ISSN 1870-4069

Planificación de movimientos para robots aéreos no tripulados

Alfredo Reyes M., Abraham Sánchez L., Fabiola Guevara S., Alfredo Toriz P.

Benemérita Universidad Autónoma de Puebla,
Computer Science Department,
México

{reyes-fred,alfredot}@hotmail.com,
abraham.sanchez@correo.buap.mx, fabiola.guevara@outlook.com

Resumen. Durante la última década, diferentes trabajos han mostrado que los algoritmos de planificación de caminos basados en muestreo, como Probabilistic RoadMaps (PRM) y Rapidly-exploring Random Trees (RRT), funcionan bien en la práctica y poseen garantías teóricas, como la completitud probabilística; pero se ha demostrado igual en muchos trabajos que su eficiencia es pobre en la resolución de problemas más complejos, por ejemplo, en el caso de los robots aéreos no tripulados. Este trabajo presenta la aplicación de algoritmos clásicos de planificación de movimientos a los robots aéreos no tripulados. Nuestra principal contribución es utilizar algoritmos más evolucionados tales como: RRG, RRT*, T-RRT, SyCLop entre otros. La planificación de movimientos para robots aéreos no tripulados no es una tarea fácil (dado que involucra calculos de espacios de configuraciones de dimensiones altas, restricciones diferenciales, etc.). Presentamos resultados comparativos y algunos experimentos en escenarios reales.

Palabras clave: vehículo aéreo no tripulado, métodos probabilistas, algoritmos RRT.

Motion Planning for Unmanned Aerial Robots

Abstract. During the last decade, it has been shown that sampling-based path planning algorithms, such as Probabilistic RoadMaps (PRM) and Rapidly-exploring Random Trees (RRT), work well in practice and have theoretical guarantees, such as probabilistic completeness; but it has been shown in many works that its efficiency is poor in the resolution of more complex problems, for example, in the unmanned aerial robots case. This paper presents the application of classic algorithms to the motion planning of unmanned aerial robots. Our main contribution is to use more evolved algorithms such as: RRG, RRT*, T-RRT, SyCLop among others. The motion planning for unmanned aerial robots is not an easy task (since it involves the computing of high configuration spaces, differential constraints, etc.). We present comparative results and some experiments in real scenarios.

Keywords: unmanned aerial vehicle, probabilistic methods, RRT algorithms.

1. Introducción

Un Vehículo Aéreo no Tripulado (UAV: Unmanned Aerial Vehicle) es un dispositivo controlado autónomamente o desde tierra utilizando planes de vuelo programados. Las aplicaciones de este tipo de vehículos es cada día mayor en tareas que implican algún tipo de dificultad o riesgo para vehículos convencionales tripulados por personas como: la detección de incendios, la identificación de manchas de petróleo en el mar, el seguimiento del tráfico, la inspección de líneas de tendido eléctrico, etc. [1].

Actualmente, existe un interés general por el control autónomo de vehículos aéreos no tripulados, ya que en los últimos años se han desarrollado muchos proyectos relacionados con este tema. Algunos, incluso, se ayudan de un sistema de localización, dependiendo de las necesidades del proyecto. Es claro que una aplicación de esta naturaleza, debe incluir un algoritmo de planificación de movimientos y una estrategia para el seguimiento de la trayectoria generada por éste. Por lo tanto el objetivo general de este trabajo es proponer algoritmos para la planificación de movimientos especializados en robots aéreos no tripulados, utilizando la experiencia acumulada en el desarrollo de diferentes proyectos de robótica con los algoritmos RRT (Rapidly-exploring Random Trees) [2].

El problema de la planificación de movimientos, es diseñar un trayecto para el vehículo teniendo en cuenta sus restricciones cinemáticas y dinámicas, así como el generar trayectorias factibles. Las restricciones pueden incluir un radio de giro mínimo y/o límites de velocidad, pero de igual forma podrían existir obstáculos dispersos en el medio ambiente del vehículo que deben tomarse en cuenta. En muchos casos, el modelar la planificación de movimientos como un vehículo de Dubins permite tener una restricción del índice de vuelta, mientras el dispositivo se mueve en un plano. Al extender el modelo de Dubins con el control de altitud, se proporcionan rutas óptimas en tiempo. Considerando los obstáculos, los algoritmos de planificación de movimientos para el vehículo Dubins se proporcionan en [3]. En [4], se propone un algoritmo de planificación de movimientos 3D libre de colisiones para un vehículo aéreo. Cuando se utiliza el modelo de síntesis del auto para un robot que sólo avanza hacia adelante, propuesto por Dubins, la trayectoria resultante se compone de líneas rectas y arcos de un radio de giro mínimo.

En las siguientes secciones, se presentan los algoritmos evolucionados de manera simplificada, aunque el lector interesado podrá si así lo desea, extender su conocimiento al recurrir a la bibliografía sugerida.

2. Planificación de movimientos con drones

Consideremos un drone que se mueve en el espacio y que puede rotar sobre sí mismo, este robot posee la característica de que cuando se encuentra en un mismo

punto del espacio, puede tener distintos estados, dependiendo de la rotación del mismo. Por lo tanto, el espacio de configuraciones asociado a un drone de este tipo viene dado por [5]:

$$\mathcal{X} = \mathcal{R}^3 \times SO(3), \tag{1}$$

donde las coordenadas $(x, y, z) \in \mathcal{R}^3$ determinan la posición del drone en el espacio y las coordenadas $(\theta, \phi, \psi) \in SO(3)$ corresponden con la orientación del drone. Aquí $SO(3)$ denota al grupo especial ortogonal de dimensión 3 (grupo de rotaciones en el espacio) y viene determinado por el conjunto de matrices cuadradas reales ortogonales de orden 3 y con determinante igual a la unidad, es decir:

$$S0(3) = \{A \in \mathcal{M}_3(\mathcal{R})/A^{-1} = A^T, |A| = 1\}. \tag{2}$$

Es conocido que $SO(3)$ es homeomorfo (incluso difeormo) al espacio proyectivo real $\mathcal{R}P^3$.

Sea $\mathcal{X}$ el espacio de configuraciones asociado a un robot aéreo no tripulado $\mathcal{D}$. Un planificador de movimientos del sistema $\mathcal{D}$ se puede describir de forma algorítmica. Consiste en diseñar un programa que tenga como entrada un par ordenado de estados del sistema (A, B), y que tenga como salida un movimiento continuo desde el estado origen A hasta el estado final B. Si consideramos el espacio de configuraciones $\mathcal{X}$ del sistema, el algoritmo se describe como [5]:

- **Entrada**: un par ordenado de puntos $(x, y) \in \mathcal{X} \times \mathcal{X}$ (correspondientes a los estados inicial y final).
- **Salida**: un camino $\alpha : I \to \mathcal{X}$ tal que $\alpha(0) = x$ y $\alpha(1) = y$ (correspondiente al movimiento desde el estado inicial hasta el final).

A partir de ahora supondremos que el espacio $\mathcal{X}$ es conexo por caminos, es decir, para cualquier par de puntos en $\mathcal{X}$ existe un camino que los une. Esta condición no es muy restrictiva. Si el espacio de configuraciones X no fuera conexo por caminos, el planificador de movimientos debería decidir primero si las posiciones x y y pertenecen a la misma componente conexa por caminos de $\mathcal{X}$.

3. Algoritmos de planificación

La planificación de caminos basada en muestreo (PCBM) ha tenido como objetivo tradicional encontrar caminos factibles, es decir, caminos libres de colisiones, para resolver complejos problemas de planificación en espacios de alta dimensión, sin cualquier consideración de la calidad de los caminos. En muchos campos de aplicación, sin embargo, puede ser importante calcular caminos de buena calidad con respecto a un criterio de costo dado (en nuestro caso estamos interesados en el cálculo de caminos óptimos para robots aéreos no tripulados).

Los primeros enfoques propuestos por la comunidad de robótica utilizaban los algoritmos RRT. Los arboles aleatorios de exploración rápida (RRT), es

una técnica desarrollada por Steven M. LaValle y su grupo de colaboradores en la universidad de Illinois, EU, [2,6,7]. La base de estos métodos resulta en un incremento en la construcción de árboles de búsqueda que intentan explorar rápida y uniformemente el espacio de estados, ofreciendo beneficios similares a los obtenidos por otros métodos exitosos de planificación basada en muestreo (como los PRM). Además, los RRTs son, particularmente, convenientes para problemas que involucran restricciones diferenciales. Desafortunadamente, se han aplicado en áreas específicas de la robótica de servicios y algunos de estos métodos sólo se evaluaron en espacios de configuraciones que involucran funciones de costo discretas [8].

El primer enfoque más general para la planificación de caminos considerando el costo-espacio fue el algoritmo RRT basado en transición (T-RRT), que combina la fuerza exploratoria de RRT con un mecanismo de optimización estocástica [9]. T-RRT se ha aplicado con éxito a varios problemas de planificación de caminos de robots, así como problemas de biología estructural [10].

Sin embargo, se ha demostrado que RRT (y por lo tanto T-RRT) no pueden converger hacia una solución óptima [11]. Es por esto que una variante de RRT ofrece una garantía de optimalidad asintótica, que se conoce como RRT*. Sin embargo, se ha observado que RRT* puede converger lentamente en espacios de alta dimensión, y que T-RRT puede proporcionar una solución razonablemente buena más rápidamente. De hecho en la parte de los resultados experimentales, compararemos estos algoritmos en la planificación de movimientos para robots aéreos no tripulados. Cabe aclarar que muchos de estos resultados se aplican a cierto tipo de problemas en robótica, lo que se propone en este trabajo es la aplicación al caso de robots aéreos no tripulados (no necesariamente se aplican estas conclusiones de trabajos previos al tema en cuestión).

3.1. RRT*

El grafo aleatorio de exploración rápida (RRG) se introdujo como un algoritmo incremental (en lugar de por lotes) para construir un roadmap conectado, que posiblemente contenga ciclos. El algoritmo RRG es similar al RRT ya que primero intenta conectar el nodo más cercano a la nueva muestra (vase la figura 1). Si el intento de conexión es exitoso, el nuevo nodo se agrega al conjunto de vértices [11].

El algoritmo RRT* se obtiene apartir de la modificación del algoritmo RRG de tal manera que evita la formación de ciclos, mediante la eliminación de los arcos "redundantes", es decir, los arcos que no forman parte de un camino corto desde la raíz del árbol (el estado inicial) a un vértice. Dado que los grafos de RRT y RRT* son árboles dirigidos con la misma raíz y conjunto de vértices, mientras que los arcos son subconjuntos de RRG. Esto permite la existencia de un "recableado" del árbol RRT, asegurando que los vértices se alcancen a través de un camino de costo mínimo.

El algoritmo RRT*, que se muestra en la figura 2, añade puntos al conjunto de vértices V de la misma manera que RRT y RRG. También considera conexiones desde el nuevo X_{new} a X_{near}, es decir, otros vértices que están dentro de la

RRG
1 $V \leftarrow \{x_{ini}\}$; $E \leftarrow 0$;
2 **para** $k = 1,...,n$
3 $x_{aleat} \leftarrow$ **ConfiguracionLibre**;
4 $x_{proxm} \leftarrow$ **Cercano**(G = (V,E),x_{aleat});
5 $x_{nuevo} \leftarrow$ **Dirige**(x_{proxm}, x_{aleat});
6 **si ObstaculoLibre**(x_{proxm}, x_{nuevo}) **entonces**
7 $x_{prox} \leftarrow$ **Cerca**(G = (V,E),x_{nuevo},min{$\gamma RRG(log(card(V))/card(V))^{1/d}, \eta$});
8 V $\leftarrow$ V $\cup$ {x_{nuevo}}; E $\leftarrow$ E $\cup$ {(x_{proxm}, x_{nuevo}),(x_{nuevo}, x_{proxm})};
9 **Para cada** $x_{nuevo} \epsilon\ x_{prox}$
10 **si LibreColisión**(x_{prox}, x_{nuevo}) **entonces** E $\leftarrow$ E $\cup$ {(x_{prox}, x_{nuevo}),(x_{prox}, x_{nuevo})}
11 **regresa** G = (V,E);

Fig. 1. Algoritmo RRG.

distancia $r(card(V)) = min\gamma_{RRT*}(log(card(V))/card(V))^(1/d), \eta$ de x_{new}. No obstante, no todas las conexiones viables dan lugar a que se introduzcan nuevos arcos en el conjunto E. En particular, (i) crea un arco desde el vértice de X_{near} que puede conectarse a X_{new} a lo largo de un camino con un costo mínimo, y (ii) los nuevos arcos se crean de X_{new} a los vértices en X_{near}, sólo si el camino a través de x_{new} tiene un costo menor que el del padre actual, se suprime el arco que lo une con su padre actual, para mantener la estructura del árbol.

RRT*
1 $V \leftarrow \{x_{ini}\}$; $E \leftarrow 0$;
2 **para** $k = 1,...,n$
3 $x_{aleat} \leftarrow$ **ConfiguracionLibre**$_i$;
4 $x_{proxm} \leftarrow$ **Cercano**(G = (V,E),x_{aleat});
5 $x_{nuevo} \leftarrow$ **Dirige**(x_{proxm}, x_{aleat});
6 **si ObstaculoLibre**(x_{proxm}, x_{nuevo}) **entonces**
7 $x_{prox} \leftarrow$ **Cerca**(G = (V,E),x_{nuevo},min{$\gamma RRT * (log(card(V))/card(V))^{1/d}, \eta$});
8 V $\leftarrow$ V $\cup$ {x_{nuevo}};
9 $x_{min} \leftarrow x_{proxm}$;$c_{min} \leftarrow$ **Costo**(x_{proxm}) + c(**Línea**(x_{prox}, x_{nuevo}));
10 **Para cada** $x_{prox} \epsilon X_{prox}$ //Conecta a lo largo de la ruta con costo mínimo
11 **si LibreColisión**(x_{prox}, x_{nuevo}) $\wedge$ **Costo**(x_{prox}) + (**Línea**(x_{prox}, x_{nuevo})) $< c_{min}$ **entonces**
12 $x_{min} \leftarrow x_{prox}$; $c_{min} \leftarrow$ **Costo**(x_{prox}) + c(**Línea**(x_{prox}, x_{nuevo}))
13 E $\leftarrow$ E $\cup$ {(x_{min}, x_{nuevo})};
14 **Para cada** $x_{prox} \epsilon X_{prox}$
15 **si ColisiónLibre**(x_{nuevo}, x_{prox})$\wedge$ **Costo**(x_{nuevo}) + c(**Línea**(x_{nuevo}, x_{prox})) $<$ **Costo**(x_{prox}) **entonces** $x_{padre} \leftarrow$ **Padre**(x_{prox});
16 E $\leftarrow$ (E $\setminus$ {(x_{padre}, x_{prox})}) $\cup$ {(x_{nuevo}, x_{prox})}
17 **regresa** G = (V,E);

Fig. 2. Algoritmo RRT*.

3.2. RRT basado en transición

T-RRT (figura 3) extiende RRT mediante la integración de una prueba de transición estocástica que permite que el bias realice la exploración hacia las regiones de bajo costo del espacio de configuración. Esta prueba de transición se basa en el criterio de Metrópolis que suele utilizarse en los métodos de optimización de Monte Carlo. Estas técnicas buscan encontrar mínimos globales en espacios complejos y utilizan la aleatoriedad como una técnica para evitar caer en mínimos locales. T-RRT utiliza una prueba de transición para aceptar o rechazar estados candidatos, la prueba está basada en la variación de costos asociada con el movimiento local. El pseudo-código de T-RRT es similar al de la extensión básica RRT, con la adición de las funciones PruebaTransición y ControlRefinamiento (figura 4).

T-RRT
entrada : configuración del espacio C ; la fun
c : C → R $_+$; la raíz x_{ini} ; la meta x_{met}
salida: el árbol $\mathcal{T}$
1 $\mathcal{T} \leftarrow$ iniÁrbol(x_{ini})
2 **mientras no** CondiciónDeParo($\mathcal{T}, x_{meta}$)
3 $x_{aleat} \leftarrow$ ESTADO_ALEATORIO(C)
4 $x_{prox} \leftarrow$ VECINO_MAS_PROXIMO($\mathcal{T}, x_{al}$
5 **si** ControlRefinamiento($\mathcal{T}, x_{prox}, x_{aleat}$) **en**
6 $x_{nuevo} \leftarrow$ EXTENDER(x_{prox}, x_{aleat})
7 **si** $x_{nuevo} \neq$ null
8 **y** PruebaTransición($\mathcal{T}, c(x_{prox}), c(x_{nuevo}$
9 agregaNuevoVerticeyArista($\mathcal{T}, c(x_{prox}),$

Fig. 3. Algoritmo T-RRT.

PruebaTransición($\mathcal{T}, c_i, c_j$)
entrada : el costo límite c_{max} ; la temperatura actual $\mathcal{T}$;
salida : *verdadero* si la transición es aceptada, *falso* en otro caso
Tasa de aumento de temperatura $\mathcal{T}_{tasa}$
1 **si** $c_j > c_{max}$ **entonces regresa** Falso
2 **si** $c_j <= c_i$ **entonces regresa**
3 **si** exp(- ($c_j - c_i$) / $\mathcal{T}$) > 0.5 **entonces**
4 $\mathcal{T} \leftarrow \mathcal{T}/2^{(c_j - c_i)/(0,1 * costoRango(\mathcal{T}))}$; **regresa** Verdadero
5 **sino**
6 $\mathcal{T} \leftarrow \mathcal{T} * 2^{\mathcal{T}_{tasa}}$ **regresa** Falso

Fig. 4. Función PruebaTransición.

La función PruebaTransición presentada en el algoritmo es una versión mejorada de propuestas previamente desarrolladas. Se utiliza para evaluar la transición

de q_{new} sobre la base de sus respectivos costos. Tres casos son posibles: 1) una nueva configuración cuyo costo es superior al valor del umbral c_{max} se rechaza automáticamente. 2) se acepta una transición correspondiente a un movimiento descendente. 3) las transiciones ascendentes se aceptan o rechazan con base a una probabilidad que disminuye exponencialmente con la variación de costo $c_j - c_i$, similarmente al criterio de Metrópolis. En este caso el nivel de dificultad de la prueba de transición es controlado por el parametro adaptativo T, llamado temperatura sólo por analogía con la física estadística. Las bajas temperaturas limitan la expansión a pendientes suaves, y las altas temperaturas permiten escalar pendientes pronunciadas. En T-RRT, la temperatura se sintoniza dinámicamente durante el proceso de búsqueda: 1) después de cada transición ascendente aceptada, T se disminuye para evitar la sobreexploración de regiones de alto costo. 2) después de cada transición cuesta arriba rechazada, T se incrementa para facilitar la exploración y evitar ser atrapado en un mínimo local.

ControlRefinamiento($\mathcal{T}, x_{prox}, x_{aleat}$)
entrada : la extensión del paso δ ; el radio de refinamiento p
salida: *verdadero* si el refinamiento no es tan alto, *falso* en otro caso
1 **si** distancia(x_{prox}, x_{aleat}) $< \delta$
2 **y** nbNodosRefinamiento($\mathcal{T}$) > p * nbNodos($\mathcal{T}$) **entonces**
3 **regresa** Falso
4 **regresa** Verdadero

Fig. 5. Función PruebaTransición.

El ajuste adaptativo de la temperatura asegura una tasa de éxito dada para las transiciones ascendentes, pero también puede producir un efecto secundario no deseado: T puede ser reducido por la aceptación de nuevos estados cercanos a los estados ya contenidos en el árbol, mientras que puede ser necesario un aumento de T para recorrer una barrera de costos locales y explorar nuevas regiones del espacio. Aceptar tales estados sólo contribuye a refinar la exploración de regiones de bajo costo y alcanzadas por el árbol. El objetivo de la función controlRefinamiento es limitar este refinamiento y facilitar la expansión del árbol hacia regiones inexploradas. La idea es rechazar una expansión que conduzca a un mayor refinamiento si el número de nodos de refinamiento ya presente en el árbol es mayor que una cierta relación p del número total de nodos, definiéndose un nodo de refinamiento como un nodo cuya distancia a su padre es menor que la extensión tamaño a δ. Otro beneficio del control del refinamiento es limitar el número de nodos en el árbol, y así reducir el costo computacional de la búsqueda de vecindad más cercana [12].

3.3. SyclopRRT

SyClop es un framework multicapas, que combina la busqueda discreta y la planificación de movimientos basada en muestreo. El efecto general es que SyClop mejora significativamente la eficiencia computacional del planificador de movimiento basado en el muestreo subyacente [13]. Las ventajas proporcionadas por SyClop se hacen aún más pronunciadas cuando se consideran los problemas de planificación de movimientos de alta dimensión con la dinámica. El objetivo de la búsqueda discreta en SyClop es proporcionar un sentido global de dirección e identificar secuencias de regiones que el planificador de movimientos basado en muestreo, puede muestrear y explorar selectivamente para avanzar significativamente en la exploración de árboles. SyClop mantiene información no sólo sobre las regiones de descomposición sino también sobre los arcos de descomposición. Esta información que se actualiza después de cada paso de exploración, mide el progreso general y se utiliza para guiar eficazmente la exploración basada en árboles.

3.4. Algunos comentarios sobre los planificadores

El estado del arte actual en la planificacion de movimientos requiere una mejora particularmente en términos de precisión, eficiencia, robustez y optimización del camino obtenido. La planificación óptima de caminos es un problema desafiante y para las aplicaciones de planificación en línea, la convergencia al camino óptimo es aún más importante. Es complicado establecer cual planificador es más óptimo que otro, porque se deberían realizar muchas pruebas experimentales, un excelente trabajo sobre uno de los algoritmos, RRT*, se puede consultar en [14]. Igualmente por motivo de espacio en este trabajo, no se detallan los otros algoritmos, por ejemplo RRT-Connect, Lazy RRT. Pero su descripción se puede consultar en la bibliografía sugerida.

4. Resultados experimentales

A continuación presentamos los resultados obtenidos al utilizar los algoritmos descritos anteriormente. Primero que nada, presentamos el robot aéreo no tripulado y posteriormente las librerias que utilizamos.

El AR.Drone 2.0 es un cuadricóptero desarrollado por Parrot. Su estructura está conformada por cuatro motores unidos en forma de cruz donde el hardware de radio frecuencia y la batería se encuentran en el centro. Cada par de motores opuestos giran de la misma forma. Un par gira en el sentido de las manecillas del reloj y el otro en el sentido contrario.

El AR.Drone 2.0 puede ser controlado por cualquier dispositivo que soporte WiFi. El control del AR.Drone 2.0 es realizado por tres servicios de comunicación principales. El control y la configuración del drone son realizados por medio del envío de comandos AT en el puerto UDP 5556. La latencia de transmisión de los comandos de control son críticos para la experiencia del usuario. Estos comandos

tienen que ser enviados de manera regular (usualmente 30 veces por segundo). La información acerca del drone como el estado, su posición, velocidad de rotación de los motores, llamado navdata (de navigation data), son enviados por el drone a su cliente en el puerto UDP 5554.

La secuencia de vídeo es enviada por el AR.Drone 2.0 al dispositivo cliente por el puerto TCP 5555. Otro canal de comunicación, llamado control port, puede ser establecido por el puerto TCP 5559 para enviar información crítica, en oposición a los otros canales de comunicación donde la información puede ser perdida sin efectos peligrosos. Es utilizado para recuperar datos de configuración, y el envío de configuraciones.

Se utilizaron las siguientes librerías:

- ROS (Robot Operative System) es un framework para escribir software para robots de código abierto. Es un conjunto de herramientas, bibliotecas, y convenios que tienen por objetivo simplificar la tarea de crear comportamientos complejos y robustos a través de una amplia variedad de plataformas robóticas.
- Ardrone autonomy es un controlador de ROS para el cuadricóptero AR.Drone 1.0 y 2.0. Este controlador está basado en el SDK del AR.Drone. Es un controlador desarrollado por el grupo Autonomy Lab de la Universidad Simon Fraser.
- TUM simulator es un paquete que contiene la implementación de un simulador de Gazebo para el AR.Drone 2.0 escrito por Hongrong Huang y Juergen Sturm del Grupo de Visión por Computadora en la Universidad Técnica de Munich. Este paquete está basado en el paquete ROS llamado tumdarmstadtrospkg de Johannes Meyer y Stefan Kohlbrecher y el simulador de AR.Drone que es proporcionado por Matthias Nieuwenhuisen. El simulador puede trabajar tanto con el AR.Drone 1.0 y 2.0.
- Open Motion Planning Library (OMPL) consiste en diversos algoritmos de planificación de movimientos basados en muestreo. La biblioteca está diseñada para que pueda ser fácilmente integrada en sistemas que proporcionan los componentes adicionales necesarios. OMPL.app, es el front-end de OMPL, contiene un sistema para el control de colisión FCL Y PQP, ademas una interfaz basada en PyQt/PySide. El front-end se puede utilizar para la planificación de movimientos de cuerpos rígidos y algunos tipos de vehículos (de primer orden y de segundo orden, un dirigible y un quadrotor). Se basa en la biblioteca Assimp para importar una gran variedad de formatos de modelado que se pueden utilizar para representar al robot y su entorno.

A través del framework ROS se utilizó la herramienta de OMPL app la cual cuenta con un entorno gráfico en donde podemos observar el procesamiento de estos algoritmos. Se realizaron 10 corridas por cada una de las 6 configuraciones de los diferentes parámetros del planificador, esto se hizo para cada uno de los 6 algoritmos utilizados en 3 escenarios diferentes (ver la siguiente figura), obteniéndose 1080 corridas en total. A continuación la siguiente tabla 1 presenta el porcentaje de eficiencia que tuvo cada algoritmo en cada uno de los escenarios,

es decir el promedio por las 60 corridas. Posteriormente se detalla algún algoritmo en los diversos escenarios.

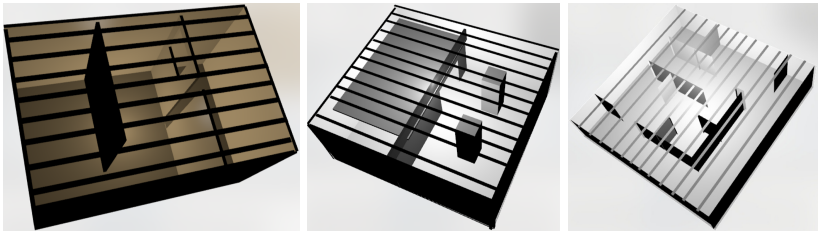

Fig. 6. Escenarios de pruebas.

Tabla 1. Porcentaje de eficacia de los algoritmos en los 3 diferentes escenarios.

Escenario	RRT	RRT-Connect	RRT*	T-RRT	Lazy RRT	SyclopRRT
1	86 %	80 %	85 %	50 %	65 %	10 %
2	91.6 %	83.3 %	76.6 %	83.3 %	0 %	11.6 %
3	96.6 %	95 %	100 %	83.3 %	81.6 %	8.3 %

Ciertos algoritmos, a pesar de que en determinados escenarios tuvieron una respuesta poco favorables, tuvieron buen desempeño, nuevamente la dificultad en el escenario es la que nos permite tener resultados diversos. Uno de los aspecto que fue posible observar dentro de estas pruebas fue la complejidad en uno de los escenarios que contiene dos niveles, varios de los algoritmos tuvieron algunas dicultades para llegar al estado final. De igual forma la configuración inicial y final del drone fue determinada apartir de un análisis previo, en donde se trató de colocar al drone en las posiciones más dificiles.

De igual forma dentro de estos algoritmos existe un aprendizaje, esto es posible observarlo en el tiempo de cómputo, ya que en las primeras ejecuciones éste es alto, pero va reduciendo conforme pasan las siguientes ejecuciones. Las configuraciones que se manejaron dentro de estos algoritmos fueron determinadas teniendo en cuenta el impacto hacia el algoritmo.

Se propusieron 6 diferentes configuraciones para obsevar que tan bueno o malo resultó el algoritmo en el escenario. Esto es posible observarlo en nuestros resultados, al modificar estos parámetros los algoritmos tuvieron mejores resultados, o bien en algunos casos no pudieron realizar la planficación dentro del tiempo limite.

A continuación se muestran los resultados para el algoritmo RRT* (dado que fue el algoritmo que tuvo mejor desempeño) en el escenario 1. La figura 7

muestra el resumen de los parámetros que se utilizan en el algoritmo RRT* y en la siguiente tabla se muestra el rendimiento del algoritmo en el mismo escenario.

Tiempo Max.(sec.)	TM
Verificación de Colisión	VC
Verificación de Colisión Tardada	VCT
Búsqueda de Foco	BF
Bias Meta	BM
Muestreo Informado	MI
Rechazo de Nuevo Estado	RNE
Num. de Intentos de Muestreo	NIM
Umbral de Poda	UP
Medida de Poda	MP
Rango	R
Factor de Rewire	FR
Rechazo de Muestras	RM
Poda del Árbol	PA
Uso de Heurísitca Admisible	UHA
Uso k-nearest	UKN

Fig. 7. Resumen de parámetros de RRT*.

Tabla 2. Rendimiento del algoritmo RRT* en el escenario 1.

TM	VC	VCT	BF	BM	MI	RNE	NIM	UP	MP	R	FR	RM	PA	UHA	UKN
60	0.010	Sí	Sí	0.05	No	No	10	0.05	No	0	1.10	No	No	Sí	Sí

Tiempo CPU (seg.)	Costo	Estados interpolados	Conectado
0.5473	2258	114	60 %

Es posible observar que los resultados dependen fuertemente de las configuraciones del algoritmo, puesto que estas determinan que tan eficiente realizará la planificación. En cada uno de los algoritmos, se puede determinar, el rango que tendrá el crecimiento de las ramas, la verificación de colisión, el bias que se quiere lograr, etc. Dentro de las corridas es posible observar que apesar que unos algoritmos tengan un mayor número de nodos el movimiento del drone es mucho más suave que otros, es decir su vuelo se realiza de mejor manera que otros con menos nodos. En la figura 8 se muestra el resultado del algoritmo RRT* en un escenario de prueba.

Los resultados en tiempo son muy variados ya que algunos algoritmos nos presentan resultados en tiempos muy mínimos, alrededor de milisegundos, pero algunos otros llegan hasta minutos sin poder resolver el problema de planficación. Unos de los mejores planificadores fue el algoritmo RRT simple que nos devolvió resultados en tiempo muy cortos, con un número de nodos reducidos. El camino obtenido para el drone no contenía muchos ciclos, pero no se podría considerar la

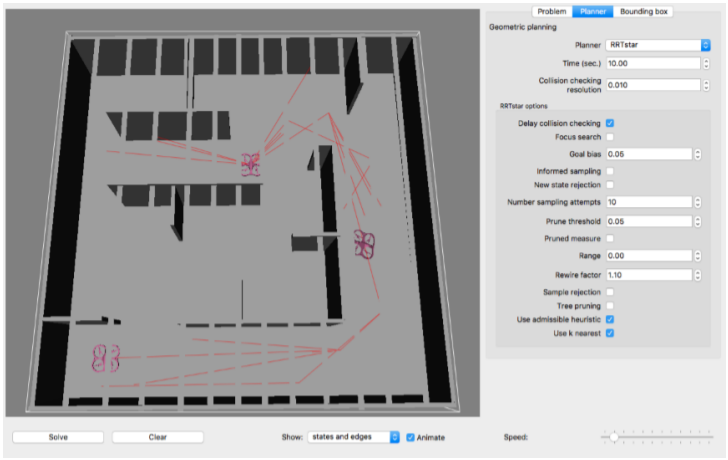

Fig. 8. Ejecución del algoritmo RRT* en el escenario 3.

óptima debido a que su vuelo no era del todo limpio, en cambio a pesar de que no siempre se llegó a tener resultados perfectos el algoritmo RRT*, brindaba al drone una trayectoria mucho más suave y fluida, lo que en pruebas reales ayudaría mucho más al AR Drone 2.

El software OMPL app al ser de código abierto, permite modificar su estructura para realizar cambios a nuestras necesidades. Uno de los requerimientos para llevar a cabo nuestra investigación, es el obtener un plan de vuelo para proveerselo al AR Drone 2. Obtenemos un archivo de texto simple que incluye las posiciones del drone. Una vez que se obtiene este archivo, es necesario modificar su entorno para que el drone sea capaz de poder interpretarlo y ejecutarlo. El sistema ROS a través de TUM_Simulator nos brinda un entorno de simulación para el AR Drone 2, dentro de este podemos crear nuestro escenario con los elementos básicos que nos provee la herramienta. Es necesario integrar el software TUM Ardrone al simulador para poder intervenir al drone con nuestro plan de vuelo previamente generado por nuestro compilador.

El equipo utilizado para realizar las pruebas de la implementación real fue el mismo que se usó para las pruebas de la versión simulada. El mapa del ambiente representará el espacio donde se desplazará el AR Drone 2. La figura 9 muestra una secuencia de imágenes del vuelo del AR Drone en un ambiente real a través de una planificación de movimientos.

Una vez ejecutadas las pruebas anteriores en el modelado del escenario real, se tradujeron las coordenadas a un plan de vuelo capaz de ser entendido por el AR.Drone mediante el software TUM ARdrone. En esta prueba queremos observar el comportamiento que muestra el cuadricóptero en un ambiente real, donde no es posible controlar diversos factores, como por ejemplo corrientes de aire. Los planes de vuelo se obtuvieron de las mejores ejecuciones previamente detalladas. Es importante mencionar que para estas pruebas se modificaron las

Fig. 9. Secuencia de imágenes del vuelo autónomo del AR Drone, utilizando el algoritmo RRT*.

velocidades de los motores del AR.Drone, con el propósito de observar si es posible realizar vuelos efectivos a diversas velocidades. El seguimiento de caminos se implementó con un control simple PID que no se detalla en este trabajo.

El AR.Drone tuvo un eficiencia alta, ya que se desempeñó siguiendo el plan de vuelo marcado por el algoritmo de forma correcta y respetando las restricciones del ambiente, es decir volando dentro de las medidas del escenario. El AR.Drone con el algoritmo RRT* tuvo un vuelo más eficiente, es decir, presentó menos movimientos bruscos e innecesarios en su trayectoria que otros algoritmos. La planificación por parte de los algoritmos se mantuvo un en un rango de milisegundos y el vuelo del drone en ejecución fue alrededor de 3 a 6 minutos. Este último tiempo es variante debido a que se pueden modificar las velocidades de los motores del AR.Drone, es necesario tener en cuenta que esto puede perjudicar su vuelo.

5. Conclusiones y trabajo futuro

Este trabajo tuvo como objetivo el uso de algoritmos de planficación de movimientos especializados en robots aéreos no tripulados para el vuelo autónomo. Los resultados ofrecidos por estos algoritmos cumplen con el objetivo que se persigue en este proyecto, debido a que es posible observar el comportamiento de diversos algoritmos basados en las técnicas antes mencionadas. Los algoritmos propuestos consideran la cinemática del drone, el resultado que muestre cada algoritmo, es estrictamente dependiente de su configuración, puesto que al ingresar parámetros aleatorios la efectividad puede incrementar o decrementar, al igual que el tiempo de ejecución. Esta situación se resolvio a través de casos de pruebas que permitieron determinar qué parámetros nos dan los resultados óptimos. Dentro de los escenarios propuestos se consideraron diversas situaciones en donde el quadricóptero pudiera tener mayor dificultad para realizar su vuelo, lo que implica mayor complejidad en la planificación de movimientos.

El algoritmo RRT* fue quien nos generó resultados adecuados al no fallar en ninguna ejecución, además de otorgarnos tiempos de ejecución mínimos. Una

vez generado el plan de vuelo es importante tener en claro las escalas debido a que puede presentarse un problema en la transformación del plan de vuelo. Fue necesarió la creación de un compilador que nos ayudo a realizar estos cambios y ejecutarlo en el AR Drone para su vuelo autónomo.

Por supuesto que el trabajo es preliminar y da pauta a trabajos futuros, podemos mencionar algunos puntos:

- Uno de los aspectos más importantes en la navegación en tiempo real sera la estabilidad del drone (cuando hay viento fuerte en exteriores). Quizás implementar un ciclo de control podría resolver este inconveniente.
- Utilizar un algoritmo basado en RRT para la planificación de movimientos en tiempo real, que permita generar nuevas trayectorias dependiendo de los obstáculos imprevistos que se pudieran presentar.
- Implementación de un middleware para extender la compatibilidad con otros tipos de quadricópteros.

Referencias

1. Villena, M.E.: El uso de vehículos aéreos no tripulados (drones) en las labores de seguridad y vigilancia de la administración. SICARM (2014)
2. LaValle, S.M., Kuffner, J. J.: Rapidly-exploring random trees: Progress and prospects. Algorithmic and Computational Robotics: New Directions, pp. 293–308 (2011)
3. Agarwal, P., Wang, H.: Approximation algorithms for curvature-constrained shortest paths. SIAM Journal on Computing 30(6), pp. 1739–1772 (2001)
4. Snape, J., Manocha, D.: Navigating multiple simple-airplanes in 3d workspace. In: Proc. of the IEEE International Conference on Robotics and Automation (ICRA), pp. 3974–3980 (2010)
5. Laarbi-Fumero, D.: La complejidad topológica del planificador de movimientos robótico. Tesis de la Universidad de la Laguna (2016)
6. LaValle, S.M.: Planning Algorithms. Cambridge University Press (2006)
7. LaValle, S.M., Kuffner, J.J.: Randomized kinodynamic planning. International Journal of Robotics Research 20(5), pp. 378–400 (2001)
8. Urmson, C., Simmons, R.: Approaches for heuristically biasing RRT growth. In: Proc. of the IEEE/RSJ IROS (2003)
9. Jaillet, L., Cortés, J., Siméon, T.: Sampling-based path planning on configuration-space costmaps. IEEE Transactions on Robotics 26(4), pp. 635–646 (2012)
10. Jaillet, L., Corcho, F.J., Pérez, J.J., Cortés, J.: Randomized tree construction algorithm to explore energy landscapes. Journal of Computational Chemistry 32(16), pp. 3464–3474 (2011)
11. Karaman S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. Robot. Research 30(7), pp. 846–894 (2011)
12. Devaurs, D., Siméon, T., Cortés, J.: Enhancing the Transition-based RRT to deal with complex cost spaces. In: Proc. of the IEEE International Conference on Robotics and Automation (2013)
13. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Motion planning with dynamics by a synergistic combination of layers of planning. IEEE Transactions on Robotics 26(3), pp. 469–482 (2010)

14. Noreen, I., Khan A., Habib, Z.: Optimal path plannning using RRT* based approaches: A survey and future directions. International Journal of Advanced Computer Science and Applications 7(11), pp. 97–107 (2016)