Proyecto Fin de Grado
Ingeniería Industrial

Generación de caminos para un multi-rotor con brazo manipulador mediante el uso de planificadores RRT

Autor: Francisco José Márquez Bonilla

Directores: Aníbal Ollero Baturone, Iván Maza Alcañiz

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Grado
Ingeniería Industrial

Generación de caminos para un multi-rotor con brazo manipulador mediante el uso de planificadores RRT

Autor:
Francisco José Márquez Bonilla

Directores:
Aníbal Ollero Baturone, Iván Maza Alcañiz

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Grado: Generación de caminos para un multi-rotor con brazo manipulador mediante el uso de planificadores RRT

Autor: Francisco José Márquez Bonilla
Directores: Aníbal Ollero Baturone, Iván Maza Alcañiz

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Al autor le gustaría agradecer a Aníbal Ollero Baturone por haberle brindado la oportunidad de la realización de dicho proyecto y la confianza depositada, y por supuesto a Iván Maza Alcañiz por aconsejarle y guiarle durante la realización del mismo.

Índice Abreviado

Índice

Notación

$\mathbb{R}$	Cuerpo de los números reales
$\mathscr{C}$	Espacio de estados
$\emptyset$	Conjunto vacío
U	Campo potencial
U_{att}	Campo potencial atracción
U_{rep}	Campo potencial repulsión
ζ, η	Factores de escala
F	Fuerza
X, Y, Z	Ejes cartesianos
$\bigtriangledown \square$	Operador nabla
D_{obs}	Distancia a obstáculos
q	Estado del robot
Q	Vector de estados del robot
q_{rand}	Estado aleatorio
q_{init}	Estado inicial del árbol
q_{new}	Nuevo estado añadido al árbol
T	Árbol de estados generados por el algoritmo de planificación
V	Conjunto de vértices del árbol
E	Conjunto de segmentos que conectan los vértices de V
c_i	Coste del estado i
d_{ij}	Distancia estados i y j
$Temp$	Parámetro Temperatura
α	Parámetro variación de Temperatura
$nFail$	Número de fallos
K	Parámetro de normalización
ρ, δ	Parámetros umbrales
$Cost(\square)$	Función para el cálculo del coste de un estado
$Line(\square,\square)$	Función para el cálculo de la distancia entre dos estados

1 Introducción

1.1 Motivación

La robótica es un campo en constante crecimiento estos últimos años, convirtiéndose cada vez más en una parte significativa en nuestra vida diaria. Unmanned Aerial Vehicle (UAVs), robots de rescate, automóviles autopilotados, robots aspiradora o robots asistentes para la tercera edad son algunos de los muchos ejemplos que tenemos hoy en día.

La planificación de caminos es uno de los problemas fundamentales y, a su vez, de los más estudiados en robótica. El aumento del uso y la complejidad de las tareas de los robots autónomos implica la aparición de problemas de planificación cada vez más difíciles, robots con más grados de libertad, entornos más complejos, etc, lo que ha impulsado el desarrollo de nuevas estrategias de planificación que puedan abordar estos problemas de una manera eficiente. Este es el origen de los algoritmos de planificación de caminos probabilísticos, sampling-based en inglés, debido al gran potencial que ofrecen para resolver problemas que escapan al alcance de la mayoría de los métodos deterministas.

Hoy en día la planificación de caminos probabilística es un sector de investigación muy activo donde constatemente están surgiendo nuevos y más eficientes algoritmos para todo tipo de aplicaciones.

Para la realización de este trabajo se ha contado con el respaldado del Proyecto Aerial Robotics Cooperative Assembly System (ARCAS), financiado por la Comisión Europea bajo el programa FP7 ICT (ICT-2011-287617).

1.2 Estado del arte

Multitud de estrategias han surgido en la historia de la planificación de caminos hasta el día de hoy [5, 20]. Los algoritmos probabilistas, los cuales son un concepto relativamente nuevo en la planificación de caminos, han adquirido una gran importancia en los últimos años. A continuación veremos un resumen con algunos de los métodos más representativos de la planificación de caminos.

1.2.1 Bug Algorithm

Este tipo de algoritmo es uno de los primeros y más sencillos planificadores basados en sensores [19].

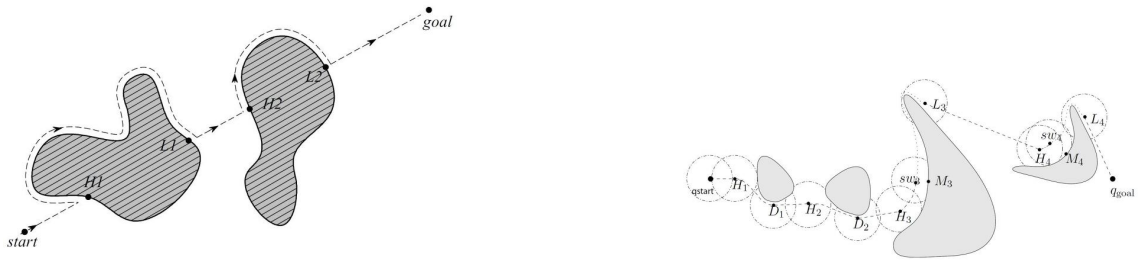
El robot se simplifica a un punto en el plano con un sensor de contacto. Si el sensor tiene un rango finito estamos hablando del algoritmo Tangent Bug, el cual usará la información del sensor para encontrar un camino más rápido. Podemos ver un ejemplo de ello en la Figura 1.1.

Los algoritmos basados en el método Bug son muy fáciles de implementar, aunque, aun así, su éxito está garantizado siempre que sea posible.

Dicho algoritmo está caracterizado por dos comportamientos:

- Movimiento hacia la meta.
- Seguimiento de fronteras.

A mayor rango disponga el sensor del robot, más óptimo será el camino obtenido.



(a) Algoritmo Bug2. **(b)** Algoritmo Tangent Bug.

Figura 1.1 Camino trazado por algoritmos basados en Bug.

1.2.2 Campos potenciales

Este método tiene como objetivo la evitación de obstáculos en robots móviles basándose en el concepto de campos potenciales artificiales [13]. Un campo potencial es una función diferenciable de valor real $U : \mathbb{R}^m \rightarrow \mathbb{R}$. El valor de dicha función potencial puede verse como energía, y el gradiente del potencial sería la fuerza.

La función potencial puede ser definida sobre el espacio libre como una suma de un potencial de atracción empujando el robot hacia la meta y un potencial de repulsión que mantiene al robot lejos de los obstáculos, como puede verse ilustrado en la Figura 1.2. El robot es representado como un punto dentro de dicho campo potencial.

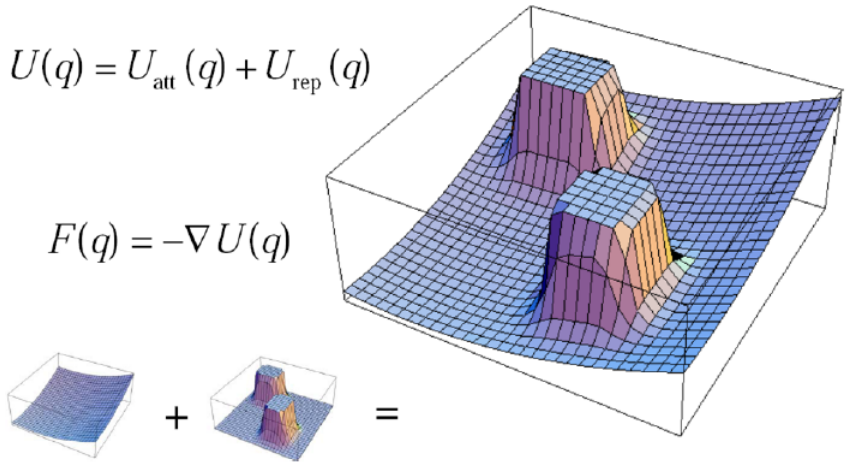


Figura 1.2 Representación de los campos potenciales artificiales.

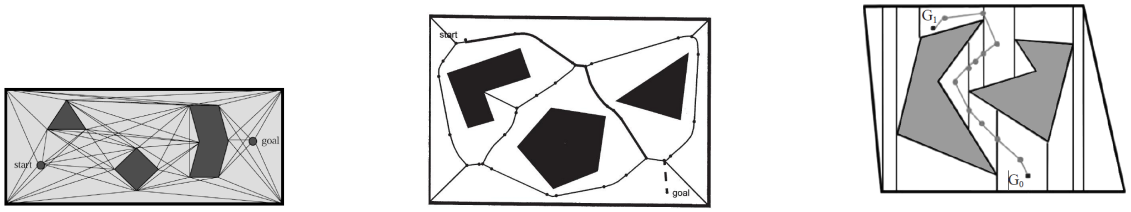
El valor de los campos atractivos, U_{att}, y repulsivos, U_{rep}, vienen expresados por las siguientes fórmulas.

$$U_{att}(q) = \frac{1}{2}\zeta d^2(q,q_{goal}),$$
$$U_{rep}(q) = \frac{1}{2}\eta d(\frac{1}{D_{obs}(q)} - \frac{1}{Q^*})^2$$

Los parámetros ζ y η son factores de escala, mientras que Q^* es una distancia constante para limitar el radio de acción de los campos potenciales de los obstáculos, a una distancia mayor de Q^* el camino del robot no se ve alterado por dicho campo, es decir,

$$U_{rep}(q) = 0 \quad si \quad D(q) > Q^*.$$

Este método es una buena alternativa para trabajar con altos grados de libertad debido a su bajo coste computacional [8], pero tiene la desventaja de quedarse atrapado en mínimos locales del campo potencial.



(a) Grafo de visibilidad. **(b)** Diagrama de Voronoi. **(c)** Descomposición en celdas.

Figura 1.3 Diferentes métodos de búsqueda de grafos.

1.2.3 Mapas de carretera

Los algoritmos de mapas de carreteras, Roadmaps, modelan el espacio de trabajo como una red de líneas conectadas. Esto permite reducir un espacio de configuración N-dimensional a una búsqueda de caminos en 1 dimensión, transformándose el problema de planificación de caminos a un problema de búsqueda de caminos en grafos.

Existen multitud de técnicas basadas en búsqueda de grafos, en la Figura 1.3 podemos ver algunas de las más conocidas.

- Gráficos de visibilidad.
- Diagramas de Voronoi.
- Descomposición en celdas.

Los grafos de visibilidad se construyen conectando los vértices de los obstáculos del espacio de trabajo. Esto genera caminos óptimos en cuanto a la longitud del camino pero con el inconveniente de la proximidad a obstáculos.

Una alternativa que busca el camino con mayor claridad de obstáculos son los diagramas de Voronoi. Las redes de Voronoi dividen el espacio con lineas equidistantes a los obstáculos. La obtención de dichos diagramas se dificulta con la presencia de obstáculos que no tengan una geometría simple.

La descomposición en celdas divide el espacio en regiones trapezoidales y triangulares, las cuales se usarán para crear el grafo. Este método no es óptimo debido a que el problema de encontrar el mínimo número de celdas es NP-completo. Existen también algoritmos como Quadtree, Figura 1.4, y Octotree en el caso de 3 dimensiones, que consisten en descomposición aproximada de celdas, en los cuales las celdas en la frontera entre obstáculo y espacio libe se van descomponiendo sucesivamente en celdas más pequeñas para crear el grafo.

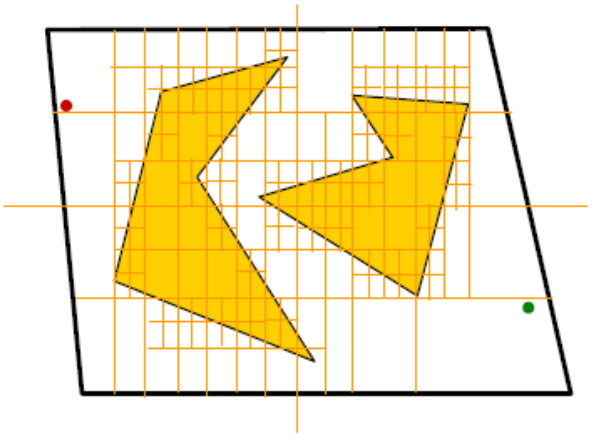


Figura 1.4 Método descomposición en celdas Quadtree.

Una vez construido el grafo, para encontrar una solución óptima se utiliza algoritmos como Dijkstra [4] y A* [6] o algoritmos como D* [22] y AD* [18] para grafos dinámicos, esto quiere decir que los parámetros

pueden cambiar durante la resolución del problema. El uso de los métodos de búsqueda de grafos implica una discretización del espacio de trabajo, aumentando el coste computacional considerablemente en problemas con muchas dimensiones.

1.2.4 Planificadores probabilísticos

A diferencia de los métodos anteriores, los planificadores de camino probabilísticos se basan en la generación aleatoria de los estados para construir el camino, mediante el muestreo del espacio de configuración, $\mathscr{C}$. Esto permite a dichos planificadores abordar problemas complejos con muchos grados de libertad manteniendo tiempos de cálculo bajos [5].

La idea fundamental de los planificadores probabilísticos es la de aproximar la conectividad del espacio de configuraciones mediante una estructura de grafos. El espacio a explorar es muestreado, generando estados que formarán los vértices de dicho grafo. Las líneas que unen los vértices del grafo denotan segmentos de camino válidos.

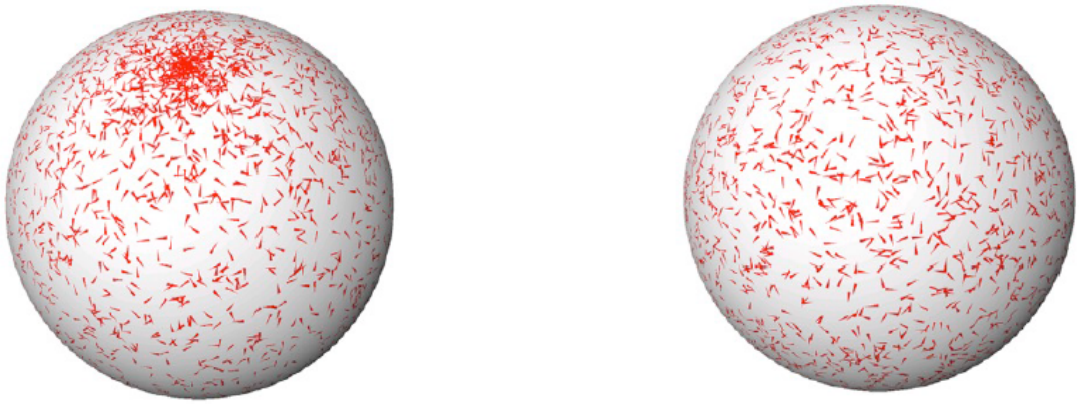
Los planificadores basados en muestreo no garantizan encontrar una solución, si es que existe. Esta es una propiedad que es referida como exhaustividad. Los planificadores probabilísticos aseguran una noción más débil, exhaustividad probabilística, esto quiere decir que la solución será encontrada, de existir, dando el tiempo de ejecución suficiente al algoritmo, el cual puede ser infinito.

Algunos de los algoritmos que están a la cabeza de los planificadores probabilísticos [23] son:

- **The Rapidly-Exploring Random Tree (RRT)**, es un planificador basado en árbol que usa la siguiente idea: el algoritmo RRT genera estados aleatorios, q_{rand}, en el espacio de estado, $\mathscr{C}$, y, tras ello, encuentra el estado del árbol más cercano, $q_{nearest}$, al generado aleatoriamente y expande el árbol desde q_c hacia q_{rand}, hasta que el estado q_{new} es alcanzado, siendo dicho estado q_{new} añadido al árbol de exploración [17].
- **Probabilistic Roadmap Method (PRM)**, es un planificador que construye un mapa de carretera generando los estados de forma aleatoria, reduciendo considerablemente el tiempo de cálculo. El movimiento que conecta 2 estados dados se traduce a una búsqueda discreta (por ejemplo usando A* o Dijkstra) en el mapa de carreteras [12].
- **Expansive Space Trees (EST)**, es un planificador con estructura de árbol el cual trata de detectar regiones menos exploradas descomponiendo el espacio en áreas de visibilidad. El planificador construye 2 árboles, uno desde la configuración inicial y el otro desde la configuración final, los cuales irán creciendo hasta que la región de visibilidad de un árbol intersecte con la del otro [7].
- **Single-query Bi-directional Lazy collision checking planner (SBL)**, Este planificador es una versión bidireccional de EST con una estrategia de comprobación de colisiones "lazy", esto significa que se pospone la comprobación de colisiones hasta que la obtención de un camino para comprobar si es válido [21].
- **Kinematic Planning by Interior-Exterior Cell Exploration (KPIECE)**, es un planificador con estructura de árbol que usa una discretización basada en celdas a múltiples niveles para estimar la cobertura del espacio de estado. La estimación de la cobertura ayuda al planificador a detectar menos exploradas áreas del espacio de estado. Está específicamente diseñado para sistemas con dinámicas complejas, donde es necesario una simulación basada en la física [24].
- **Path-Directed Subdivision Trees (PDST)**, es un planificador de movimiento basado en árbol que intenta detectar la zona menos explorada del espacio a través del uso de una partición de espacio binario de una proyección del espacio de estado. La exploración está sesgada hacia células grandes con pocos segmentos de trazado. A diferencia de la mayoría de los planificadores basados en árboles que se expanden desde el punto final de un segmento aleatoriamente seleccionado del camino, PDST se expande desde un punto seleccionado al azar a lo largo de un segmento del camino seleccionado de forma determinista [15].

Los planificadores probabilísticos están constituidos por los siguientes elementos:

- **Muestreo:** este procedimiento se utiliza para seleccionar una configuración, al azar o cuasi-azar, y agregarlo al árbol o mapa de carreteras. Tales muestras pueden estar en el espacio de configuración libre de obstáculos o no. Se puede considerar como el núcleo de la planificación y la principal ventaja de los planificadores basados en probabilidad sobre otras técnicas. Es muy importante el uso de un



(a) Incorrecto. **(b)** Correcto.

Figura 1.5 Generadores de estados aleatorios.

generador de estados que asegure cierta uniformidad, en caso contrario la eficiencia del algoritmo se verá degradada como puede observarse en la Figura 1.5.

- **Métrica:** dadas dos configuraciones q_a y q_b, este procedimiento devuelve un valor o coste, que simboliza el esfuerzo que se requiere para llegar a q_b desde q_a. Es importante que sea una representación fiel del esfuerzo, en caso contrario las soluciones obtenidas serán altamente subóptimas.
- **Vecino más cercano:** es un algoritmo de búsqueda que devuelve el nodo más cercano a un estado dado. Para ello, se utiliza el valor obtenido en la función de métrica predefinida.
- **Selección de padre:** este procedimiento selecciona un nodo existente para conectarse al nodo recién generado. Dicho nodo existente se considera padre. RRT, por ejemplo, selecciona el nodo más cercano como padre. PRM conecta la muestra a varios nodos dentro de su vecindad. Por otro lado, EST selecciona un nodo padre para extender al azar basándose en sus nodos vecinos.
- **Planificación local:** dadas dos configuraciones q_a y q_b, este procedimiento intenta establecer un camino entre ellos. Aunque es intuitivo emplear caminos en línea recta, para la mayoría de sistemas robóticos este no es un plan factible debido a restricciones cinemáticas o dinámicas.
- **Comprobador de Colisiones (CC):** es mayormente una función booleana que devuelve el éxito o el fracaso de la conexión entre dos configuraciones. Una conexión es correcta cuando está libre de obstáculos.

1.3 Objetivos

Para la realización de dicho trabajo dividiremos el problema en dos partes: el cuerpo del multi-rotor y el brazo manipulador. Esta medida facilitará notablemente los cálculos al dividir el número de grados de libertad en dos grupos. Además de esto, el trabajo se dividirá en dos fases: OMPL y MoveIt!.

La primera fase nos dará la oportunidad de tener una toma de contacto con las distintas versiones del algoritmo RRT que veremos a continuación. Podremos comprobar como funcionan y modificarlos a voluntad para que encajen con nuestros requisitos de diseño.

La segunda fase, MoveIt!, nos permitirá conectar los RRT procedentes de OMPL con una plataforma multifuncional en la que podremos aprovechar las funcionalidades de ROS, como la ejecución de simulaciones en Rviz, emplear modelos usando URDF y, especialmente, incluir el concepto de articulaciones lo cual es imprescindible para nuestro diseño. Todos estos conceptos serán explicados en los siguientes capítulos.

2 Planificación para el modelo del multi-rotor con modelos cinemáticos

2.1 OMPL

La librería OMPL [23] es una implementación de muchos de los algoritmos de planificación de caminos que componen el estado del arte de éste ámbito. OMPL por sí mismo no contiene ninguna de las partes adyacentes al planificador como puede ser el CC, la visualización de resultados, etc. Esto es debido a que OMPL no quiere atarse a ningún CC o visualizador concreto, es una librería pensada para ser lo más general y abierta posible.

Existe una interfaz gráfica, OMPL.app, que proporciona una plataforma fácil e intuitiva para la representación de los caminos obtenidos con cuerpos rígidos y algunos tipos de vehículos. Dicha interfaz utiliza la librería Open Asset Import Library (Assimp) para cargar modelos gráficos del robot y del entorno en formato COLLADA.

Para la correcta ejecución de los planificadores de caminos, OMPL está compuesto de una serie de partes que explicaremos brevemente a continuación:

La pieza central está compuesta por el InformationSpace o Espacio de Información que contiene toda la información relacionada con el estado del robot y el entorno. Cada nuevo estado que se genera es validado por un comprobador de colisiones que comprobará tanto los nuevos estados, "StateValidityChecker", como la transición entre estados, "Motion Validator". Los diferentes estados que conformarán el camino obtenido en el RRT pueden ser generados de dos formas distintas dependiendo de si las restricciones del modelo son diferenciales o geométricas. La diferencia radica en que mientras que con restricciones geométricas se utiliza un generador de estados o "State Sampler" para crear los estados que serán comprobados en el Espacio de información. Restricciones diferenciales necesitarán la creación de estados de control, esto quiere decir que se generará las variables de control del modelo, por ejemplo, se determinarán las velocidades y aceleraciones, que, en el caso de que respeten las restricciones establecidas, se usarán para generar el próximo estado del robot, que será aceptado si a su vez es validado por el CC. Por último, deberemos indicar un punto de partida y de meta para completar el "ProblemDefinition", seleccionando el "Planner" que vamos a usar en nuestra prueba. Todas estas clases pueden configurarse por separado o usando "SimpleSetup", una clase que encapsula a todas las anteriores con el objetivo de agilizar la labor de programación.

Al ser uno de los objetivos de OMPL tratar de ser tan general y polivalente como sea posible, muchas de las clases mencionadas, como se indica en el diagrama, deben ser implementadas por el usuario. Para facilitar esta tarea, existe una aplicación, OMPL.app, que incluye varias opciones predefinidas como el comprobador de colisiones, en concreto Flexible Collision Library (FCL) y Proximity Query Package (PQP), generadores aleatorios de estados y una librería gráfica, Assimp, con el objetivo de poder cargar diseños simples y poder visualizar los resultados del planificador de caminos.

Para la elaboración de las simulaciones usaremos dicha interfaz gráfica para mostrar la solución de los problemas que se han implementado.

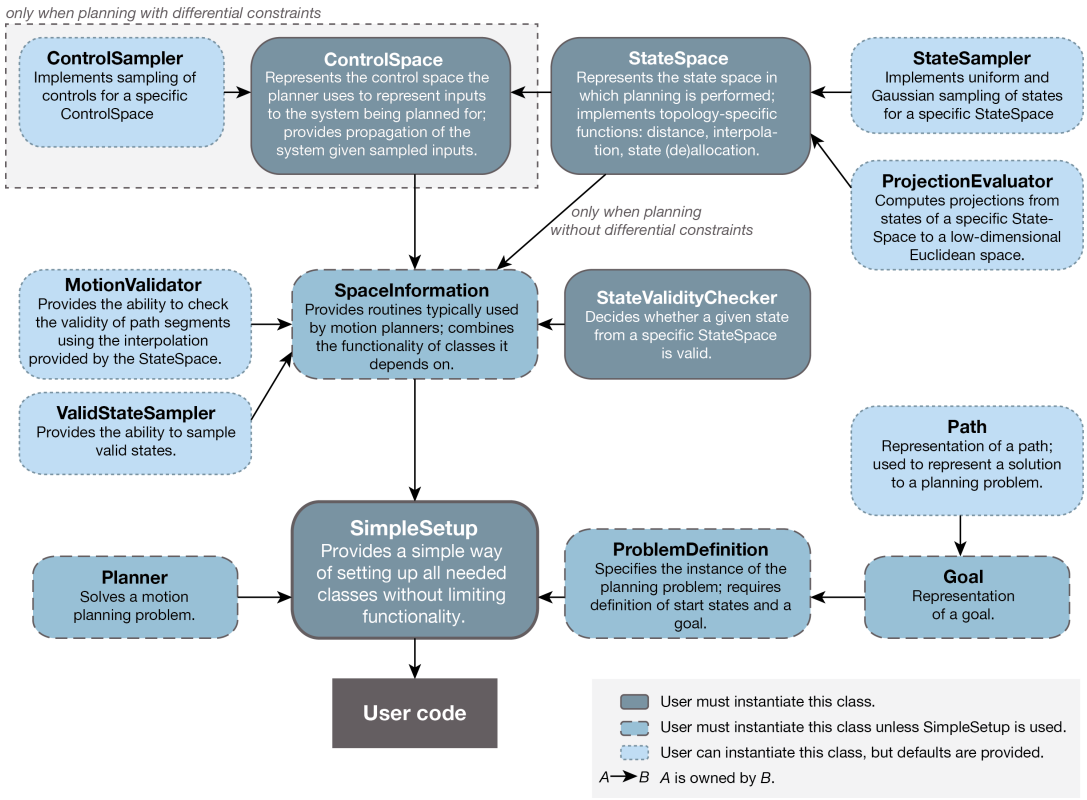


Figura 2.1 API de OMPL.

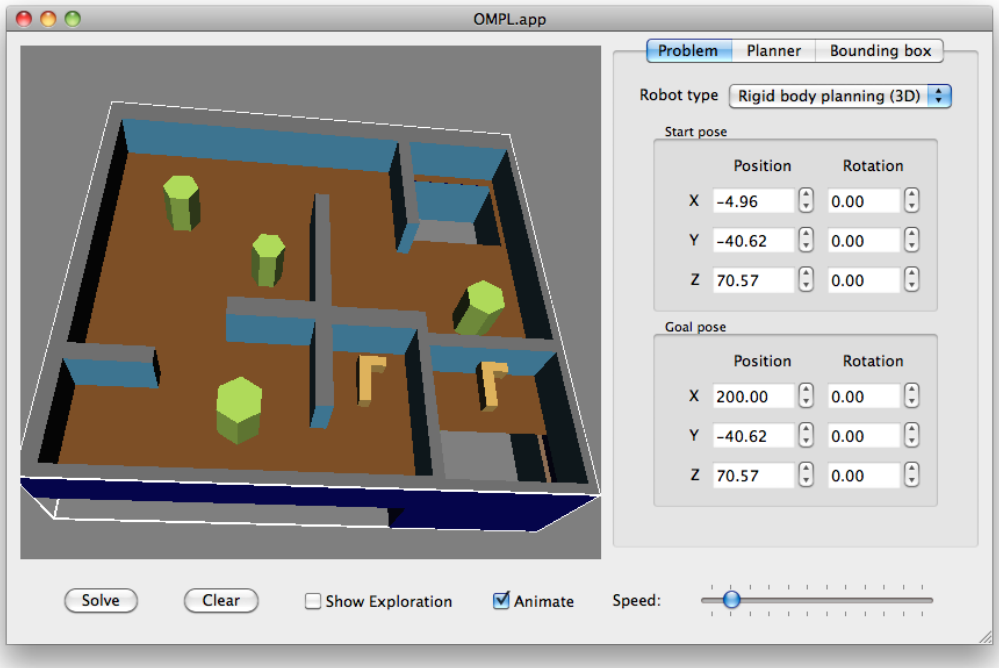


Figura 2.2 Interfaz gráfica OMPL.

(a) Multi-rotor. **(b)** Entorno CATEC.

Figura 2.3 Modelos gráficos usados en OMPL.

2.2 Modelo cinemático

Para las primeras pruebas se ha usado como modelo gráfico un cuerpo simple, una esfera, y un escenario predefinido en la interfaz gráfica de OMPL.

Tras ello, se ha implementado el modelo de un multi-rotor genérico, y un escenario del Centro Avanzado de Tecnologías Aeroespaciales (CATEC), Figura 2.3. De esta forma podemos ilustrar el problema de manera más realista y, a su vez, podemos comprobar que el multi-rotor efectivamente respeta las restricciones geométricas durante el camino generado.

2.3 Algoritmos

A continuación se describirá el algoritmo básico de un RRT [17] y el algoritmo RRT-CONNECT [14], el cual usaremos en las simulaciones en OMPL.

Tal como hemos comentado en el capítulo anterior, el algoritmo del RRT, Algorithm 1 y Algorithm 2, consiste en la creación de un árbol siguiendo el procedimiento de la Figura 2.4.

Algorithm 1 Algoritmo RRT

1: $T.init(q_{init})$;
2: **for** $k = 1$ **to** K **do**
3: $\quad q_{rand} \longleftarrow RANDOM\_CONFIG()$;
4: $\quad EXTEND(T, Q_{rand})$;
5: **end for**
6: Return T;

Algorithm 2 Función EXTEND

1: $q_{rand} \longleftarrow NEAREST\_NEIGHBOR(q,T)$;
2: **if** ($NEW\_CONFIG(q, q_{near}, q_{new})$) **then**
3: $\quad T.add\_vertex(q_{new})$;
4: $\quad T.add\_edge(q_{near}, q_{new})$;
5: $\quad$ **if** ($q_{new} = q$) **then**
6: $\quad\quad$ Return *Alcanzado*;
7: $\quad$ **else**
8: $\quad\quad$ Return *Adelantado*;
9: $\quad$ **end if**
10: **end if**
11: Return *Atrapado*;

El algoritmo de RRTConnect, Algorithm 3, incluye una serie de modificaciones que lo hace especialmente interesante. Es una variante bidireccional del RRT, esto quiere decir que se construyen dos árboles, uno desde el punto de inicio y el otro desde la meta. Esto puede ser muy útil en el caso que tengamos, por ejemplo, un problema tipo trampa de insectos [16], bugtrap en inglés, como ilustra la Figura 2.5.

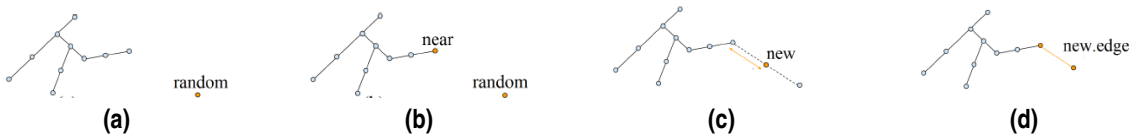


Figura 2.4 Generación de un nuevo estado en un RRT.

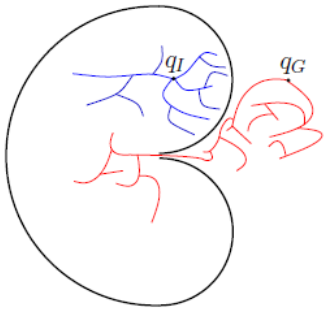


Figura 2.5 Bugtrap.

Además de esto, como podemos ver en Algorithm 4, el RRTConnect extiende de manera iterativa cada nuevo nodo hasta alcanzar un obstáculo, Figura 2.6, o alcanzar el estado generado aleatoriamente q_{rand}.

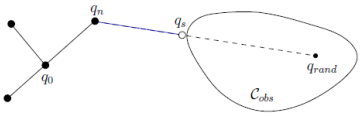


Figura 2.6 Nuevo estado q_s obtenido con la función CONNECT.

Algorithm 3 Algoritmo RRT-CONNECT

```
1: T_a.init(q_init); T_b.init(q_goal);
2: for k = 1 to K do
3:     q_rand ⟵ RANDOM_CONFIG();
4:     if not ( EXTEND(T_a, q_rand)) = Atrapado ) then
5:         if ( CONNECT(T_b, q_new) = Alcanzado ) then
6:             Return PATH(T_a, T_b);
7:         end if
8:     end if
9:     SWAP(T_a, T_b);
10: end for
11: Return Fracaso;
```

Otros añadidos que suelen tener los algoritmos RRT son el suavizado de los caminos obtenidos para mejorar la calidad del mismo o la utilización de un sesgo hacia la meta durante la planificación, goal-biased en inglés, que sustituye el estado aleatorio q_{rand} por el estado final un porcentaje determinado por el valor que asignemos a dicho sesgo. Esto último permite al algoritmo expandirse en mayor medida hacia la meta incrementando la velocidad de obtención del camino. Ambos métodos deben ser configurados con cuidado porque, el suavizado puede provocar colisiones que no habían inicialmente, y, si el valor de goal-biased es muy elevado, puede llevar al algoritmo a caer en mínimos locales, como se muestra en la Figura 2.7. También existen versiones del algoritmo RRT que limitan el uso del CC al camino completo una vez generado, como es el caso de los LazyRRT.

Algorithm 4 Función CONNECT

1: **repeat**
2: $S \longleftarrow EXTEND(T,q)$;
3: **until not** $(S = Avanzado)$
4: Return *Fracaso*;

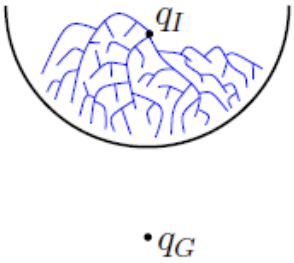


Figura 2.7 RRT atrapado en mínimo local.

2.4 Resultados

Para el estudio de los resultados de dichos algoritmos se ha usado una herramienta de evaluación comparativa, benchmarking en inglés, empleando el modelo del entorno del CATEC y el modelo de multi-rotor mencionado anteriormente tal y como se aprecia en la Figura 2.8. De esta forma podremos evaluar el camino desarrollado usando distintas versiones del algoritmo RRT y, a su vez, comprobar el efecto producido por la modificación de los parámetros de dichos algoritmos.

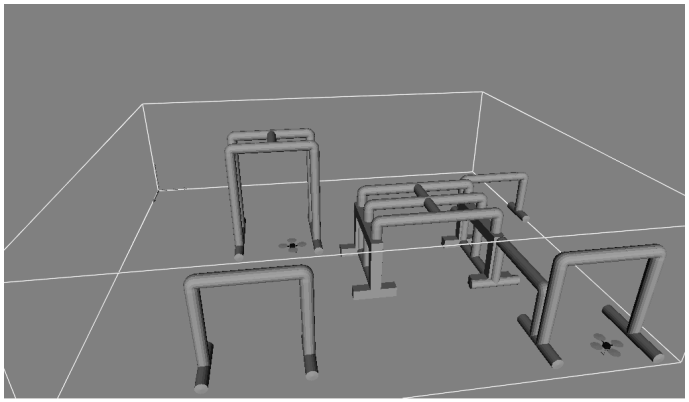


Figura 2.8 Escenario de pruebas en OMPL.

En primer lugar se han comparado los algoritmos RRT, RRTConnect y LazyRRT con sus valores por defecto. Los datos más relevantes se encuentran en la Figura 2.10.

Como puede apreciarse, el algoritmo RRTConnect se caracteriza por la rapidez en alcanzar un resultado aunque esto ocurre en detrimento de la suavidad del camino obtenido, especialmente en el punto de unión de ambos árboles de estados, como se puede apreciar en la Figura 2.9, por lo que un buen algoritmo de suavizado es necesario con este tipo de planificadores. El algoritmo LazyRRT no parece presentar una gran diferencia en los resultados frente al RRT original. Según OMPL, la version "Lazy" del algoritmo RRT no ha aportado mejoras representativas en ninguna clase de problemas.

A continuación se evalúa el efecto de los parámetros más importantes en los RRT:

- Rango, en la Figura 2.11.
- Sesgo de Meta, en la Figura 2.13.

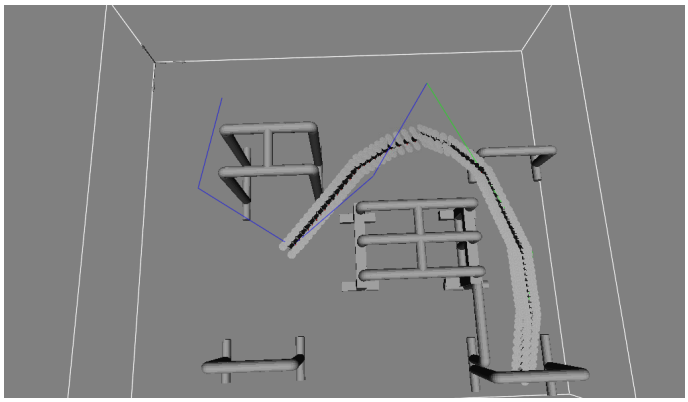
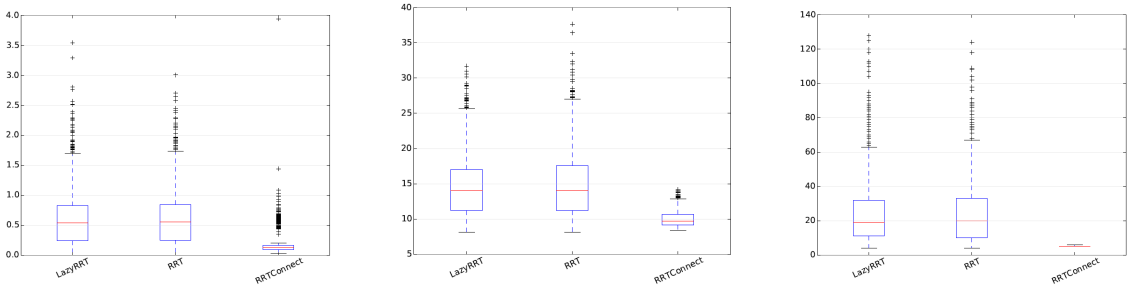


Figura 2.9 Camino obtenido con algoritmo RRTConnect.



(a) Suavidad de la solución. **(b)** Longitud de la solución. **(c)** Número de estados del árbol generado.

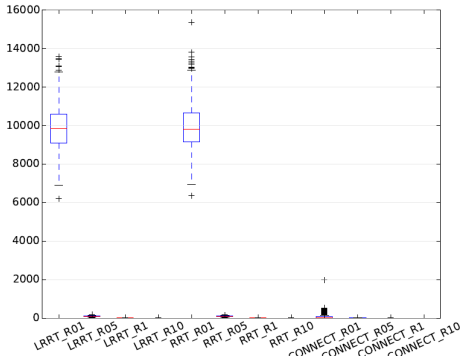
Figura 2.10 Comparación RRT geométricos.

El rango equivale a la máxima distancia entre dos estados conectados del árbol, mientras que el sesgo de meta, como ya se ha explicado, es el tanto por ciento de estados aleatorios que se sustituyen por el estado de meta.

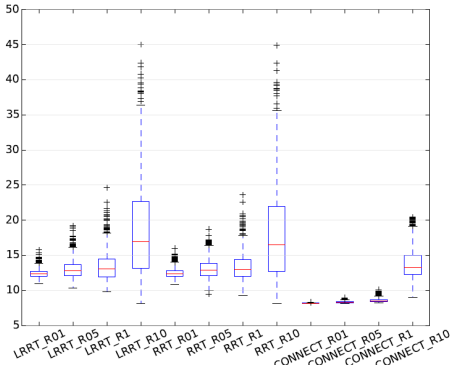
Respecto a los resultados obtenidos modificando el parámetro rango, podemos observar en la Figura 2.11 que a menor rango mejoramos la suavidad y minimizamos la longitud del camino obtenido, un ejemplo representativo de ello podemos encontrarlo en la Figura 2.12. Sin embargo, esto también produce un aumento en el tiempo de cómputo necesario para calcular el camino, por lo que se deberá buscar un equilibrio dependiendo de la prioridad de cada característica.

El sesgo de meta tiene una utilidad bastante evidente, como puede observarse en la Figura 2.13 y en la Figura 2.14. Aunque existe una gran diferencia entre usar y no usar el sesgo de meta, la velocidad de la solución varía poco usando diferentes valores en el problema presentado. Cabe destacar que este valor debería limitarse alrededor del 10 % debido al riesgo de no cubrir suficiente parte del mapa y quedar atrapado. Como aclaración, el valor de 5 segundos en el tiempo de planificación cuando el sesgo de meta es igual a cero en la Figura 2.13 es debido a que este es el tiempo límite de la simulación, lo que indica que al menos una gran parte de las pruebas durante la evaluación comparativa no han aportado soluciones válidas agotando los 5 segundos de tiempo límite.

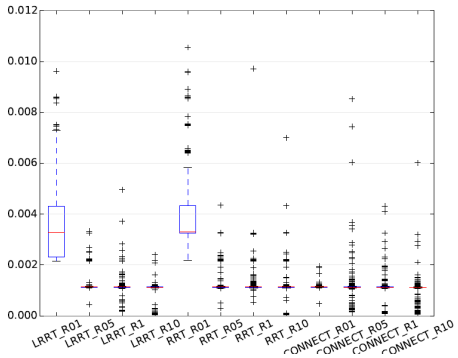
Por último, se ha realizado en la Figura 2.15 una comparación con algunos de los planificadores geométricos más usados en OMPL para tener una perspectiva más amplia de los resultados obtenidos con los algoritmos RRT.



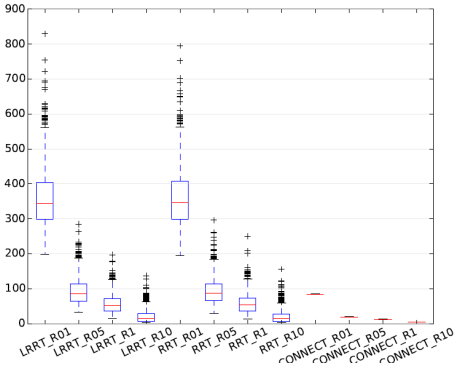
(a) Suavidad de la solución.



(b) Longitud de la solución.

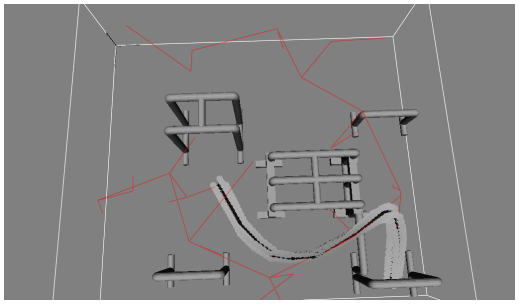


(c) Tiempo requerido.

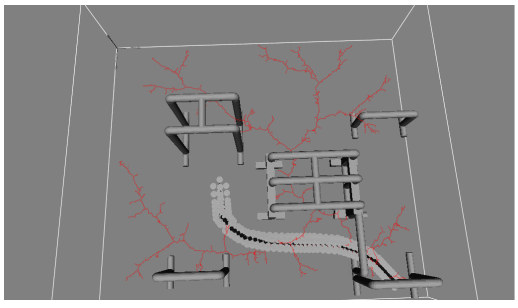


(d) Número de estados.

Figura 2.11 Comparación del parámetro Rango en RRT geométricos.

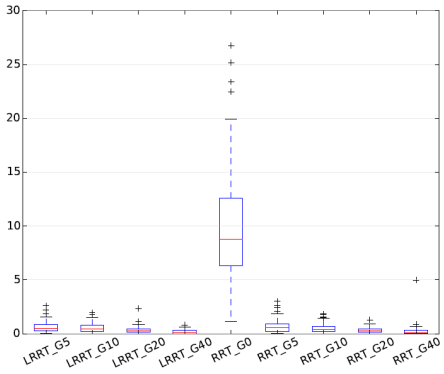


(a) Algoritmo RRT con $Rango = 1$.

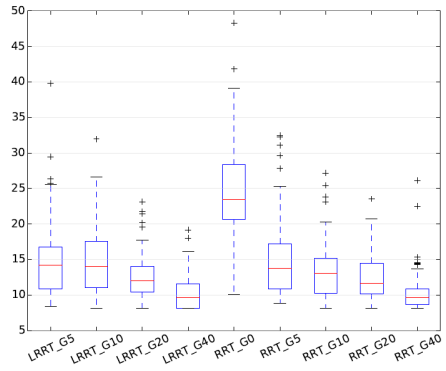


(b) Algoritmo RRT con $Rango = 0.1$.

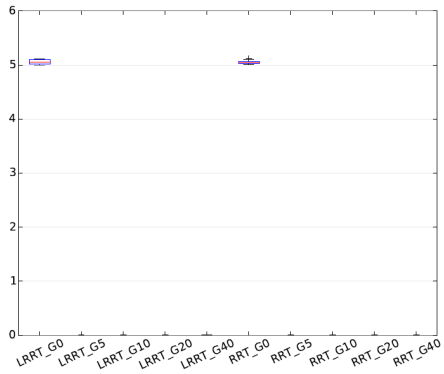
Figura 2.12 Comparación gráfica entre distintos valores del Rango en el algoritmo RRT.



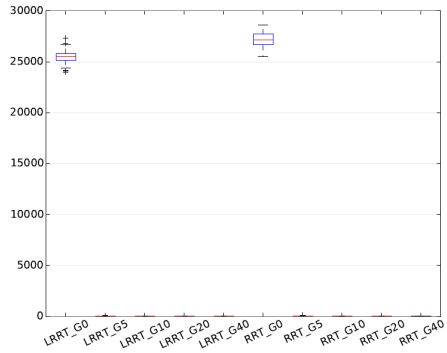
(a) Suavidad de la solución.



(b) Longitud de la solución.



(c) Tiempo requerido.



(d) Número de estados.

Figura 2.13 Comparación del parámetro Sesgo de meta en RRT geométricos.

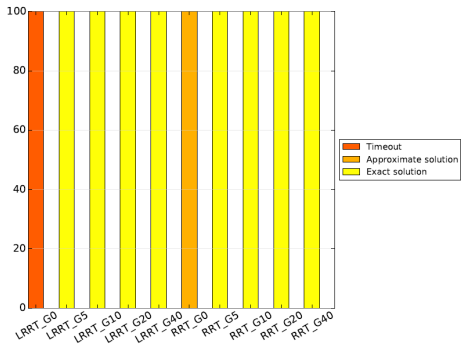
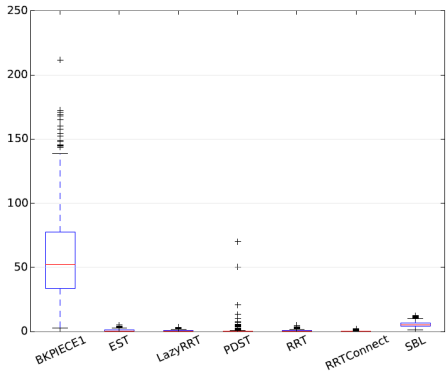
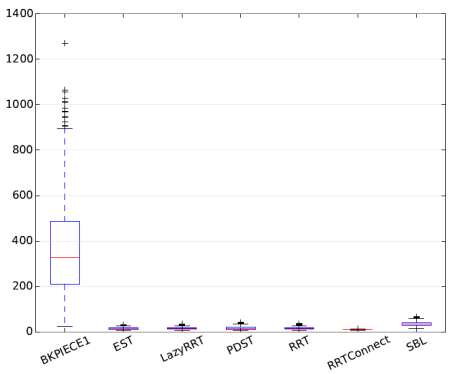


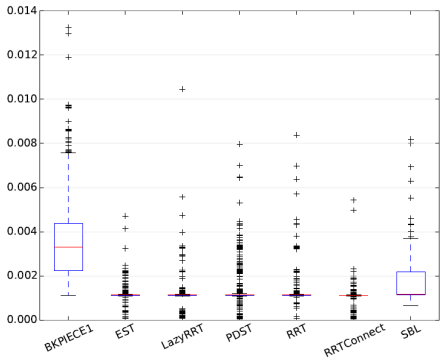
Figura 2.14 Estado de la solución variando el parámetro Sesgo de meta.



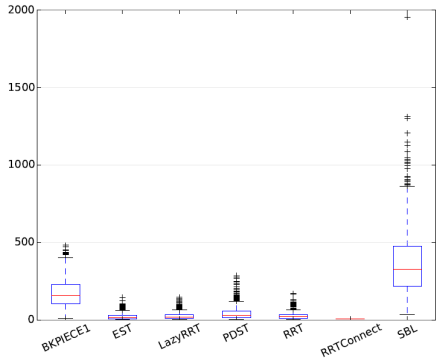
(a) Suavidad de la solución.



(b) Longitud de la solución.



(c) Tiempo requerido.



(d) Número de estados.

Figura 2.15 Comparación del parámetro Sesgo de meta en RRT geométricos.

3 Planificación para el modelo del multi-rotor con funciones de coste

3.1 Introducción

Como hemos podido observar en el capítulo anterior, los algoritmos RRT pueden reportarnos una solución a un problema complejo a un coste computacional relativamente bajo, pero dicha solución puede estar bastante alejada de lo que sería la mejor solución al problema. En muchas aplicaciones, obtener una de las muchas posibles soluciones de un problema no es suficiente. Pueden haber muchos factores, como la distancia a los obstáculos circundantes, la longitud del camino a recorrer, las áreas de cobertura, etc., que, de no ser tenidos en cuenta en la generación del camino, pueden desembocar en la obtención de soluciones poco eficientes o incluso no deseadas.

Para este tipo de problemas deberemos de tener en cuenta un objetivo de planificación en el generador de caminos para obtener una solución lo más óptima posible. El camino óptimo estará determinado por una función de coste que traducirá el objetivo que se desea maximizar en un mapa de costes en el espacio de configuraciones, tal como se ilustra en la Figura 3.1.

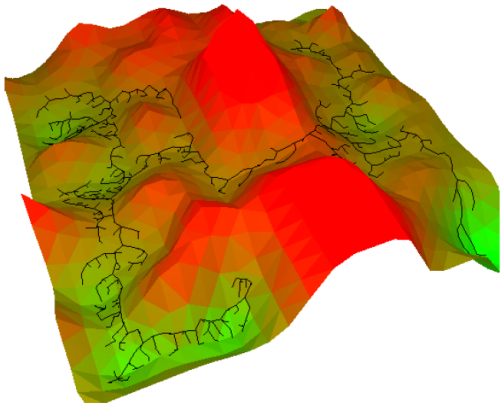


Figura 3.1 Representación de un mapa de coste de un algoritmo T-RRT en 2D (la elevación representa el coste).

3.2 Algoritmos

3.2.1 RRT*

El algoritmo RRT* es una variante del algoritmo RRT con algunas diferencias que le permiten el uso de funciones de coste. El algoritmo RRT genera estados aleatorios y, de ser válidos, se añade el nuevo estado al

conjunto de vértices existente conectando con el más cercano. La primera diferencia entre ambos algoritmos es la conexión del nuevo vértice. En lugar de conectar con el nodo más cercano, primero se obtienen los nodos cercanos al nuevo estado utilizando la fórmula empleada en el Algorithm 5 en la línea 7. Tras ello, se calcula cual de dichos nodos minimiza el coste del camino hasta el nuevo estado. El coste de un nodo consiste en el coste del nodo anterior (o nodo padre) más el coste que supone el paso entre ambos nodos . El coste del nodo inicial sería cero. De esta forma la expansión del árbol tenderá a zonas con un menor coste, mejorando la calidad del camino.

La segunda diferencia consiste en la reconexión de los nodos cercanos con cada nuevo estado que se añade. Cuando un nuevo vértice es conectado al árbol, se comprueba si el coste para alcanzar los nodos cercanos mencionados es menor a través del nuevo estado que a través de la estructura existente.

De ser así, se conecta, tal como vemos en la Figura 3.3, el nuevo estado con los estados cercanos que cumplen dicho criterio, eliminando el antiguo vínculo entre estos nodos y sus antiguos nodos padre con el objetivo de preservar la estructura de árbol. Estas diferencias le otorgan al árbol de estados del algoritmo RRT* su característico aspecto que podemos observar en la Figura 3.2.

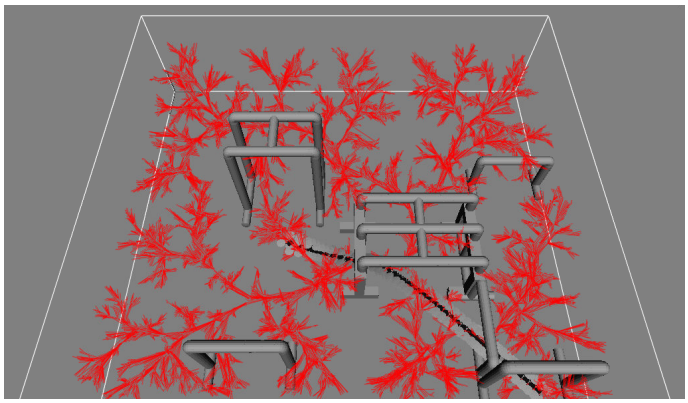


Figura 3.2 Camino obtenido con algoritmo RRT*.

En el Algorithm 5 se presenta el pseudocódico del RRT* [11].

3.2.2 T-RRT

El algoritmo T-RRT se vale del mapa de coste para la tarea de aceptar o rechazar nuevos estados [9]. La técnica usada para evaluar la calidad del camino se basa en el criterio de trabajo mínimo, Minimal Work (MW), el cual es una alternativa al clásico criterio Integral de Coste (IC). La diferencia principal entre estos dos métodos es que IC trata de minimizar el coste total del camino, mientras que MW minimiza las variaciones de coste entre estados. Una de las mayores ventajas del criterio MW frente al IC es la reducción de los máximos locales, es decir, la evitación de fuertes variaciones en el mapa de coste, lo cual resulta muy importante en diversas aplicaciones. Dicho contraste se puede apreciar en la Figura 3.4 en la que el objetivo de IC es mínimo a costa de atravesar un máximo local no deseado.

Para la exploración del mapa y la validación de los nuevos estados, usaremos dos funciones que están representadas en el Algorithm 7 y el Algorithm 8.

TransitionTest

Esta función es la encargada de rechazar estados con un elevado gradiente del coste. Para ello, la ecuación que podemos ver en el Algorithm 7 asigna una probabilidad dependiendo del valor de dicho gradiente, siendo menor cuanto mayor sea dicha diferencia de coste. A continuación explicaremos algunos de los parámetros de la misma.

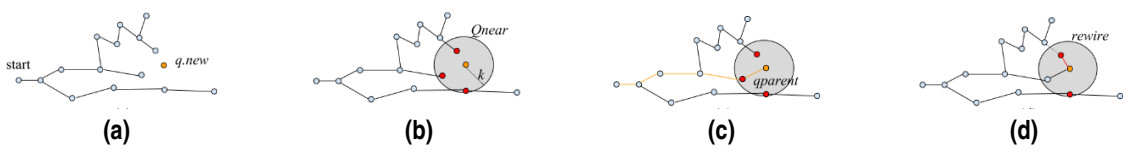
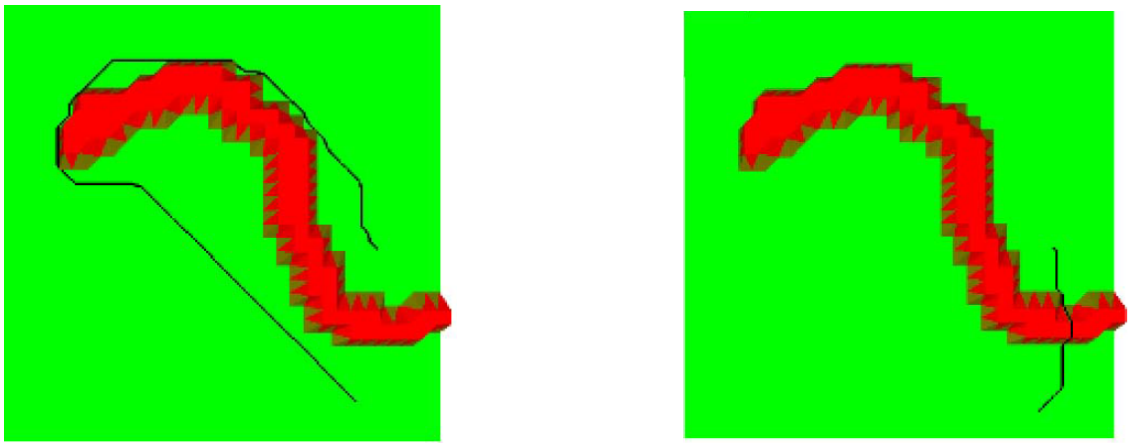


Figura 3.3 Generación de un nuevo estado en un RRT*.

Algorithm 5 Algoritmo RRT*

1: $V \longleftarrow x_{init}$; $E \longleftarrow \emptyset$;
2: **for** $i = 1,...,n$ **do**
3: $q_{rand} \longleftarrow SampleFree_i$;
4: $q_{nearest} \longleftarrow Nearest(G = (V,E), q_{rand})$;
5: $q_{new} \longleftarrow Steer(q_{nearest}, q_{rand})$;
6: **if** ObstacleFree($q_{nearest}, x_{new}$) **then**
7: $Q_{near} \longleftarrow Near(G = (V,E), q_{new}, min(\gamma_{RRT^*}(\frac{\log(card(V))}{card(V)})^{\frac{1}{d}}, \eta)$;
8: $V \longleftarrow V \cup q_{new}$;
9: $q_{min} \longleftarrow q_{nearest}$; $c_{min} \longleftarrow Cost(q_{nearest}) + c(Line(q_{nearest}, q_{new}))$;
10: **for each** $q_{near} \in Q_{near}$ **do**
11: **if** CollisionFree(q_{near}, q_{new}) $\wedge$ $Cost(q_{near}) + c(Line(q_{near}, q_{new})) < c_{min}$ **then**
12: $q_{min} \longleftarrow q_{nearest}$; $c_{min} \longleftarrow Cost(q_{nearest}) + c(Line(q_{nearest}, q_{new}))$;
13: **end if**
14: **end for**
15: $E \longleftarrow E \cup (q_{min}; q_{new})$;
16: **for each** $q_{near} \in Q_{near}$ **do**
17: **if** CollisionFree(q_{near}, q_{new}) $\wedge$ $Cost(q_{new}) + c(Line(q_{near}, q_{new})) < Cost(q_{near})$ **then**
18: $q_{parent} \longleftarrow Parent(q_{near})$;
19: **end if**
20: **end for**
21: $E \longleftarrow (E \backslash q_{parent}, q_{near}) \cup (q_{new}; q_{near})$;
22: **end if**
23: **end for**
24: **return** $G = (V,E)$;



(a) Camino según MW. **(b)** Camino según IC.

Figura 3.4 Problema con barrera de alto coste.

- p simboliza la probabilidad de aceptación del nuevo estado
- $(cj - ci)/d$ es la rampa del coste.
- K es una constante para normalizar la ecuación.
- $Temp$ es llamado Temperatura y permite la autoregulación del algoritmo. Altas temperaturas permiten subir empinadas rampas de coste, mientras que bajas temperaturas mantienen los nuevos estados en áreas más llanas.

Nótese la capacidad de autoregulación del algoritmo. Al subir pendientes en el mapa de coste, la temperatura baja para disminuir la probabilidad de ser aceptados dichos estados. De esta forma se evitan caminos menos óptimos, y cuando el número de fallos alcanza un máximo la Temperatura se incrementa, esto permite al algoritmos salir de mínimos locales o valles.

Algorithm 6 Algoritmo T-RRT

$T \longleftarrow$ InitTree(q_{init});
while not StopCondition(T,q_{goal}) **do**
  $q_{rand} \longleftarrow SampleConf(mathcalC)$;
  $q_{near} \longleftarrow NearestNeighbor(q_{rand},T)$;
  $q_{new} \longleftarrow Extend(T,q_{nearest},q_{rand})$;
  if $q_{new} \neq$ **NULL** $ANDTransitionTest(c(q_{near}),c(q_{new}),d_{near-new})ANDMinExpandControl(T,q_{near},q_{rand})$ **then**
    AddNewNode(T,q_{new});
    AddNewNode(T,q_{near} ,q_{new});
  end if
end while
return $G = (V,E)$;

Algorithm 7 TransitionTest(c_i,c_j,d_{ij})

$nFail = GetCurrentNFail()$;
if $c_j > c_{max}$ **then**
  return False;
end if
if $c_j < c_i$ **then**
  return True;
end if
$p = \exp(\frac{-\frac{(c_j - c_i)}{d_{ij}}}{K \cdot Temp})$;
if Rand(0,1) < p **then**
  $Temp = \frac{Temp}{\alpha}$;
  $nFail = 0$;
  return True;
else
  if $nFail > nFail_{max}$ **then**
    $Temp = Temp \cdot \alpha$;
    $nFail = 0$;
  else
    $nFail = nFail + 1$;
  end if
  return False;
end if

MinExpandControl

Esta función del algoritmo tiene el propósito de mantener un ratio mínimo de expansión del planificador. Esto se consigue rechazando los estados en zonas ya exploradas, llamados nodos de refinamiento, con el objetivo de incrementar el número de nuevos estados en la frontera, nodos de exploración.

3.3 Resultados

Para la evaluación comparativa de los algoritmos RRT con funciones de coste usaremos, al igual que en el caso anterior, la herramienta de benchmarking de OMPL en el mismo escenario. Para este caso, vamos a usar los algoritmos RRT* y T-RRT alternando entre 3 diferentes objetivos para la planificación.

- Length.
- Mechanicalwork.
- Maxminclearance.

El primer objetivo, como indica, trata de reducir la distancia del camino generado, mientras que el segundo, como ya hemos explicado anteriormente, trata de reducir los gradientes de coste durante la generación del

Algorithm 8 MinExpandControl(T,q_{near},q_{rand})

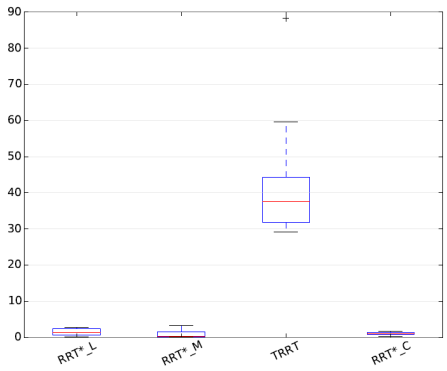
```
1: if Distance(q_near,q_rand) > δ then
2:     UpdateNbNodeTree(T);
3:     return True;
4: else
5:     if (NbRefineNodeTree(T)+1)/(NbNodeTree(T)+1) > ρ then
6:         return False;
7:     else
8:         UpdateNbRefineNodeTree(T);
9:         UpdateNbNodeTree(T);
10:        return True;
11:    end if
12: end if
```

camino, esto es, reducir los máximos locales. El tercer objetivo sigue una idea similar al objetivo Mechanicalwork. En lugar de intentar maximizar la distancia total a objetos, lo cual podría incrementar mucho la longitud del camino e incluso mínimos locales que degradarán la solución encontrada, MaxminClearance trata de maximizar los mínimos locales que se dan en la función claridad, es decir, distancia a obstáculos.

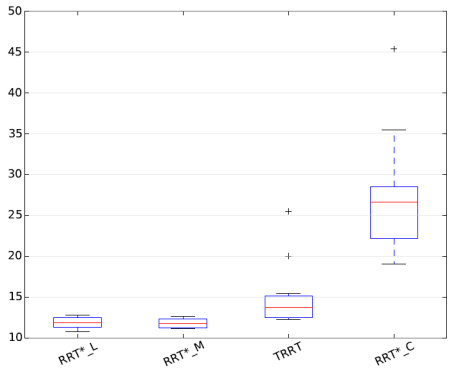
En la Figura 3.5 puede verse una comparación entre los algoritmos RRT* con los 3 diferentes objetivos de optimización y el algoritmo T-RRT. En ella podemos ver la principal diferencia entre ambos algoritmos, el RRT* buscará el camino más óptimo hasta llegar al tiempo límite, mientras que el algoritmo T-RRT devolverá el primer camino encontrado. Aun así, podemos apreciar la gran ventaja que presenta el algoritmo T-RRT en la suavidad de la solución con respecto a los RRT*.

A continuación, en la Figura 3.6 comprobaremos, debido a su mayor similitud, los resultados entre el algoritmo T-RRT y el algoritmo RRT* con el objetivo de Mechanicalwork sin la condición de cumplir un tiempo determinado. De esta forma podremos comparar el algoritmo RRT* y el T-RRT en condiciones más parecidas. Por desgracia la herramienta de evaluación comparativa de OMPL no contempla la opción de incluir gráficas que muestren el MW.

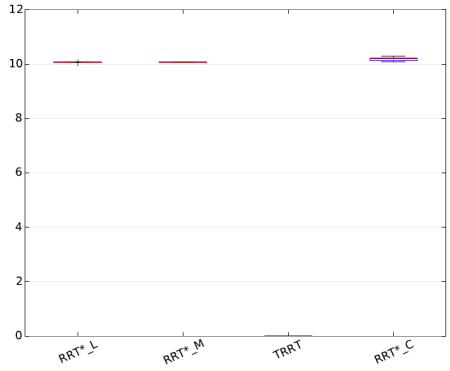
La Figura 3.6 nos muestra que el algoritmo RRT* y T-RRT tienen características muy similares con excepción de la suavidad de la solución. Por lo tanto, el algoritmo T-RRT parece ser muy adecuado para problemas donde prime más la velocidad de obtención de un camino válido, mientras que el algoritmo RRT* se usaría en el caso en el que el tiempo de cómputo no sea crítico, pudiendo encontrar una solución lo más óptima posible.



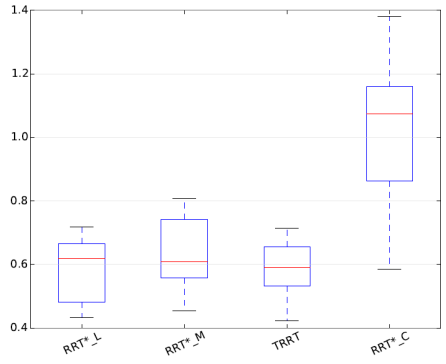
(a) Suavidad de la solución.



(b) Longitud de la solución.

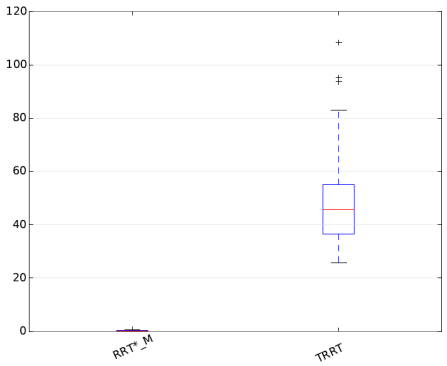


(c) Tiempo requerido.

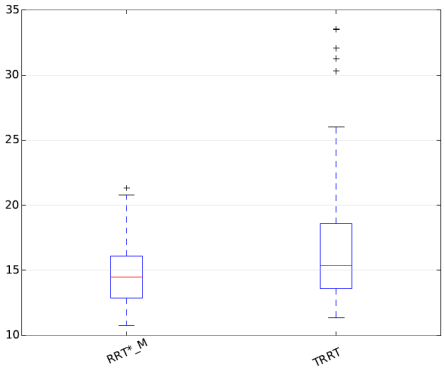


(d) Claridad de la solución.

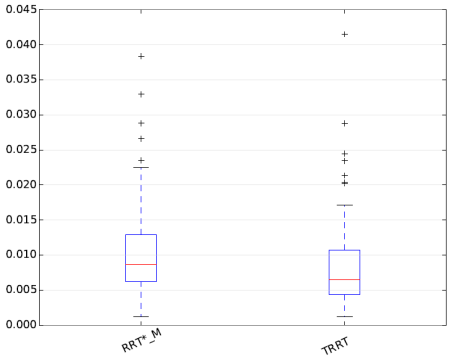
Figura 3.5 Comparación algoritmos RRT* y T-RRT.



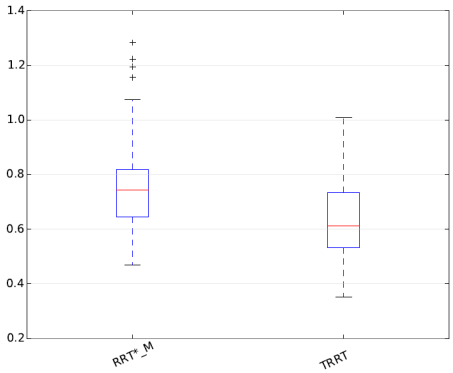
(a) Suavidad de la solución.



(b) Longitud de la solución.



(c) Tiempo requerido.



(d) Claridad de la solución.

Figura 3.6 Comparación entre T-RRT y RRT* con objetivo Mechanicalwork.

4 Planificación puramente geométrica del modelo completo con RRT

4.1 MoveIt!

MoveIt! es un software a la vanguardia en tareas de planificación con robots móviles, el cuál incorpora los últimos avances en la planificación de movimientos, manipulación de objetos, percepción 3D, cinemática, control y navegación. A su vez proporciona una plataforma fácil de usar para el desarrollo de aplicaciones de robótica avanzada, evaluación de nuevos diseños de robots y de construcción de productos con robótica integrada y otros dominios [3].

MoveIt! permite extender y configurar OMPL para su uso aplicado en robots móviles y manipuladores, permitiendo de manera directa el uso de articulaciones y, además, integrando la comunicación entre el generador de caminos y el resto de partes que constituyen el robot gracias al uso del sistema de mensajes de Robot Operating System (ROS). Asimismo, existe una considerable cantidad de información y tutoriales a nuestra disposición para aprender como usar MoveIt!, en la página web de MoveIt! [2] podemos encontrar algunos ejemplos usando el famoso robot PR2, ilustrado en la Figura 4.1.

Por todo ello, se ha optado por el uso de MoveIt! para el desarrollo de este proyecto.

Figura 4.1 Robot PR2.

4.2 Modelo completo

4.2.1 Primeros pasos

En primer lugar fue necesario crear un URDF que modelara el brazo, para poder usarlo en MoveIt! en la planificación de caminos. Se realizó un primer modelo básico, de sólo 3 articulaciones muy simples implementadas con cilindros. Un poco más adelante se nos proporcionó el modelo URDF del brazo más el multi-rotor completo, y a partir de entonces fue ese modelo el usado, con algunas variaciones. Ver Figura 4.2

Por último, se ha generado un SRDF para modelar completamente el robot aéreo y a su vez vincularlo con el entorno con el fin de poder realizar la generación de caminos.

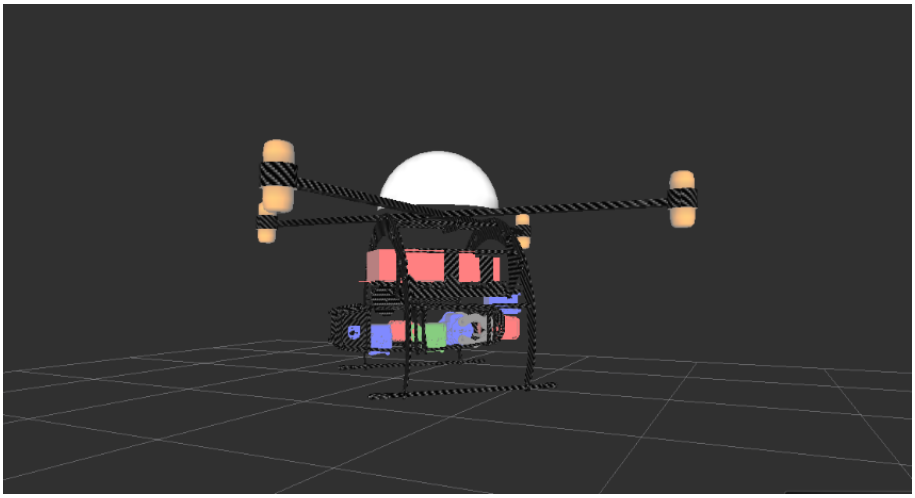


Figura 4.2 Modelo multi-rotor más brazo manipulador.

4.2.2 URDF

Un URDF es un tipo de archivo en el que se declaran todas las articulaciones y enlaces del brazo robótico, así como la relación entre éstas. Además, se pueden añadir los tamaños de las articulaciones y enlaces y sus posiciones de origen en el espacio; así como, datos de masa e inercia, color, geometría, etc.

De manera general, se crea un elemento "link" para cada enlace del brazo y un elemento "joint" para cada articulación. Cada link contiene una parte visual en la que se le indica su geometría. Además, los links contienen información de la masa y la inercia del enlace. Por último pueden contener un elemento "collision" que define una geometría sobre ese enlace para usarla a la hora de calcular las posibles colisiones entre enlaces. Un ejemplo de los elementos que componen un link en un URDF podemos verlo en el Listing 4.1.

La geometría del elemento de colisión puede ser la misma que la del propio link, pero para reducir el coste computacional a veces conviene usar geometrías más simples como cilindros.

Código 4.1 Ejemplo URDF link.

```
<link name="base_link">
    <inertial>
      <mass value="1.487"/>
      <origin xyz="0 0 0"/>
      <inertia ixx="0.01152" ixy="0.0" ixz="0.0" iyy="0.01152" iyz="0.0" izz="
          0.0218"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://ual/Media/models/bonebraker/bonebraker.dae"/>
      </geometry>
    </visual>
```

```
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://ual/Media/models/bonebraker/bonebraker.stl"/>
      </geometry>
    </collision>
  </link>
```

Por otro lado, las articulaciones sirven para conectar dos links ya definidos, se les da un nombre y se indica su tipo. Además ha de indicarse el link "padre" y el link "hijo". El código de la articulación base-brazo manipulador se muestra en Código 4.2.

Código 4.2 Ejemplo articulación fija en el URDF.

```
<joint name="base\arm\joint" type="fixed">
    <origin rpy="0.0 0.0 0.7853" xyz="0.07 0.07 0.11"/>
    <parent link="base\link"/>
    <child link="arm\base\link"/>
  </joint>
```

Esta articulación es de tipo "fixed" porque realmente sólo es un punto de unión entre el modelo del multi-rotor y el brazo manipulador y no permite ningún tipo de movimiento. Para articulaciones de rotación el tipo sería "revolute". Un ejemplo de este tipo de articulaciones sería la que pertenece a la unión entre la muñeca y los dedos del brazo, como muestra el Código 4.3.

Código 4.3 URDF de la articulación wirst_1-claw_1.

```
<joint name="claw_1_y_arm_joint" type="revolute">
    <axis xyz="0 -1 0"/>
    <limit effort="1000" lower="-2.932" upper="0.0" velocity="0.001"/>
    <origin rpy="03.14159 0.235 0.0" xyz="0.019 0.0 0.015"/>
    <parent link="wirst_1"/>
    <child link="claw_1"/>
  </joint>
```

4.2.3 SRDF

El SRDF es un archivo que contiene información adicional al URDF sobre el modelo. El SRDF está compuesto por un elemento "robot" que es la raíz del archivo y contiene toda la información extra necesaria para la planificación de caminos. El elemento "robot" está dividido en varios subconjuntos con la etiqueta de "group", como podemos ver en el Código 4.4, los cuales relacionan y agrupan un conjunto de articulaciones y enlaces. Esto nos permitirá dividir la planificación en varias fases, en las que usaremos solo los grados de libertad necesarios. Esto quiere decir que podemos seleccionar que grupos de articulaciones podrán tenerse en cuenta en la planificación. De esta manera, reduciremos el coste computacional innecesario seleccionando solo el conjunto de articulaciones necesarios y, a su vez, también evitaremos movimientos indeseados de ciertas partes del robot. El archivo contiene los siguientes grupos:

- **Base**, que estará formado por el cuerpo del multi-rotor. Lo usaremos para la tarea de desplazar el robot hasta una posición de manipulación, manteniendo el brazo inmóvil.
- **Arm**. Este grupo contiene las articulaciones del brazo manipulador con la excepción del efector final. Se utilizará para las tareas de planificación de caminos del brazo.
- **Claw**. Está compuesto por las articulaciones de los dos dedos que forman la garra del brazo manipulador. Este grupo trabajará de forma conjunta con el grupo arm para las tareas de agarre y colocar objetos.
- **Whole_robot**. Este grupo engloba todos los grados de libertad del multi-rotor más el brazo manipulador. Dicho grupo no será usado en la manipulación y su función se reduce a la primera toma de contacto para coger experiencia en el uso de MoveIt!.

De esta forma podemos separar la planificación del movimiento del cuerpo del multi-rotor y el del brazo manipulador. Esto nos permitirá ejecutar dichos movimientos de forma secuencial, minimizando el efecto de la dinámica del sistema.

Código 4.4 Articulación virtual entre el robot y el entorno en SRDF.

```
<group name="base">
       <joint name="world_joint" />
   </group>
   <group name="arm">
       <chain base_link="base_link" tip\_link="wirst_1" />
   </group>
   <group name="whole_robot">
       <group name="base" />
       <group name="arm" />
   </group>
   <group name="claw">
       <link name="claw_0" />
       <link name="claw_1" />
   </group>
```

Como se puede apreciar en el Código 4.4, en el SRDF se define una articulación virtual que definirá el estado del robot con respecto a un punto fijo del entorno. Dicha articulación tendrá los mismos Degrees of Freedom (DOF), grados de libertad en inglés, que el multi-rotor, y permitirá conectar la posición del robot con el entorno. Quedando el árbol de articulaciones como se muestra en la Figura 4.3.

Es posible definir conjuntos de configuraciones fijas que podremos usar como checkpoint en la planificación mediante "group_state", en el Código 4.5 tenemos un ejemplo de la posición de inicio del brazo, también podemos crear configuraciones del tipo "brazo_doblado" y "brazo_estirado".

Código 4.5 Grupo de manipulación en SRDF.

```
<group_state name="start" group="arm">
       <joint name="elbow_0_p_arm_joint" value="2" />
       <joint name="elbow_0_r_arm_joint" value="0" />
       <joint name="shoulder_p_arm_joint" value="-2" />
       <joint name="shoulder_y_arm_joint" value="0" />
       <joint name="wirst_0_p_arm_joint" value="0" />
       <joint name="wirst_1_r_arm_joint" value="0" />
   </group\_state>
```

El efector final también se define en el SRDF bajo la etiqueta de "end_effector", ilustrado en el Código 4.6, el cual contiene la información sobre el efector como su nombre, el grupo al que pertenece y el link padre.

Código 4.6 Declaración efector final.

```
<end_effector name="eef" parent_link="wirst_1" group="claw" />
```

De esta forma podremos primero mover el multi-rotor con el grupo base, tras ello planificaremos el camino del brazo hasta la posición de agarre. Tras ello ejecutaremos el cierre de la garra con el grupo del efector final, colocando la barra en el lugar deseado usando el mismo procedimiento.

Por último, el SRDF también contiene la matriz que evita la comprobación de colisiones redundantes, "disable:collisions" que tiene gran interés, ya que permite indicarle al programa una serie de colisiones entre enlaces que no debe comprobar. En un modelo existen varios enlaces o links que en la realidad nunca podrían llegar a colisionar debido al diseño del robot. Por defecto, el CC comprobaría todas las colisiones posibles entre todos los links. Al indicarle qué colisiones entre enlaces no debe comprobar se minimiza considerablemente el tiempo de cómputo necesario para la planificación.

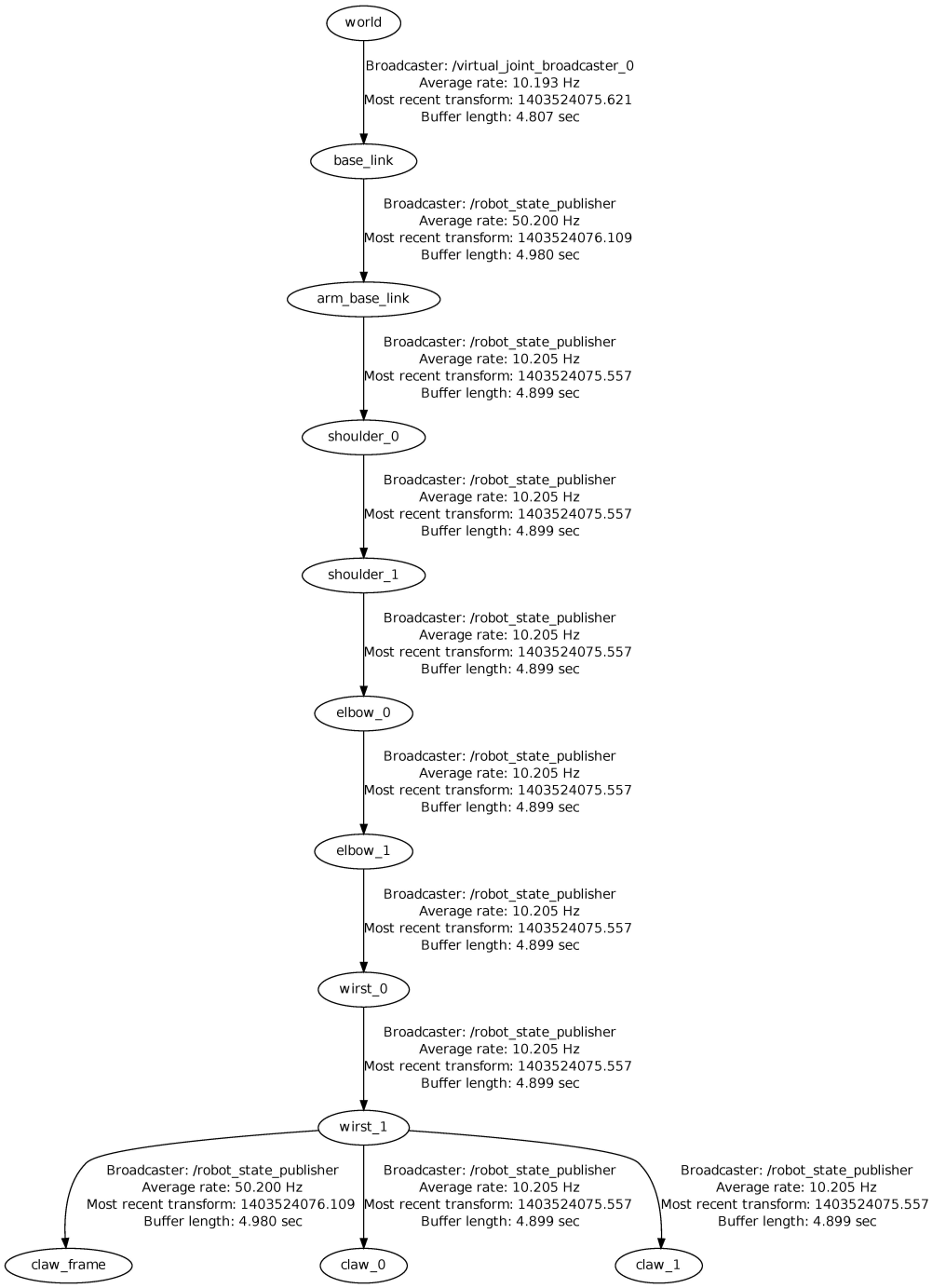


Figura 4.3 Interfaz gráfica OMPL.

4.3 Manipulación

Tal como muestra la Figura 4.4, para la elaboración de las pruebas de generación de caminos con el modelo completo, el robot tendrá la tarea de manipular una barra cilíndrica situada verticalmente en una superficie plana, y deberá depositarla de igual modo en otra posición en dicha superficie. Para ello el multi-rotor tendrá que desplazarse desde una posición inicial a una próxima a la barra, evitando los posibles obstáculos del entorno del CATEC, desplegar el brazo manipulador e iniciar el procedimiento de recogida y colocación evitando el obstáculo que se interpone entre la ubicación inicial y final de la barra.

En las tareas de planificación de caminos que implican una manipulación con objetos, el agarre y/o colocación de dichas piezas es una de las partes más críticas del problema. La idea original fue limitar dicho problema a una serie de posiciones prefijadas, reduciendo el problema a indicar el estado inicial y final del brazo manipulador. Durante la realización de este proyecto se decidió dar un paso adelante e implementar un algoritmo propio para poder incluir la manipulación en las tareas de planificación de una manera más realista. Para ello hemos dividido la planificación del manipulador en dos partes, preagarre y postagarre, que describirán el camino antes y después del agarre o colocación respectivamente. A su vez, cada una de estas partes se dividirá en dos debido a la necesidad de incluir un movimiento de acercamiento y otro de retirada. Esto es debido a que en la aproximación final del manipulador a su objetivo se debe aplicar restricciones en el movimiento de forma que solo pueda moverse en una dirección. Esta dirección debe ser perpendicular a la superficie donde queramos dejar el objeto en caso de colocación, o el desplazamiento frontal del efector final para agarrar una pieza. De esta forma la manipulación estaría dividida en cuatro tramos, que se ilustra en el Código 4.7 con el caso de agarre de una pieza.

- El movimiento del manipulador hasta una posición cercana orientada al objetivo.
- El acercamiento a la pieza restringido a la dirección pieza-garra, la cual suele ser el eje de la muñeca del manipulador.
- La retirada de la pieza en dirección perpendicular a la superficie que sostenía el objeto.
- Transporte de la pieza.

Durante el último tramo del camino, el planificador deberá tener en cuenta no sólo la geometría del robot manipulador, sino también la geometría del objeto que está siendo manipulado en el CC. El estado intermedio antes de la fase de acercamiento será un checkpoint que servirá para colocar el manipulador en la posición correcta para efectuar la aproximación final al objeto en el preagarre. El estado intermedio tras la retirada será un punto de inicio seguro, esto es, suficientemente alejado de los obstáculos, para el resto de la planificación con el objeto adjunto.

Esta estructura está implementada en las funciones "pick" y "place" en MoveIt! de la siguiente en que se ilustra en el Código 4.7.

Código 4.7 Configuración posición de agarre.

```
pose.header.frame_id = "base_link";
pose.header.stamp = ros::Time::now();
pose.pose.position.x = origin.m_floats[0];
pose.pose.position.y = origin.m_floats[1];
pose.pose.position.z = origin.m_floats[2];
grasp.grasp_pose = pose;

// Creación postura preagarre
grasp.pre_grasp_posture.header.frame_id = BASE_LINK;
grasp.pre_grasp_posture.header.stamp = ros::Time::now();

// Nombre de las articulaciones preagarre
grasp.pre_grasp_posture.joint_names.resize(2);
grasp.pre_grasp_posture.joint_names[0] = EE_JOINT0;
grasp.pre_grasp_posture.joint_names[1] = EE_JOINT1;

// Estado de las articulaciones preagarre
grasp.pre_grasp_posture.points.resize(2);
grasp.pre_grasp_posture.points[0].positions.resize(1);
grasp.pre_grasp_posture.points[0].positions[0] = -0.5;
grasp.pre_grasp_posture.points[1].positions.resize(1);
grasp.pre_grasp_posture.points[1].positions[0] = 0.5;

// Creación postura postagarre
grasp.grasp_posture.header.frame_id = BASE_LINK;
grasp.grasp_posture.header.stamp = ros::Time::now();

// Estado de las articulaciones
grasp.grasp_posture.joint_names.resize(2);
grasp.grasp_posture.joint_names[0] = EE_JOINT0;
grasp.grasp_posture.joint_names[1] = EE_JOINT1;
```

```
// Posición de las articulaciones
grasp.grasp_posture.points.resize(2);
grasp.grasp_posture.points[0].positions.resize(1);
grasp.grasp_posture.points[1].positions.resize(1);
grasp.grasp_posture.points[0].positions[0] = M_PI/18;
grasp.grasp_posture.points[1].positions[0] = M_PI/18;

// Nums
grasp.pre_grasp_approach.min_distance = 0.1;
grasp.pre_grasp_approach.desired_distance = 0.15;
grasp.pre_grasp_approach.direction.vector.x=1;
grasp.pre_grasp_approach.direction.header.frame_id="wirst_1";

grasp.post_grasp_retreat.min_distance = 0.05;
grasp.post_grasp_retreat.desired_distance = 0.1;
grasp.post_grasp_retreat.direction.vector.z=1;
grasp.post_grasp_retreat.direction.header.frame_id="base_link";
```

En el Código 4.7 podemos identificar los siguientes términos:

- **Pose**, posición de la pieza a agarrar o lugar de colocación.
- **Pre_grasp_posture**, define las articulaciones del manipulador y los valores que utilizará para el agarre, es decir, las articulaciones del efector final, antes del movimiento acercamiento al objeto.
- **Pre_grasp_approach**, define la restricción en el acercamiento.
- **Post_grasp_retreat**, define la restricción en la retirada.
- **Grasp_posture**, define el valor delas articulaciones del efector final para agarrar el objeto.

Las funciones "pick" y "place" necesitan únicamente el nombre del objeto a manipular y definir la superficie de apoyo del mismo para ejecutar la planificación, pero, en la práctica, esto conlleva un alto riesgo de que no se obtenga una solución válida. Por ello, se han creado dos funciones, "generate_grasps" y "generate_places" que generan un vector de posibles estados de agarre/colocación que serán dados como argumentos a las funciones "pick" y "place", tal y como podemos ver en el Código 4.8. Dichas funciones reciben como argumento la posición del objeto o lugar de colocación y calculan la posición del efector final para una serie de orientaciones de agarre por medio de una serie de transformaciones, conocida la posición del robot y del objetivo. De esta forma conseguimos aumentar notablemente la probabilidad de obtener un camino válido.

Código 4.8 Configuración posición de agarre.

```
//Para coger el objeto "part"
grasps=generate_grasps(block_pose.position.x,block_pose.position.y,block_pose.position.z);
group.pick("part", grasps);

//Para situar el objeto "part" en la superficie "table"
loc=generate_places(pos.pose.position.x,pos.pose.position.y,pos.pose.position.z);
group.setSupportSurfaceName("table");
group.place("part", loc);
```

En el proceso de manipulación, la planificación se dividirá en dos grupos, el cuerpo del multi-rotor y el brazo manipulador por dos razones.

- Disminuir la complejidad del problema disminuyendo el número de grados de libertad.
- Eliminar el efecto de la dinámica del problema lo máximo posible. Al mover la base del multi-rotor y el brazo manipulador secuencialmente nos aseguramos que el robot pueda seguir el camino generado de una forma estable.

4.4 Código desarrollado

Para la realización de la tarea de manipulación, el brazo manipulador debe colocarse en la posición de inicio (posición en la que el brazo se encuentra plegado bajo el multi-rotor) para, a continuación, coger la pieza, representada por una barra cilíndrica vertical, de una superficie que hará la función de mesa. Tras ello,

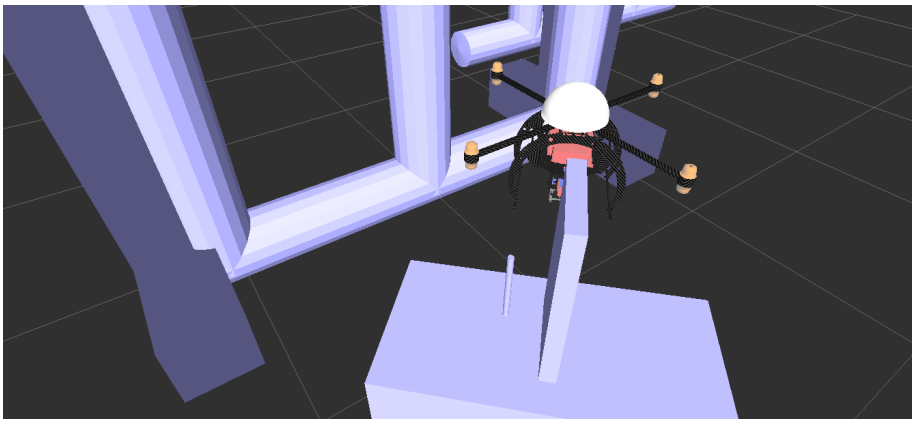


Figura 4.4 Escenario de manipulación.

debe moverla hasta el punto de anclaje indicado evitando un obstáculo en el centro de dicha superficie. Para dicha tarea se ha desarrollado un archivo en C++ llamado "pick_place.cpp". El archivo consta de varias funciones entre las que destacan: quadmove, plegar, pick y place. Además, se ha necesitado el conocimiento y la manipulación de varios nodos de ROS para la elaboración de dicho código. Todo ello lo explicaremos en los siguientes apartados.

4.4.1 Move_group

El nodo central es move_group que es el encargado de controlar todas las acciones del robot, y de comunicarse con el resto de nodos. Para ello dispone de los siguientes servicios, en los que no entraremos en detalle, que dan una idea de las capacidades de move_group.

- MoveGroupCartesianPathService
- MoveGroupExecuteService
- MoveGroupKinematicsService
- MoveGroupMoveAction
- MoveGroupPickPlaceAction
- MoveGroupPlanService
- MoveGroupQueryPlannersService
- MoveGroupQueryPlannersService
- MoveGroupStateValidationService
- MoveGroupGetPlanningSceneService

4.4.2 Joint_state_publisher y Robot_state_publisher

Joint_state_publisher se encarga de almacenar en el topic joint_states los valores de las variables articulares (sólo aquellas con un grado de libertad). Robot_state_publisher traduce esos valores al topic tf, donde se guardan las transformaciones entre los frames de los distintos links del robot y el entorno.

4.4.3 Virtual_joint_broadcaster_0

Es el nodo encargado de publicar la transformación entre el frame fijo del entorno, "world", y el frame de la base del multi-rotor a través de la herramienta static_transform_publisher. Esto es necesario porque en joint_states sólo se almacena información de articulaciones de 1 DOF y la "articulación" del multi-rotor con el entorno tiene 6 DOF. Por este motivo se deben publicar las transformaciones de forma alternativa al joint_state_publisher, y para el caso en cuestión, movimiento del brazo manipulador, se ha usado un publicador de transformaciones fijas.

4.4.4 Pick and place

Estos nodos se encargan del proceso de agarrar y soltar piezas. Se proporciona la pieza a coger y un vector con posiciones de agarre, y realiza la planificación de la tarea dividiéndola en tres partes: el pre-agarre, el agarre y el post-agarre. Se debe especificar en qué dirección y a qué distancia deberá acercarse y alejarse la garra antes y después de agarrar o soltar la pieza. Para ello realiza 3 etapas de cómputo en las cuales procesa qué posiciones de agarre son válidas, la ejecución del acercamiento y el alejamiento y, por último, la trayectoria completa.

4.4.5 Rviz

Es el nodo que se encargará de visualizar las acciones de los nodos move_group, pick y place.

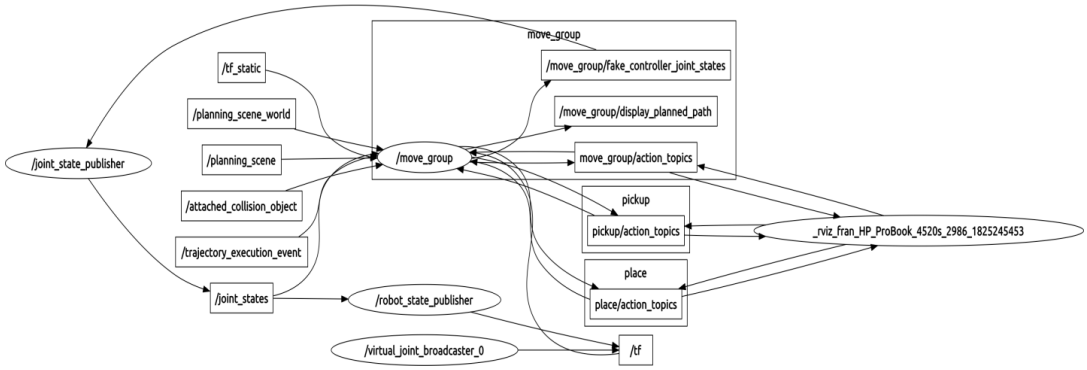


Figura 4.5 Nodos conectados a Rviz.

4.4.6 Arm_pick_place

Éste es el nodo desarrollado para la realización del trabajo. Será el encargado de crear el entorno de trabajo e interactuar con el resto de nodos que se ilustran en la Figura 4.6 para conseguir realizar una tarea específica, quedando conectados como muestra la Figura 4.7. Los detalles del código se explican en el siguiente apartado.

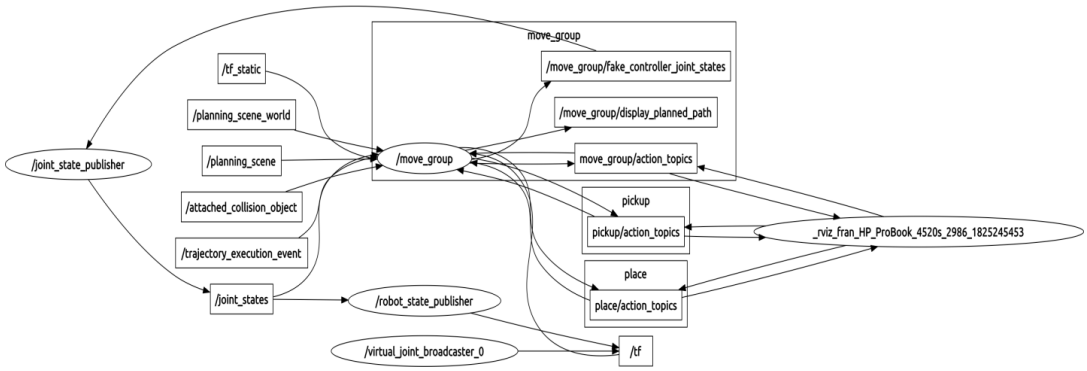


Figura 4.6 Grafo de los nodos antes de insertar Arm_pick_place.

4.4.7 Pick_place.cpp

La función main de dicho archivo se encarga de crear el entorno a través de un tipo de mensaje CollisionObject, el cual tendrá en cuenta el CC del planificador escogido a la hora de identificar estados no válidos. Además de las piezas que conforman el escenario de manipulación (barra, mesa y obstáculo), también está el objeto “Arcas”, el cual corresponde al entorno de tuberías facilitado por CATEC. Todos estos objetos se publican en el topic Collision_object y serán visualizados en Rviz. Para evitar problemas en el código

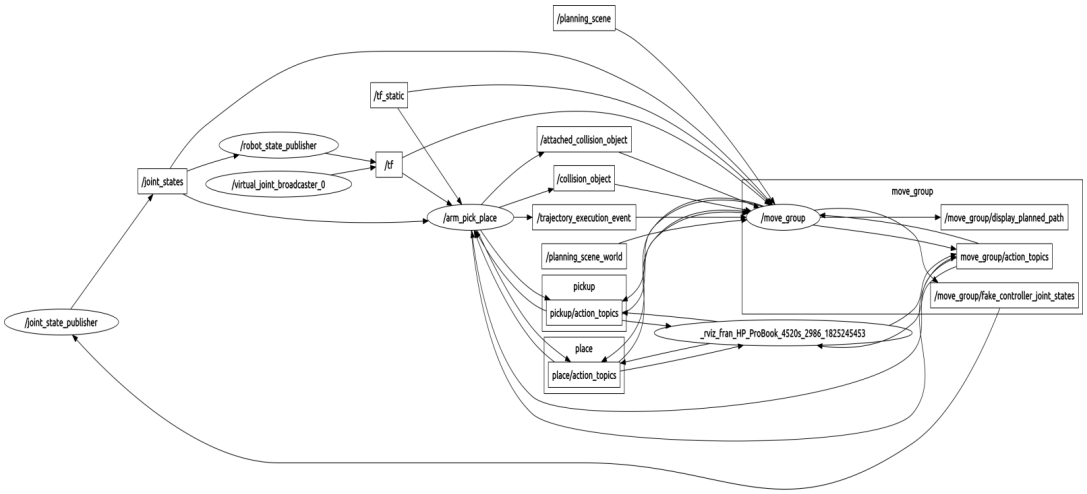


Figura 4.7 Grafo de la conexión de Arm_pick_place con el resto de nodos.

primero se reinicia el entorno publicando la orden de borrar los objetos actuales para después publicarlos en las posiciones correctas (en caso de que se haya quedado una simulación a medias o se haya desplazado un objeto indebidamente). En la función main también se definen los dos grupos de planificación que se usarán: "group" y "quad" en referencia a los grupos definidos en el SRDF "base" y "arm" para mover el multi-rotor y el brazo manipulador respectivamente. Éste será el vínculo con el nodo Move_group.

4.4.8 Plegar y quadmove

La función plegar recibe la variable group, le asigna unos valores al brazo para plegarlo bajo la base y ejecuta dicho movimiento. Se ha escogido una posición de reposo en la que el brazo se mantenga lo más centrado y cercano al multi-rotor posible sin que haya contacto. El objetivo de ello es evitar en la medida de lo posible que la dinámica del movimiento del multi-rotor se vea lo menor influenciada posible por el brazo manipulador cuando se ejecute dicho camino. Los valores articulares elegidos para dicho estado de reposo se guardan en la posición "start" como puede verse en Código 4.5.

La función quadmove funciona de forma similar pero actuando en el grupo de planificación "base", el cual corresponde al movimiento del multi-rotor.

4.4.9 Pick y place

Las funciones pick y place tienen una metodología muy similar. El objetivo de estas funciones es el de crear el vector de posiciones para pasárselas a las funciones pick y place de la clase MoveGroup y, a su vez, a los nodos de Pick y Place. Para ello usamos las funciones tf_transform_to_grasp / tf_transform_to_place y generate_grasps / generate_places. Las primeras son las encargadas de convertir un tipo de dato transform en otro tipo Grasp o PlaceLocation. Las segundas se encargan de conseguir el vector de posiciones de agarre, Grasp, y de posiciones de colocación, PlaceLocation. Para ello reciben la posición de la pieza a manipular y calculan la transformación entre la base del multi-rotor y la muñeca del brazo. Hay que tener en cuenta que el extremo del grupo de planificación es la muñeca, no la posición de agarre del efector final. Por esto es necesario transformar la posición de la pieza a la de la muñeca para poder planificar respecto a la posición de la muñeca. Sabiendo la distancia entre la muñeca y la posición de agarre, 5 cm en este caso, se genera la transformada pieza->muñeca y a su vez, sabiendo la posición de la base del multi-rotor respecto del frame fijo definido por static_transform_publisher, se calcula la transformada $world -> base\_link$ y calculando su inversa tenemos: $Tbase\_link \rightarrow world * Tworld \rightarrow pieza * Tpieza \rightarrow mueca = Tbase\_link \rightarrow mueca$.

Para generar el vector se usa un bucle que irá variando el ángulo de giro de la muñeca en la posición de agarre de la pieza. Como curiosidad, en el agarre, en la trasformación entre la pieza y la muñeca hubo que hacer una rotación de 90º de Roll para poder coger la barra vertical, mientras que en la función Place se conseguía el mismo resultado sólo sin realizar dicha rotación. Como comentario añadido, para que no se produzcan fallos de colisión entre la pieza a manipular y la superficie donde reposa se define dicho objeto como superficie de apoyo en MoveGroup para obviar las colisiones entre la pieza y la superficie. Esta es una

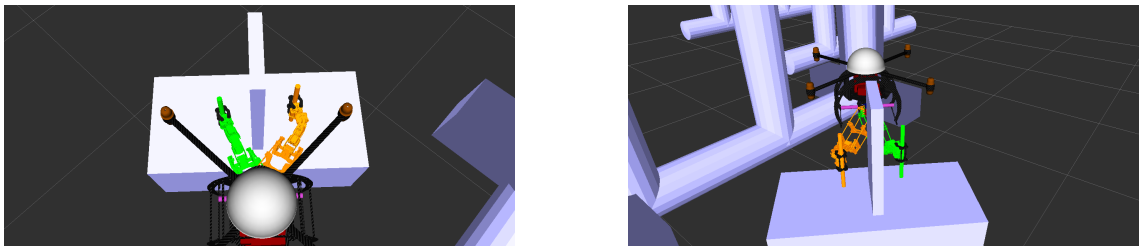


Figura 4.8 Estados inicial y final de la simulación.

de las razones que hacen crítico el escoger unas direcciones de aproximación y alejamiento adecuados para la tarea. Por ello se han escogido la base del multi-rotor y la muñeca para el acercamiento y alejamiento en el eje Z de la base y el eje X del brazo.

4.5 Resultados

En este apartado comprobaremos como responden los algoritmos RRT en el caso de un brazo manipulador de 6 DOF. Específicamente, se ha simulado el caso del transporte de una barra desde una parte de una superficie, "table", hasta la opuesta evitando un obstáculo, "pole", situado en el medio de dicha superficie, como puede verse en la Figura 4.8.

Los resultados varían dependiendo de la versión de MoveIt! usada. Usando la versión de MoveIt! basada en ROS Hydro nos encontramos con unas estadísticas bastante pobres para el problema del brazo manipulador como podemos observar en la Figura 4.9. Sin embargo, utilizando MoveIt! basado en ROS Indigo usando la versión de OMPL 1.0.0 obtenemos resultados radicalmente diferentes, Figura 4.8, en el cual la tasa de obtención de caminos válidos incrementa notablemente. Además, con versiones previas, muchos de los algoritmos no funcionaban correctamente durante la ejecución de las herramientas de evaluación comparativa de MoveIt!. Respecto a la última versión instalada, cuyos resultados podemos ver en la Figura 4.10, se han encontrado algunos errores con el algoritmo T-RRT y para tener al menos una referencia en las gráficas se ha añadido un benchmarking obtenido con una versión antigua de MoveIt!, de ahí se entiende el bajo porcentaje de la Figura 4.10. Para mayor claridad en los resultados se ha utilizado el objetivo de planificación de minimizar la longitud del camino. Como se puede observar en Figura 4.11, los algoritmos RRT obtienen una respuesta similar a los PRM, consiguen un camino más corto aunque la suavidad del camino es inferior a otros planificadores como EST, KPIECE o SBL. Cabe una mención especial al algoritmo RRTConnect que ha obtenido en todas las pruebas alto porcentaje de éxito empleando unos tiempos de cómputo muy bajos.

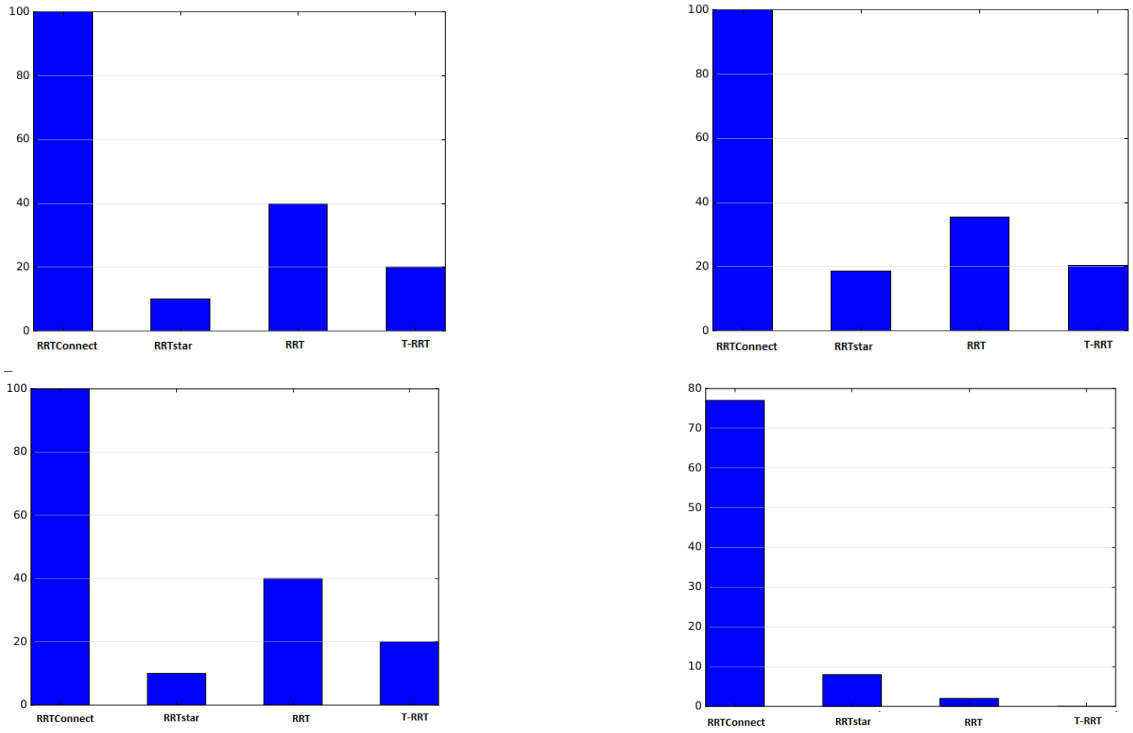


Figura 4.9 Porcentaje de resultados válidos con MoveIt! versión Hydro.

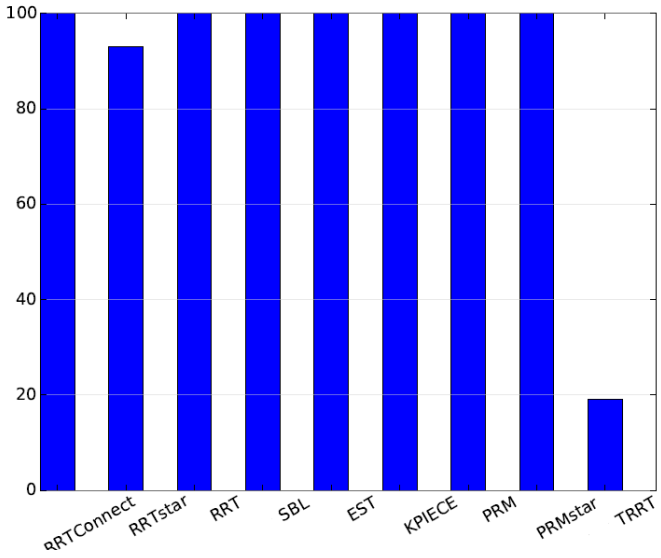
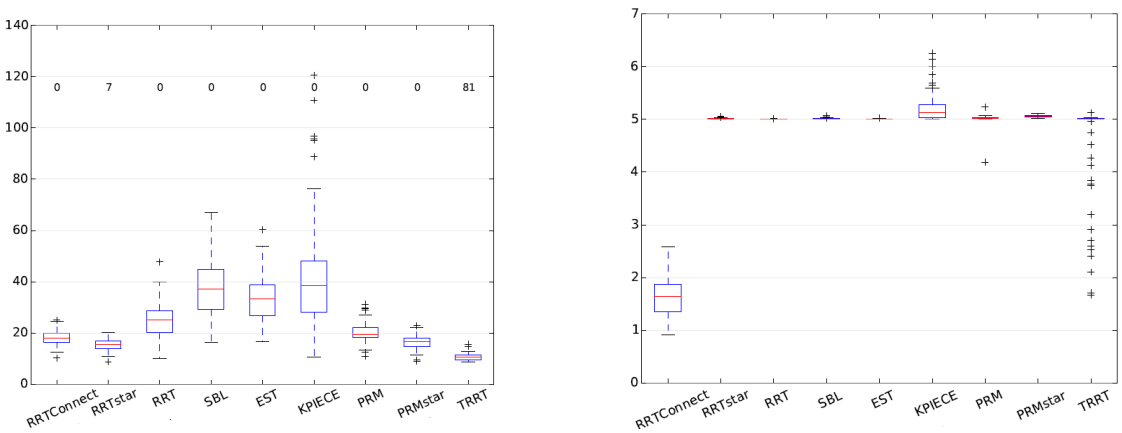


Figura 4.10 Porcentaje de resultados válidos con MoveIt! versión Indigo.



(a) Longitud de la solución .

(b) Tiempo de cómputo .

Figura 4.11 Resultados de la planificación del brazo.

5 Conclusiones y desarrollos futuros

5.1 Conclusiones

Se ha presentado el algoritmo de planificación de caminos RRT con sus distintas variantes considerando el caso de un robot multi-rotor con brazo manipulador en entornos conocidos. Los RRT han demostrado ser unos algoritmos de planificación capaces de resolver problemas complejos como el propuesto en este documento, un robot aéreo con un total de 14 DOF (6 DOF del multi-rotor + 6 DOF del brazo manipulador + 2 DOF de la garra) con tiempos de cómputo bajos, pudiendo ser usados en aplicaciones de tiempo real. Además cuenta con la ventaja de ser un algoritmo muy usado y estudiado, existiendo multitud de variantes fácilmente implementables para todo tipo de problemas y situaciones.

Para ello se han probado dichos planificadores de caminos en la librería OMPL usando simplemente un multi-rotor, pasando tras ello a la plataforma de MoveIt! para la integración de las articulaciones del brazo manipulador a la tarea de planificación. Al ser MoveIt! parte de ROS, esto nos da la oportunidad de utilizar todas las funcionalidades de este último como el visualizador Rviz con el que se han realizado las simulaciones, así como el sistema de mensajes (nodos) que facilitan la comunicación entre las distintas partes del robot.

5.2 Desarrollos futuros

Existen dos puntos muy interesantes para futuros desarrollos que constarían de:

- Integración de la dinámica del sistema.
- Planificación cooperativa con varios robots.

El presente trabajo se ha centrado en modelos cinemáticos por varias razones. En primer lugar para acotar el alcance del proyecto, pero también por el hecho de la escasa variedad existente para este tipo de problemas. Por ejemplo, los planificadores óptimos aún no han sido implementados en OMPL. Una extensión del trabajo existente podría añadir la dinámica del sistema con el objetivo de obtener caminos más precisos y eficientes. Con ello se podría planificar conjuntamente el multi-rotor y el brazo simultaneamente para tareas de manipulación, funcionalidad que se ha limitado a la alternancia entre el movimiento del multi-rotor y el brazo manipulador para mitigar lo máximo posible la influencia de dicha dinámica.

Siguiendo la temática del proyecto ARCAS, otra idea interesante podría ser el desarrollo de estrategias de planificación conjuntas entre varios multi-rotors, basándose en algotimos RRT [10], para tareas de manipulación como por ejemplo el transporte de objetos y construcción de estructuras. Un ejemplo de ello lo encontramos en [1]

Índice de Figuras

Índice de Algoritmos

Índice de Códigos

Bibliografía

[1] Markus Bernard, Konstantin Kondak, Ivan Maza, and Anibal Ollero, *Autonomous transportation and deployment with aerial robots for search and rescue missions*, Journal of Field Robotics **28** (2011), no. 6, 914–931.

[2] S. Chitta, I. Sucan, and S. Cousins, *Moveit! website*.

[3] _____, *Moveit! [ros topics]*, Robotics Automation Magazine, IEEE **19** (2012), no. 1, 18–19.

[4] E. W. Dijkstra, *A note on two problems in connexion with graphs*, NUMERISCHE MATHEMATIK **1** (1959), no. 1, 269–271.

[5] M. Elbanhawi and M. Simic, *Sampling-based robot motion planning: A review*, Access, IEEE **2** (2014), 56–77.

[6] P.E. Hart, N.J. Nilsson, and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths*, Systems Science and Cybernetics, IEEE Transactions on **4** (1968), no. 2, 100–107.

[7] D. Hsu, J.-C. Latombe, and R. Motwani, *Path planning in expansive configuration spaces*, Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on, vol. 3, Apr 1997, pp. 2719–2726 vol.3.

[8] Y.K. Hwang and N. Ahuja, *A potential field approach to path planning*, Robotics and Automation, IEEE Transactions on **8** (1992), no. 1, 23–32.

[9] L. Jaillet, J. Cortés, and T. Siméon, *Sampling-based path planning on configuration-space costmaps*, Robotics, IEEE Transactions on **26** (2010), no. 4, 635–646.

[10] S. Kamio and H. Iba, *Cooperative object transport with humanoid robots using rrt path planning and re-planning*, Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, Oct 2006, pp. 2608–2613.

[11] S. Karaman and E. Frazzoli, *Sampling-based algorithms for optimal motion planning*, International Journal of Robotics Research **30** (2011), no. 7, 846–894.

[12] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, Robotics and Automation, IEEE Transactions on **12** (1996), no. 4, 566–580.

[13] O. Khatib, *Real-time obstacle avoidance for manipulators and mobile robots*, Robotics and Automation. Proceedings. 1985 IEEE International Conference on, vol. 2, Mar 1985, pp. 500–505.

[14] J.J. Kuffner and S.M. LaValle, *Rrt-connect: An efficient approach to single-query path planning*, Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on, vol. 2, 2000, pp. 995–1001 vol.2.

[15] Andrew M. Ladd, Rice Unversity, Lydia E. Kavraki, and Rice Unversity, *Motion planning in the presence of drift, underactuation and discrete system changes*, In Robotics: Science and Systems I, MIT Press, 2005, pp. 233–241.

[16] S.M. LaValle, *Motion planning*, Robotics Automation Magazine, IEEE **18** (2011), no. 1, 79–89.

[17] Steven M. Lavalle, *Rapidly-exploring random trees: A new tool for path planning*, Tech. report, 1998.

[18] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun, *Anytime search in dynamic graphs*, Artificial Intelligence **172** (2008), no. 14, 1613–1643.

[19] Vladimir J. Lumelsky and Alexander A. Stepanov, *Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape*, ALGORITHMICA (1987).

[20] N. B. Ismail S. H. Tang, W. Khaksar and M. K. A. Ariffin, *A review on robot motion planning approaches*, Pertanika Journal of Science and Technology **20** (2012), no. 1, 15–29.

[21] Gildardo Sánchez and Jean claude Latombe, *A single-query bi-directional probabilistic roadmap planner with lazy collision checking*, 2001.

[22] Anthony Stentz and Is Carnegle Mellon, *Optimal and efficient path planning for unknown and dynamic environments*, International Journal of Robotics and Automation **10** (1993), 89–100.

[23] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki, *The Open Motion Planning Library*, IEEE Robotics & Automation Magazine **19** (2012), no. 4, 72–82, *http://ompl.kavrakilab.org*.

[24] IoanA. Şucan and LydiaE. Kavraki, *Kinodynamic motion planning by interior-exterior cell exploration*, Algorithmic Foundation of Robotics VIII (GregoryS. Chirikjian, Howie Choset, Marco Morales, and Todd Murphey, eds.), Springer Tracts in Advanced Robotics, vol. 57, Springer Berlin Heidelberg, 2010, pp. 449–464 (English).

Glosario

ARCAS Aerial Robotics Cooperative Assembly System. 1, 39

Assimp Open Asset Import Library. 7

CATEC Centro Avanzado de Tecnologías Aeroespaciales. 9, 11, 29, 33

CC Comprobador de Colisiones. 5, 7, 10, 28, 30

DOF Degrees of Freedom. 28, 32, 35, 39

EST Expansive Space Trees. 4, 5, 35

FCL Flexible Collision Library. 7

IC Integral de Coste. 18, 19

KPIECE Kinematic Planning by Interior-Exterior Cell Exploration. 4, 35

MW Minimal Work. 18, 19, 21

OMPL The Open Motion Planning Library. III, V, 5, 7, 9, 11, 20, 25, 39, 41

PDST Path-Directed Subdivision Trees. 4

PQP Proximity Query Package. 7

PRM Probabilistic Roadmap Method. 4, 5

ROS Robot Operating System. 25, 32, 35, 39

RRT The Rapidly-Exploring Random Tree. 4, 5, 9–15, 17, 41

SBL Single-query Bi-directional Lazy collision checking planner. 4, 35

SRDF Semantic Robot Description Format. V, 26–28, 34

T-RRT Transition-based RRT. V, 17, 18, 20–23, 41, 43

UAVs Unmanned Aerial Vehicle. 1

URDF Unified Robot Description Format. V, 5, 26