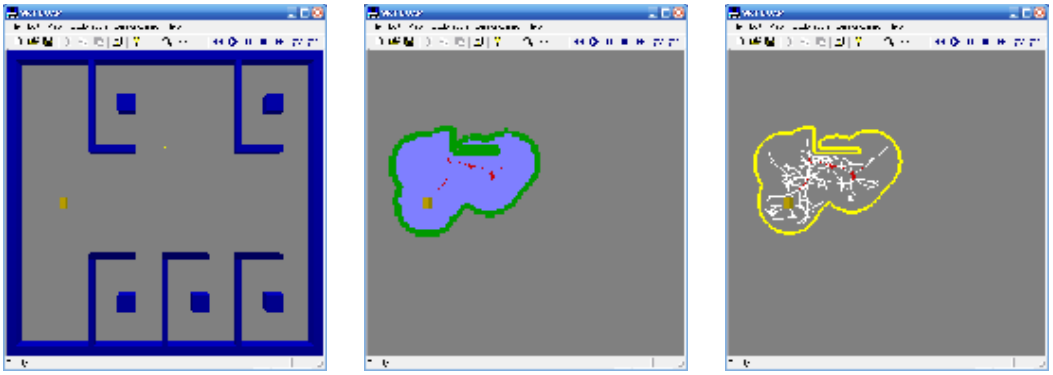
5.10 Comportamiento SRRT_localB_v1 con sensado star.

En las siguientes figuras mostramos algunos ejemplos de los árboles aleatorios de exploración obtenidos, junto con los caminos encontrados por el algoritmo SRRT_localB_v con la estrategia sensor_star.



Ambiente original | Mapa obtenido | Árbol obtenido

Figura 5.18 Ambiente 7.



Ambiente original | Mapa obtenido | Árbol obtenido

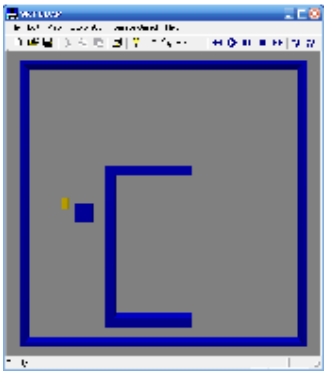
Figura 5.19 Ambiente 8.

Ambiente 7	SRRT_LocalB_v1
Número de nodos máximo	3350
Número de nodos mínimo	934
Tiempo de planificación máximo	107.90 seg.
Tiempo de planificación mínimo	28.59 seg.

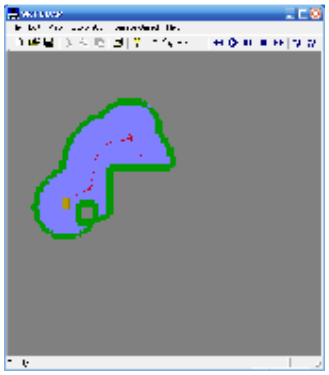
Tabla 5.10 Resultados de las estrategias de SRRT_LocalB_v1 con tiempos máximos y mínimos en el ambiente 7 con el sensor Star.

5. 11Comportamiento SRRT_localB_v2 con sensado radial.

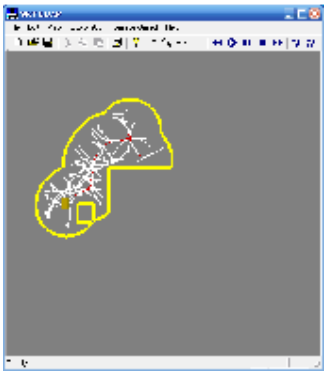
En las siguientes figuras mostramos algunos ejemplos de los árboles aleatorios de exploración obtenidos, junto con los caminos encontrados por el algoritmo SRRT_localB_v2 con la estrategia sensor_radial.



Ambiente original

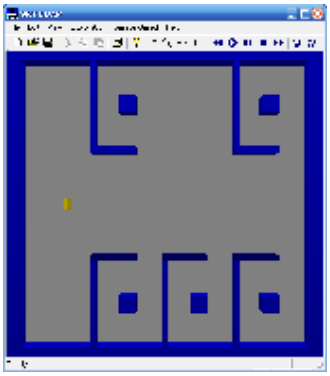


Mapa obtenido

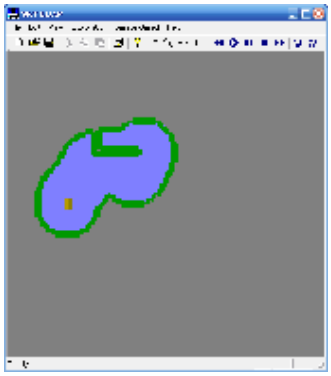


Árbol obtenido

Figura 5.19 Ambiente 7.



Ambiente original



Mapa obtenido

Árbol obtenido

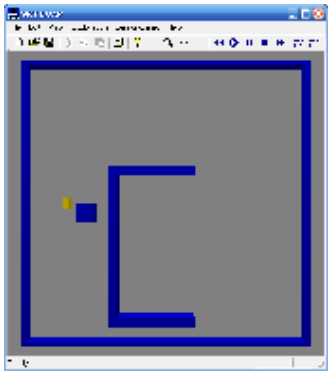
Figura 5.20. Ambiente 8.

Ambiente 7	SRRT_LocalB_v2
Número de nodos máximo	1142
Número de nodos mínimo	701
Tiempo de planificación máximo	16.93 seg.
Tiempo de planificación mínimo	11.07 seg.

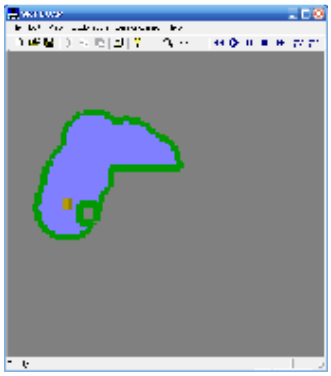
Tabla 5.11 Resultados de las estrategias de SRRT_LocalB_v2 con tiempos máximos y mínimos en el ambiente 7 con el sensor Radial.

5.12 Comportamiento SRRT_localB_v2 con sensado star.

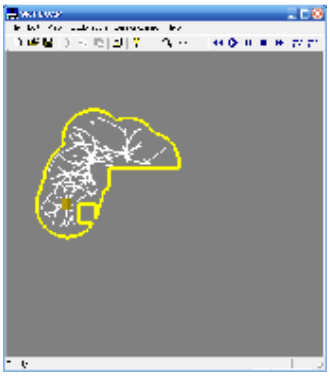
En las siguientes figuras mostramos algunos ejemplos de los árboles aleatorios de exploración obtenidos, junto con los caminos encontrados por el algoritmo SRRT_localB_v2 con la estrategia sensor_star.



Ambiente original

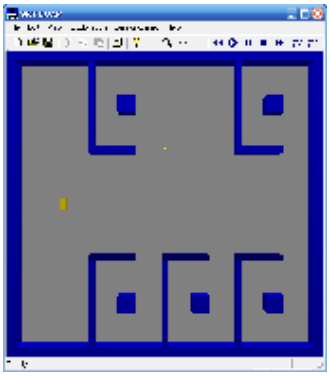


Mapa original

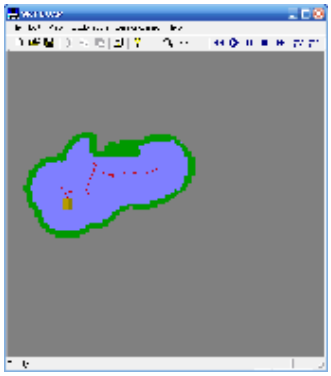


Árbol obtenido

Figura 5.21 Ambiente 7.



Ambiente original



Mapa obtenido

Árbol obtenido

Figura 5.22 Ambiente 8.

Ambiente 7	SRRT_LocalB_v2
Número de nodos máximo	1391
Número de nodos mínimo	686
Tiempo de planificación máximo	49.45 seg.
Tiempo de planificación mínimo	20.79 seg.

Tabla 5.12 Resultados de las estrategias de SRRT_LocalB_v2 con tiempos máximos y mínimos en el ambiente 7 con el sensor Star.

Capitulo VI.

Conclusiones.

Los algoritmos propuestos en esta tesis (tales como SRRT, SRT y sus extensiones) constituyen una alternativa a los métodos de planificación convencionales para determinados entornos complejos y robots con restricciones de no-holonomía, los requisitos computacionales hacen viable su implementación en tiempo real, incluso para una posible re-planificación en línea ante obstáculos no modelados. En este sentido la inclusión de las maniobras restringidas como herramientas en los algoritmos existentes puede suponer una mejora interesante.

Adaptamos al método SRT, la estrategia de percepción SRT_Radial y SRT_Star, para aprovechar todo lo posible la información de los sensores, que como se ha comprobado en las pruebas su eficacia varía de acuerdo a tipo de algoritmo que se utiliza por ejemplo cuando se aplica el SRT_Radial con el algoritmo de exploración Goal es más rápido en tiempo de ejecución por que logra encontrar más rápido su camino meta, en cambio si al utilizar ese mismo algoritmo en la otra percepción su tiempo de ejecución es más lenta por que hace exploración de toda el área que más pueda explorar. Aplicamos y adaptamos el robot móvil que tiene una restricción no-holonómica simple en el ángulo de conducción.

Se desarrollaron 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_Local_Bias, con dos variantes cada una de ellas, adaptan el planificador 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 libe 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 estrategia, guiados por una heurística sencilla sobre la distancia euclidiana que separa al estado candidato del estado meta.

En el árbol los nodos se distinguen por dos clases, nodos intermedios, aquellos donde se lleva a cabo un proceso de percepción del ambiente; y nodos terminales, que sirven para darle movilidad al robot que tiene una descripción de la región segura local y contiene la entrada de control.

Los resultados obtenidos a través de simuladores muestran que las estrategias 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.

Apéndice A.

Librería MFC.

La biblioteca de Microsoft Foundation Class (MFC Library) es una librería de clases C++ creada para ayudar en el desarrollo de aplicaciones de Microsoft Windows. Las MFC son utilizadas en aplicaciones GUI, también pueden ser utilizadas en cualquier tipo de aplicación. La biblioteca MFC consta de una serie de clases, que son ligeros 'envoltorios' de alto nivel para interfaces de programación de aplicaciones (API) como ODBC. Todo el kernel Win32, GDI y objetos de usuario se asocian a MFC.

Se le denomina biblioteca vertical, por que utiliza en gran medida la herencia de clases y en pequeña escala las plantillas C++. Las librerías pueden cargarse dinámicamente o estáticamente.

Todas las clases MFC tienen la letra mayúscula C de prefijo. Por ejemplo las usadas en este proyecto son: CString, CFile, que son declaradas en afxwin.h.

A.1. Descripción general.

Se puede aprovechar las características que tiene para derivar sus propias clases desde las clases del sistema, lo que ahorrara ingentes cantidades de tiempo por el hecho de no tener que crear sus propios objetos Wondows desde cero, sino aprovechar el código base de MFC y añadir la funcionalidad requerida.

CDialog	Clase base empleada para mostrar cuadros de diálogo.
CFileDialog	Incluye el cuadro de dialogo común en Windows para seleccionar un archivo.
CFrameWnd	Ventana marco SDI (un documento).
CView	Incluye una vista básica dentro de la aplicación básica document/view.
CClientDC	Construye un contexto de dispositivos que se asocia con el área cliente de una ventana.

Figura A.1 Clases de MFC.

MFC proporciona una cantidad de características [12] entre las que se incluyen:

Arquitectura *Document/view* (documento/vista).

Interfaz de documentos múltiples (MDI).

Uso y creación de controles ActiveX.

Soporte para bases de datos ODBC y Access.

Soporte multithread.

A.2 Jerarquía de MFC.

Proporciona un soporte general para archivos (que están directamente derivado de CObject), ventanas (cualquier elemento de la interfaz de usuario, desde un botón hasta cuadro de lista tiene asociado el manejo de ventanas), gráficos (dentro de eta categoría se hacen dos subdivisiones: 1. Representa un contenedor donde el gráfico o texto se dibuja, 2. Son dispositivos gráficos, un objeto que se emplea para realizar el dibujo), soporte para bases de datos (ODBC proporcionan acceso a muchas plataformas cliente-servidor de bases de datos), como se muestra en la figura A.1.

Apéndice B.

Librería MSL.

La librería Motion Strategy Library, MSL, desarrollada por Steven La Valle 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 probabilísticas (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 propósito independiente. La relación entre ellas se muestra en la figura B1. Este no es un diagrama jerárquico; solo muestra el flujo de la información a través de la jerarquía.

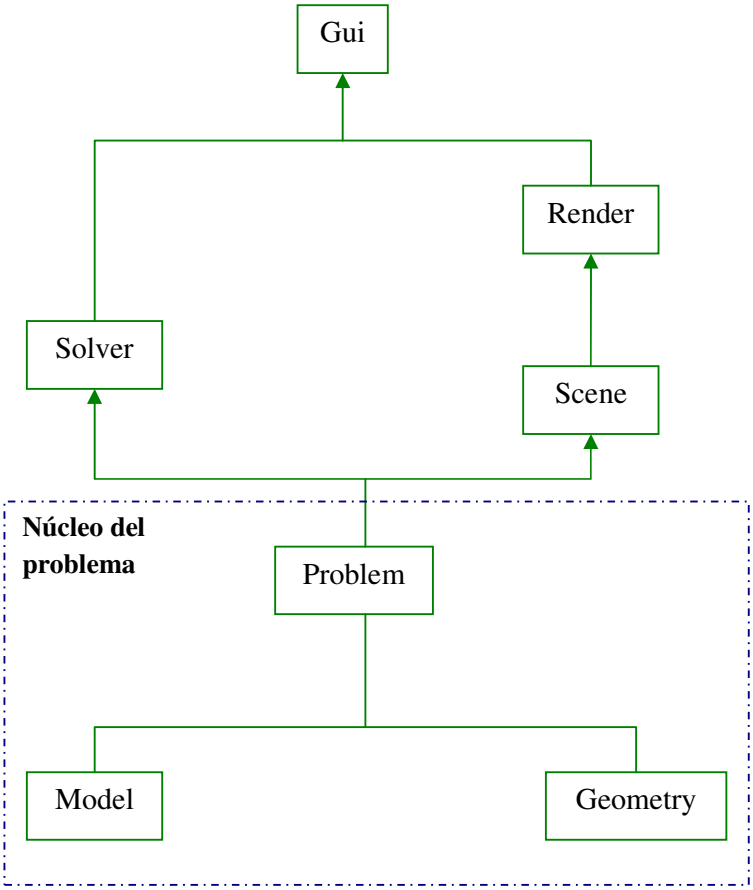


Figura B1: Clases que constituyen la librería MSL.

Cada una de las siete clases jerárquicas se explica 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, 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 interfaz 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 inicial 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 estrategia de movimientos que resuelva el problema.

Scene: Esta es una clase interfaz 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 renderzar las imágenes, como el punto de vista de la cámara.

Render: Esta jerarquía de clase contiene diferentes implementaciones de soluciones de renderizado gráfico. Por ejemplo, cuando una interfaz de usuario gráfica (GUI) solicita que el camino solución sea animado, un método en la 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 desarrollar SGI, IRIS, Open GL, Open Inventor. La flexibilidad proporcionada por estas clases permite fácilmente extensiones creadas por otras librerías gráficas y plataformas.

Gui: La interfaz gráfica de usuario (GUI) esta diseñada como una jerarquía de clases para generar interfaces de usuario especificas, diseñadas para una variedad de problemas de estrategia 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. Utiliza una seria de técnicas para acelerar el dibujo y el diseño espacial de la GUI. Permite construir fácilmente los controles y otros elementos GUI, mediante la adopción de los controles existentes y la creación de una clase derivada que añade o simplemente redefine el comportamiento deseado.

Proximity Query Package (PQP). Este paquete fue desarrollado por la universidad de Carolina del Norte, y es libre para uso no comercial. Realiza tres tipos de consultas de proximidad en un par de modelos geométricos compuestos de triángulos: (a) detección de colisiones. Detecta si los dos modelos se superponen y, opcionalmente, de todos los triángulos que se solapan. (b) calculo a distancia. Calcula la distancia mínima entre un par de modelos, es decir, la distancia más cercana entre el par de puntos. (c) la tolerancia de verificación. Determina si dos modelos están más cerca o más lejos que una distancia de tolerancia.

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 características del render que usa GL, más algunos otros. Se recomienda ya que en 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 objeto son modelos artísticos.

B.3. Crear un nuevo problema.

Cada problema se describe por un directorio completo de archivos. La mayoría 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 están 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 del modelo de movimientos es usado. Por ejemplo, Model2DRigid y Model3DRigidMulti.

GeomXXX: Un archivo nombrado exactamente después del modelo geométrico, tal como GeomP3DRigid para un cuerpo rígido hecho de triángulos en un mundo 3D (con el uso del paquete PQP para la detección 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.

Sensor: Una lista de los datos registrados por el sensor Star.

SensorRadial: Una lista de los datos registrados por el sensor Radial, por default es cargada.

Explora: Es una carpeta donde va estar el archivo Model2DRigidCar para darle al ambiente que se va a explorar.

Los siguientes archivos son opcionales:

ModelDeltaT: El incremento de tiempo a ser usado para la integración numérica de la ecuación de transición de estado (ecuaciones de movimiento).

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 nombre de archivos que corresponde a cuerpos móviles. Si *BodyList* no existe, entonces *Robot* es cargado.

LowerState: El vector estado con los valores más bajos posibles. Cada elemento es el valor más pequeño para una variable de estado.

UpperState: El vector estado con los valores más altos posibles. Cada elemento son el valor más grande para esa variable estado.

LowerWorld: El valor más pequeño en (x,y,z) del ambiente.

UpperWorls: El valor más grande en (x,y,z) del ambiente.

B.4. Jerarquía de clases.

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

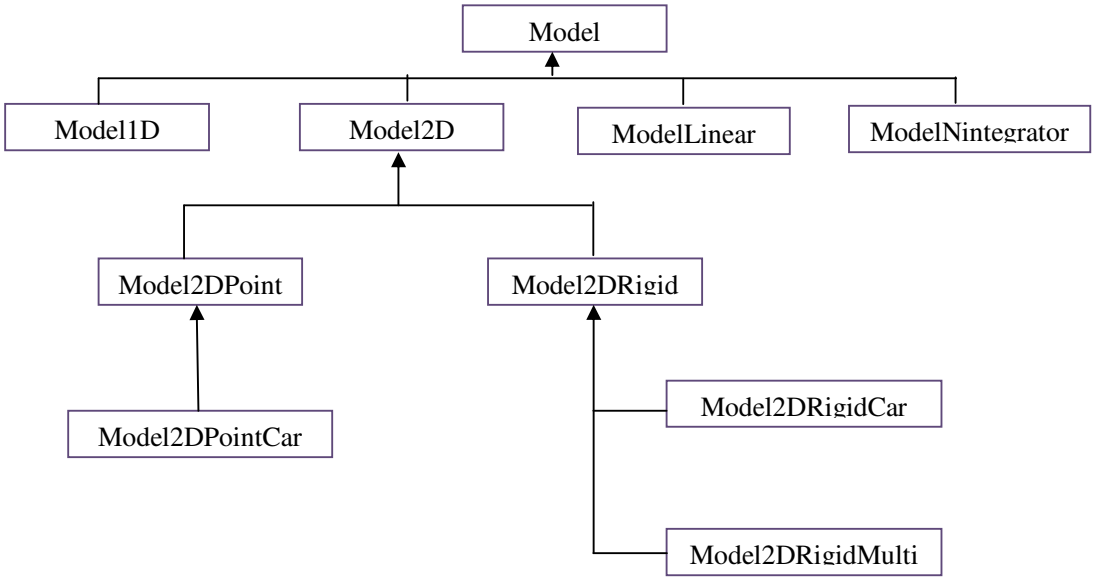


Figura B.2: Jerarquía clase Model.

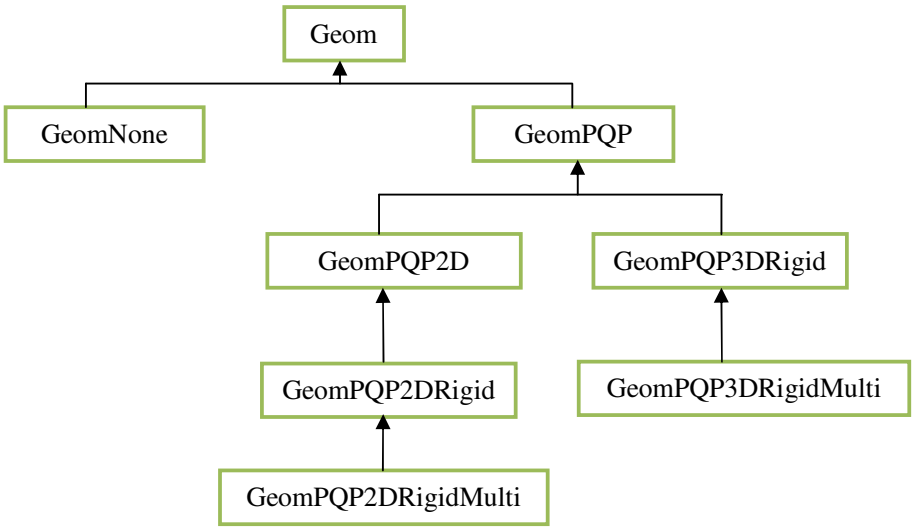


Figura B.3. Jerarquía clase Geom.

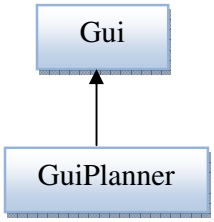


Figura B.4. Jerarquía clase Gui.

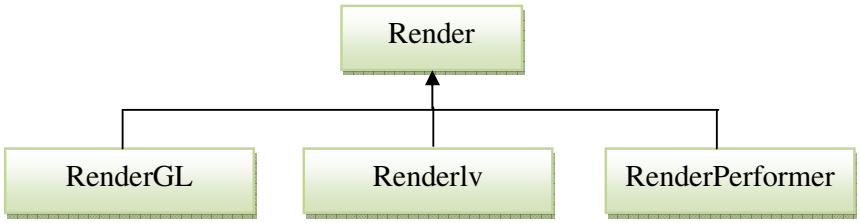


Figura B.5. Jerarquía clase Render.

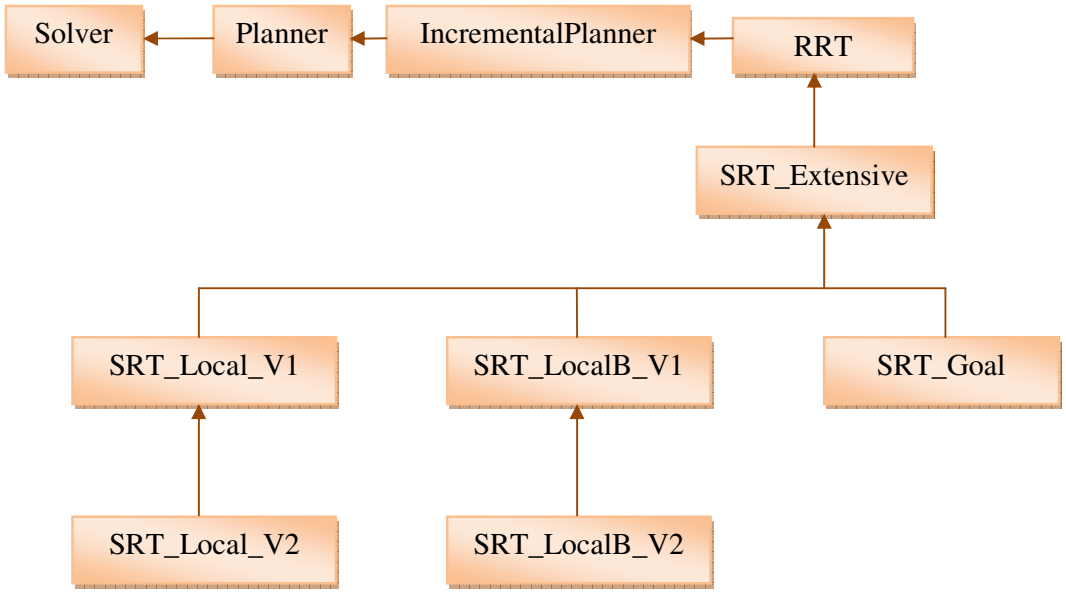


Figura B.6. Jerarquía clase Solver.

Apéndice C.

Librería GPC.

Tradicionalmente el recorte de polígonos se ha usado para recortar las porciones de un polígono que se encuentra 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 tiende 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 (polígono sujeto) como el polígono utilizado para el corte, (polígono de corte), pueden ser convexos o cóncavos, interceptarse a sí mismo, 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 dichas librerías 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).

C.1 Descripción.

Un polígono genérico (o un 'conjunto polígono') 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, cóncavo o interceptarse a si mismo. Los huecos internos pueden formarse debido a los contornos. Ver la figura C.1 donde se muestra un conjunto de polígonos constituido por cuatro contornos. A la izquierda tenemos un contorno cóncavo 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 si 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.

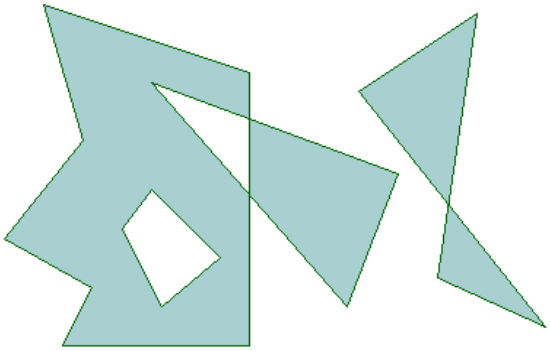
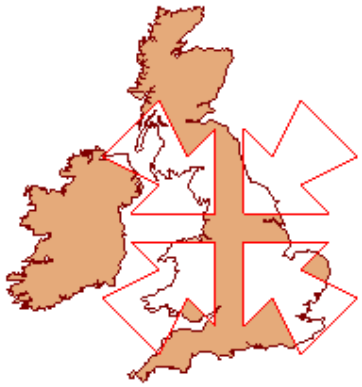
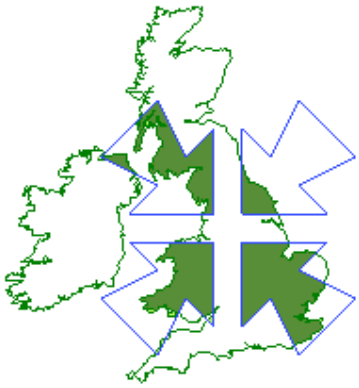


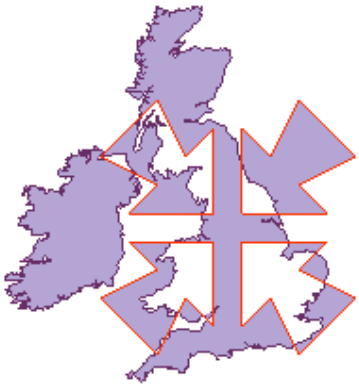
Figura C.1 polígono genérico con cuatro contornos.



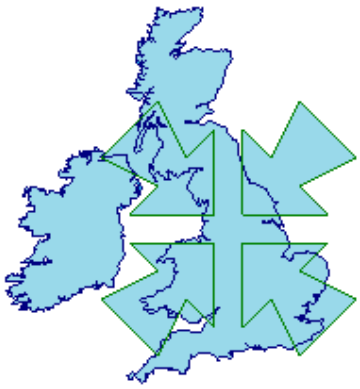
a) Diferencia.



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 están 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:

‹*numero-contornos*›

‹*numero-vertices-primer-contorno*›

[‹*bandera-huecos-primer-contorno*›]

‹*lista-vertice-primer-contorno*›

‹*numero-vertice-segundo-contorno*›

[‹*bandera-huecos-segundo-contorno*›]

‹*lista-vertice-segundo-contorno*›

..

.

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 cuadrilátero y su correspondiente archivo ASCII con las banderas de huecos incluidas.

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

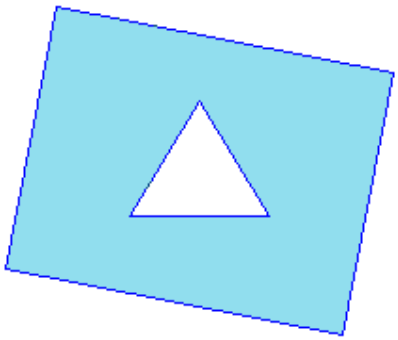


Figura C.3 polígono y su correspondiente archivo ascii.

La función `gpc_add_contour()` facilita la fusión de un nuevo contorno con un polígono existente. El parámetro final indica si el contorno sería considerado como hueco o no, los valores que utiliza son 1(true) o 0(false) 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_polygn *result_p);
```

Si el resultado se requiere en un tristrip se utilice 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 almacenara 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 si misma) 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 estado 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 interceptan.

La figura C.4 a) muestra ejemplos de cuadrados interceptándose, 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, pro 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), muestra figuras similares que se interceptan a si mismas, sin embargo, cada una crea dos contornos externos tocándose 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-esimo 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 encuentre 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 producirá un resultado nulo a lo largo del límite compartido con la operación de intersección. La precisión de los limites 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, puede resultar figuras poligonales incorrectas si se da a GPC_EPSILON un valor bastante grande.

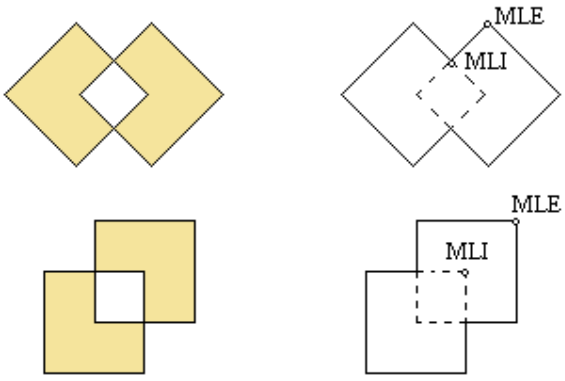


Figura C.4 a) Los contornos se descomponen en un contorno externo que contiene un hueco en su interior.

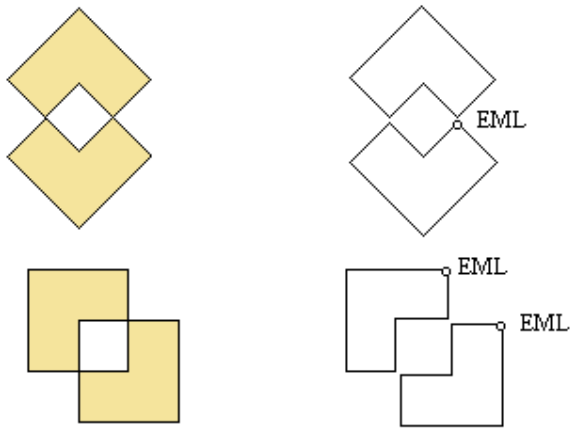


Figura C.4 b) Los contornos, sin embargo, se descomponen en dos contornos externos tocándose en el punto de intersección.

Figura C.4 Contornos que se interceptan a si mismos.

Bibliografía.

[1] S.M. La Valle. Rapidly-exploring random trees: A new tool for path planning.

Department of computer Science, lowa State University, 1998.

[2] S.M. La Valle and Z. S. Peng Cheng. RRT-based trajectory design for autonomous automobiles and spaceraft. *Archives of control sciences*, Vo. 11, No. 3, pp. 51-78, 2001.

[3] S.M. La Valle and J.J.Kuffner. RRT-connect: An efficient approach to single-query path planning. *Proceeding of the IEEE, International Conference on Robotics & Automation*, pp. 995 -1001, 2000.

[4] S. M. La Valle and J. J. Kuffner. Randomized kinodynamic planning. *Algorithmic and Computational Robotics:New Directions*, Vol. 20, No. 5 pp. 378-400, 2001.

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

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

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

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

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

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

[11] J. M. Ahuaczin 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.

[12] Richard C. Leinecker y Tom Archer. Visual C++ 6. 1999.

[13] Espinoza León Judith "Estrategias de exploración de ambientes desconocidos en robótica móvil" Tesis de maestría FCC-BUAP (2006).