Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Métodos reactivos basados en campos potenciales para UAVs

Autor: Carmen Mera Prieto
Tutores: Aníbal Ollero Baturone
José Antonio Cobano Suárez

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

Sevilla, 2015

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Métodos reactivos basados en campos potenciales para UAVs

Autor:
Carmen Mera Prieto

Tutores:
Aníbal Ollero Baturone
José Antonio Cobano Suárez

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Trabajo Fin de Grado:Métodos reactivos basados en campos potenciales para UAVs

Autor: Carmen Mera Prieto

Tutores: Aníbal Ollero Baturone
José Antonio Cobano Suárez

El tribunal nombrado para juzgar el proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A mis padres y hermana por estudiar una ingeniería sin estar matriculados.

A todos mis amigos, a aquellos con los que ya venía y a los nuevos que se fueron incorporando haciendo que mis "etsidías" fueran felices.

Agradecimientos

A Jenifer Delgado Rich por su paciencia, apoyo incondicional y amistad en este último año. A Lucas Rodríguez Díaz por aportarme algo de su sensatez, por su compañerismo y aguante. A José Antonio Cobano Suárez por guiarme cuando me encontraba más perdida. Finalmente, a Aníbal Ollero Baturone por introducirme en el mundo de la robótica e inspirar una nueva pasión en mí.

Resumen

Este trabajo presenta un método reactivo para evitar colisiones de vehículos aéreos no tripulados (Unmanned Aerial Vehicles, UAVs, en inglés) mofificando la trayectoria. El método reactivo implementado está basado en campos potenciales. El objetivo es que cualquier vehículo aéreo no tripulado (UAV) alcance las posiciones deseadas desde una posición de inicio y siguiendo un plan de vuelo, evitando las posibles colisiones con el entorno que se detecten durante el vuelo.

Se ha realizado una primera implementación en Matlab considerando problemas en dos dimensiones y posterioremente una implementación en C++ considerando tres dimensiones. Para esta última se ha utilizado un framework llamado Sistema Operativo Robótico (Robot Operating Systems, ROS, en inglés) muy útil para el desarrollo de software para robots. La detección de obstáculos en el entorno se hace a partir de las lecturas de un láser Hokuyo UTM 30 LX que va montado en un servo Dynamixel y gira un ángulo determinado para calcular la nube de punto 3D del entorno.

ROS (ROS) nos facilita librerías, visualizadores, gestión de mensajes y paquetes de datos entre otras. Con ella conseguimos olvidarnos del hardware y realizar simulaciones bastante acordes con la realidad.

Este trabajo forma parte del proyecto Aerial Robotics Cooperative Assembly System (ARCAS) cuyo propósito el desarrollo y validación experimental del primer sistema robótico cooperativo para ensamblaje y construcción de estructuras en lugares poco accesibles.

Abstract

This project presents a reactive approach to prevent collisions of UAVs (Unmanned Aerial Vehicles) by modifiying the path. The reactive method implemented is based on potential fields. The goal is that any Unmanned Aerial Vehicle (UAV) reaches the desired positions from a starting point and following a flight plan, avoiding possible collisions with the environment which will be detected during the flight.

There has been a first implementation in Matlab considering problems in two dimension and, after that, an implementation in C++, this time, considering three dimensions. For this last one, we used a framework called Robot Operating System (ROS), a very useful tool for the development of robots softwares. The detection of obstacles in the environment is done by using the readings of a Hokuyo UTM 30 LX laser which is coupled to a Dynamixel servo and rotates at a certain angle to determinate the 3D point cloud environment.

ROS will facilitates us with libraries, displays, message management and data packs among others. With it, we get to forget about the hardware and perform simulations quite consistent with reality.

This project is part of the Aerial Robotics Cooperative Assembly System (ARCAS) whose purpose is the development and experimental validation of the first cooperative robotic system for assembly and construction of structures in places where the acces is limited.

Índice

Índice de Figuras

1. INTRODUCCIÓN

Si quieres cambio verdadero, pues camina distinto.

-René Pérez Joglar -

La planificación de trayectoria es uno de los temas de estudio de mayor interés actual. Esto se debe a la infinidad de aplicaciones que nos pueden ofrecer los UAV. Estas aplicaciones de las que hablamos se ven limitadas ante la necesidad de mayor autonomía [2].

Las tareas que van desempeñando los UAV tienen cada vez una mayor complejidad en lo que a las maniobras respecta, sobre todo en las aplicaciones en tiempo real donde métodos de planficiación local y reactivos son necesarios para evitar colisiones que se pueden detectar debido a eventos no esperados. Por tanto, la implementación de este tipo de métodos para llevar a cabo las misiones con UAVs de una manera segura juega un papel importante.

1.1. Antecedentes

Para planificar una trayectoria un ser humano tiene en cuenta muchos factores, de muchos de ellos ni siquiera somos conscientes, por eso hay veces que no tomamos el camino acertado. Esta serie de decisiones se basa en nuestra percepción humana.

Para la planificación de trayectoria de un UAV ya nos encontramos ante un sistema de percepción automático que una vez ha obtenido a través de algún sensor información del entorno, trata los datos, en este caso, usando diferentes métodos matemáticos. Estos nos facilitarán una ruta.

Comencemos haciendo una división entre una planificación global o local. La planificación de trayectoria global aparece en entornos conocidos. Podríamos decir que tenemos un mapa y éste será estático. Como nuestro escenario no se va a ver sometido a ningún cambio, sabemos cual es la trayectoria óptima desde el comienzo. Por ejemplo, ir desde Sevilla a Madrid. Nuestro algoritmo tomaría las carreteras que hacen que estas dos ciudades se unan cumpliendo el criterio que hayamos seleccionado previamente, por ejemplo, menor distancia. No existirá ningún cambio en la ruta.

Por el contrario, en la planificación de trayectoria local tan solo conocemos un escenario próximo a nuestro robot. Es decir, tendríamos conocimiento del entorno que nos rodea pero dentro de un margen de unos pocos metros, dependiendo del alcance de los sensores utilizados. Debido a esto no podemos crear una trayectoria completa desde el inicio de nuestra búsqueda, sino que obtendremos un punto intermedio o *waypoint* que será la posición más favorable tal y como se encuentra el mapa justo en ese instante. En este caso saldríamos desde Sevilla e iríamos obteniendo posiciones intermedias hasta llegar a nuestro destino, imaginemos que tenemos conocimiento de un mapa a 200 km, nuestro planificador nos destinaría a la posición intermedia más

favorable, por ejemplo Mérida, una vez vamos avanzando iremos obteniendo información nueva de nuestro entorno y recalcularemos nuestro punto más favorable. Si por ejemplo una carretera estuviera cortada podríamos valorarlo para así no tomar esa vía. Finalmente, iríamos recorriendo estas posiciones intermedias hasta llegar a nuestro punto objetivo.

Esta característica hace que tengamos un método reactivo, es decir, que va a actuar debido a los acontecimientos vayan sucediendo. Estos métodos generan una respuesta rápida a información del entorno que obtenemos de sensores.

Ambas corrientes son útiles, todo depende del objetivo que persigamos. La diversidad de planificadores se debe a que no existe ningún planificador ideal [2]. Actualmente, lo único que podemos hacer es seleccionar la estrategia más adecuada a nuestra aplicación e introducirle mejoras que veamos necesarias. El objetivo debe ser conseguir el planificador de trayectoria que mejores resultados nos dé en cuestiones de fiabilidad, robustez, eficiencia, seguridad y baja carga computacional.

En nuestro trabajo vamos a centrarnos en el planificador reactivo de los campos potenciales o *potential fields*. Éste selecciona el *waypoint* más favorable para evitar una posible colisión en función de los valores de un campo de fuerzas creado con fuerzas atractivas que serán las distancias a nuestro punto objetivo y fuerzas repulsivas que serán aquellas que provocan los obstáculos.

1.2. Proyecto ARCAS

Como ya hemos adelantado, ARCAS [18] es un proyecto europeo cuyo objetivo es el desarrollo y experimentación de la cooperación entre robots aéreos para la construcción de estructuras. La importancia de este proyecto se ve claramente reflejada en las misiones que se persiguen. Entre otras podemos destacar la construcción de plataformas para la evacuación de personas en operaciones de rescate, también para la instalación de plataformas en obstáculos irregulares y otros servicios más industriales como en las operaciones de mantenimiento y construcción de estructuras.

Este proyecto se estructura en cuatro partes esenciales:

- Nuevos métodos para el control de robots aéreos con manipuladores. También para la coordinación de varios robots aéreos en contacto con el mismo objeto.
- Mejorar los métodos de percepción de nuestros robots: modelo, identificación y reconocimiento del escenario en el que estamos.
- Nuevos métodos de cooperación entre robots aéreos para la construcción de estructuras.
- Estrategias para operadores de asistencia, con retroalimentación visual y de fuerza.

Figura 1-1. Ejemplo de coordinación de varios robots aéreos en contacto con el mismo objeto [18].

1.3. Organización del proyecto

Este proyecto se divide en seis capítulos:

- **Capítulo 1.** Introducción:

  Presenta una visión global de la meta del trabajo y el problema abordado. También se presenta el proyecto ARCAS.

- **Capítulo 2.** Estado del arte o *state of art*:

  Todo tema de investigación tiene un estado del arte, por tanto se hace breve repaso por los métodos de planificación de trayectoria que nos parecen más útiles. Se muestra una clasificación de estos métodos y se comentarán sus características principales.

- **Capítulo 3.** ROS:

  Una vez estudiadas las características fundamentales de esta herramienta de software, se hace un breve resumen de sus conceptos más importantes. Se comentan las ventajas de los archivos .launch y el visualizador 3D que se ha utilizado.

- **Capítulo 4.** Descripción del método implementado:

  Es aquí donde se encuentra el *grosso* del trabajo, detallando las partes del algoritmo implementado. Se presenta el pseudocódigo del algoritmo centrándonos con más profundidad en aquellas cuestiones que durante el trabajo supusieron un problema, para que el lector pueda con facilidad enfrentarse a éstos si quisiera.

- **Capítulo 5.** Experimentos:

  Se presentan los diferentes escenarios y situaciones considerados para probar el algoritmo presentado en este trabajo. En ellas demostraremos que el robot mantiene siempre un comportamiento acertado y, por lo tanto, seguro.

- **Capítulo 6.** Conclusiones y trabajo futuro:

  Es necesaria una reflexión final del estudio realizado en los últimos meses. También se muestran las próximas metas en el trabajo, siendo este proyecto una introducción al mundo de la planificación. Y habiendo tan solo llegado a la primera de muchas paradas.

2 ESTADO DEL ARTE

Para unos, la vida es galopar un camino empedrado de
horas, minutos y segundos.
Yo, más humilde soy, y sólo quiero que la ola que surge
del último suspiro de un segundo,
me transporte mecido hasta el siguiente

-Roberto Iniesta Oiea-

El objetivo de este capítulo es exponer la gran variedad de estrategias de planificación presentes en la literatura. Todas han sido motivo de estudio desde hace años, pues no se ha conseguido obtener el método definitivo.

Cada método de planificación tiene una serie de ventajas y desventajas y es su aplicación lo que nos hace decantarnos por uno en concreto. Para comenzar haremos una pequeña introducción sobre los pasos que se siguen cuando se ejecuta un método de planificación y posteriormente nos centraremos en recorrer todas las opciones que se nos ofrece.

2.1. Método de planificación de trayectoria

Las siguientes secciones describen los pasos que siguen los diferentes métodos [6]:

2.1.1. Modelo y representación del terreno

El sensor utilizado detectará los obstáculos que existan en el entorno en su radio de alcance. La Figura 2-1 muestra los obstáculos con puntos azules utilizando Matlab. El punto inicial sería la cruz azul y el punto al que queremos ir la cruz roja. En este caso el mapa del entorno es conocido.

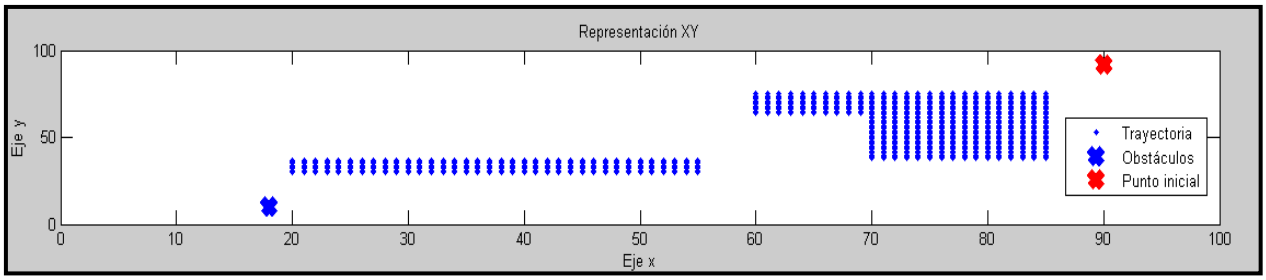

Figura 2-1. Modelo en Matlab

Se ha detectado uno de nuestros obstáculos (ver Figura 2-2). En este caso se señaliza con puntos en marrón. El resto, de azul, simboliza un espacio libre de obstáculos

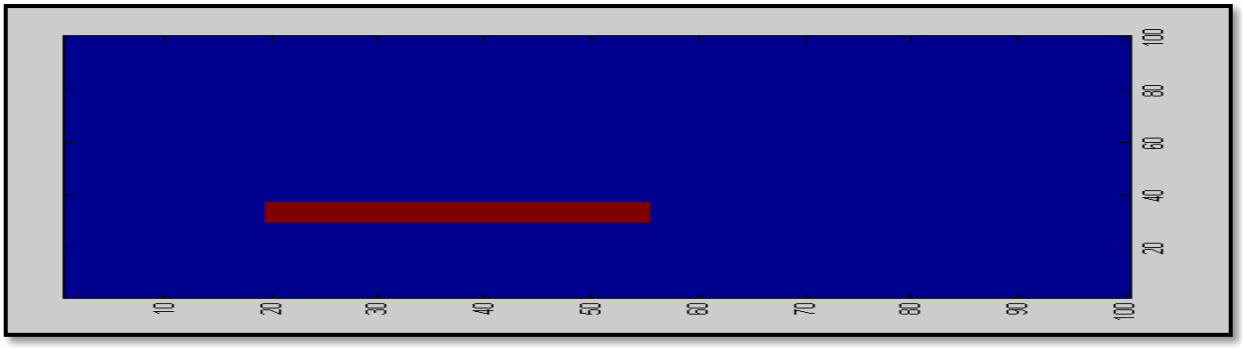

Figura 2-2. Representación del terreno

2.1.2. Generación del mapa

Este es el punto que hace diferente un método de otro, más adelante veremos las gran cantidad de posibilidades que existen. Como ejemplo, consideramos los campos potenciales, que es el método usado en el trabajo.

La Figura 2-3 representa el campo de fuerzas generado. El punto inicial está representado con una cruz azul y el punto final con una cruz roja. El resto es el campo de fuerzas. Cada casilla tiene un valor de potencial. Los colores más oscuros son aquellos con menor valor de campo y a los que nos dirigiremos. Se puede observar cómo el campo de fuerzas va decreciendo de forma uniforme a medida que la distancia al objetivo es menor, exceptuando la zona donde se encuentra el obstáculo, que produce un incremento en su valor.

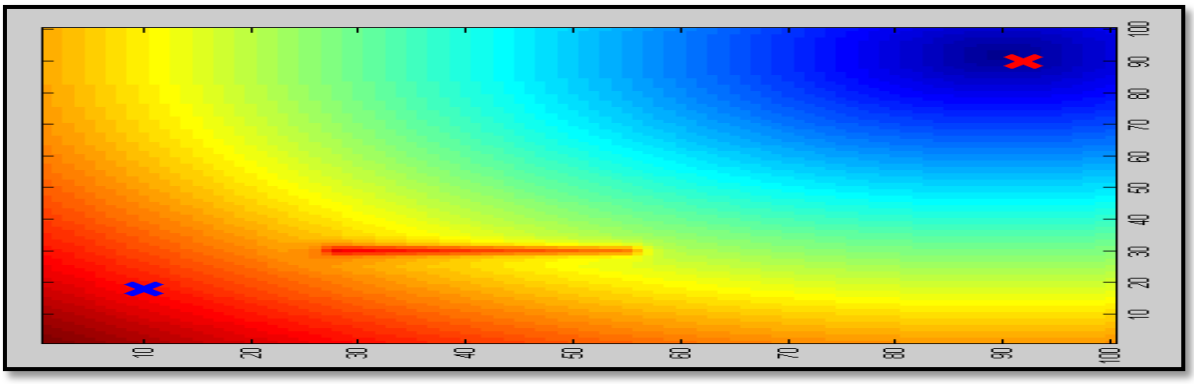

Figura 2-3. Campo de fuerzas del mapa

2.1.3. Búsqueda y generación de trayectoria

Gracias a nuestra estrategia conseguimos calcular la trayectoria óptima. La línea roja representa dicha trayectoria.

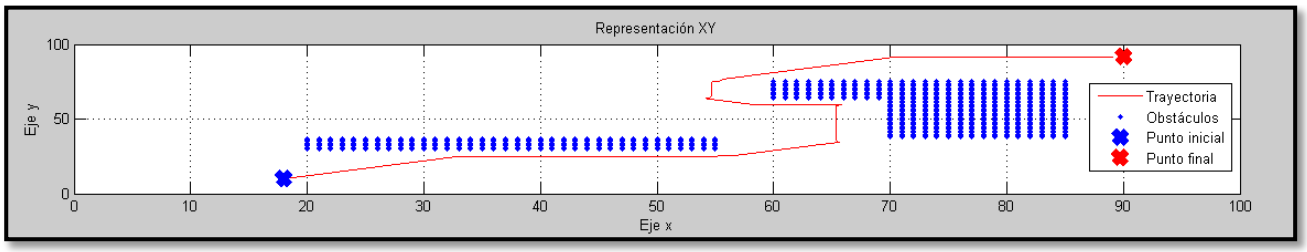

Figura 2-4. Trayectoria generada

2.2. Clasificación de métodos de planificación

Existen diferentes estrategias de planficación de trayectorias [6]. Estas se pueden dividir según la representación del entorno que se haya usado. A continuación, se explican las más relevantes dentro de cada categoría.

Comenzamos dividiéndolos en métodos de:

- *roadmap*
- descomposición en celdas
- potencial
- planificación probabilística.

2.2.1. Métodos de *roadmap*

Son todos aquellos que convierten un modelo multidimensional en una red de curvas unidimensional [8], éstas redes se encuentran en una zona libre de obstáculos, es decir, nos están mostrando todos los caminos posibles que existen. Lo que hace diferente a un método de *roadmap* de otro es cómo generar estas redes. Podemos destacar dos de ellos:

- grafos de visibilidad
- diagrama de Voronoi

2.2.1.1. Grafo de visibilidad

El mapa se crea trazando rectas desde cada vértice hasta todos los vértices visibles, incluidos también los puntos inicial y final. Estas rectas serán las que iremos seleccionando para crear una trayectoria.

Lo que define qué camino debemos tomar es el criterio de trayectoria, que será previamente definido. Estos criterios pueden ser por ejemplo: realizar el trayecto más corto o más seguro o la trayectoria óptima desde el punto de vista perceptivo.

Como podemos observar, el mayor inconveniente de esta estrategia es que el camino se genera demasiado cerca de los obstáculos. Para intentar solucionar este problema, ponemos un radio de seguridad que va a rodear el obstáculo.Este radio de seguridad, que será un parámetro de nuestro algoritmo nos permite adaptar nuestra estrategia para robots de distintos tamaños.

La Figura 2-5 muestra este radio de seguridad de color rojo, el obstáculo real de color amarillo y las rectas verdes nos muestran los caminos que podemos seguir, siendo la combinación de ellos los que crearán la

trayectoria reflejada en color azul.

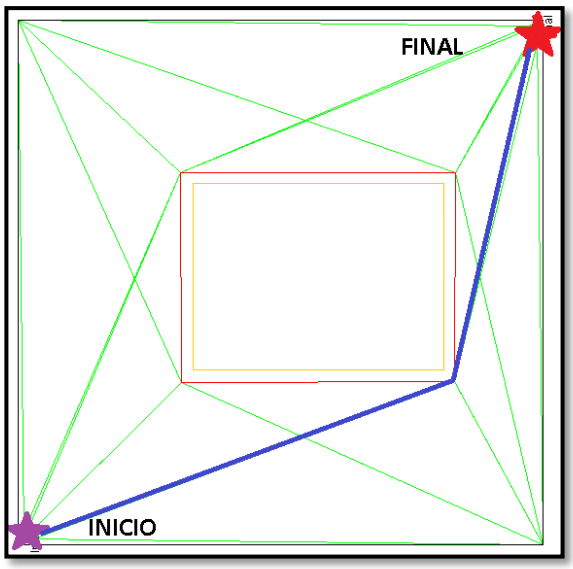

Figura 2-5. Ejemplo de grafo de visibilidad [20]

2.1.1.2. Diagrama de Voronoi

El diagrama de Voronoi se genera calculando líneas equidistantes de los obstáculos más cercanos [9]. La Figura 2-6 muestra un diagrama de Voronoi generado con el comando de Matlab: voronoi. En ella podemos ver los obstáculos señalizados como puntos y un conjunto de líneas que se llaman aristas que son las que definirán nuestra trayectoria.

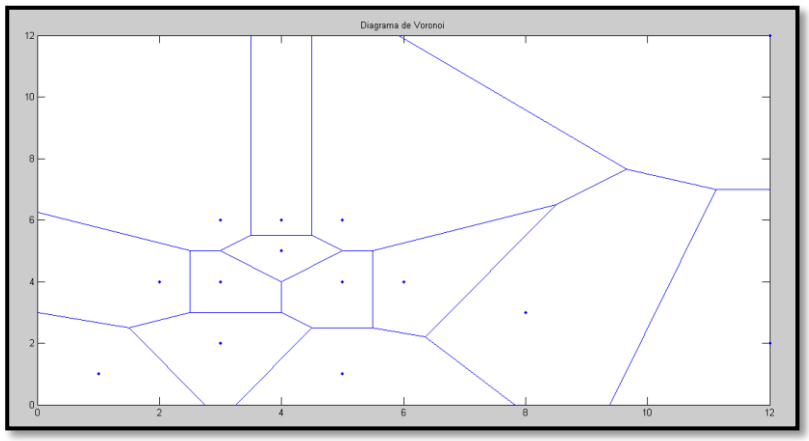

Figura 2-6. Ejemplo de diagrama de Voronoi

Con este método conseguimos que nuestra trayectoria pase siempre lo más alejada posible de los obstáculos, digamos que este método nos proporciona trayectorias muy seguras. Debido a esto es fácil ver lo útil que resulta para la planificación multi-robot.

Una vez hemos generado nuestro diagrama de Voronoi son los algoritmos de planificación los que entran en acción para calcular el camino adecuado para unir el punto inicial con nuestro punto final. Entre los algoritmos de búsqueda se destacan los siguientes: A*, D*, Dijkstra,etc.

2.1.2. Métodos de descomposición en celdas

Los métodos de descomposición en celdas se pueden subdividir en descomposición en celdas exactas o aproximadas [7].

2.1.2.1. Descomposición en celdas exactas

Esta estrategia tiene una serie de ventajas, pues es fácil de implementar, puede modelar cualquier entorno y hace la planificación en este de forma sencilla. Por el contrario, la resolución de la celda resulta un inconveniente, al igual que el uso de recursos.

Cada celda va a estar "rellena" con un atributo. Estos atributos pueden ser, por ejemplo, el grado de ocupación, la probabilidad, alguna información del terreno… Estos métodos tienen que encontrar un grafo que indique qué celdas están libres de obstáculos.

Dentro de sus variantes podemos destacar la descomposición trapezoidal (ver Figura 2-7) que aunque presente mayor complejidad computacional resulta más exacta. En este método tomamos los vértices *v* y dibujamos rectas en el eje *y*. Estas rectas serán tanto para aumentar la *y* como para disminuirla, pararemos una vez lleguemos a los límites de nuestro mapa. Con esto hemos construido nuestras celdas, lo que posteriormente va a convertirse en nuestra trayectoria.

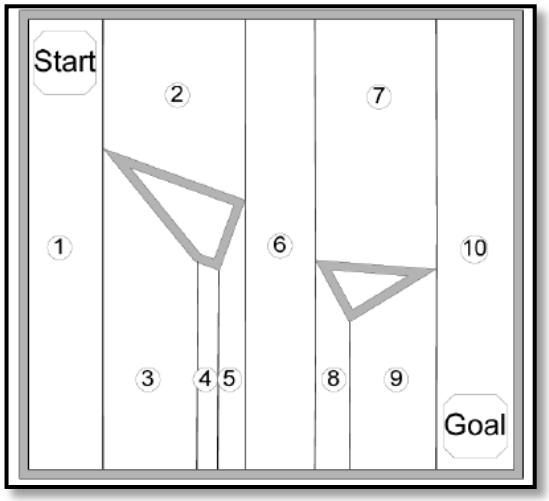

Figura 2-7. Ejemplo de descomposición en celdas exactas [10]

Debe quedar claro que nosotros seleccionamos una celda, no un punto exacto al que ir. Este punto que seleccionaremos será el centro de la celda elegida.Una vez hemos unido los puntos medios, tenemos nuestra trayectoria (ver Figura 2-8).

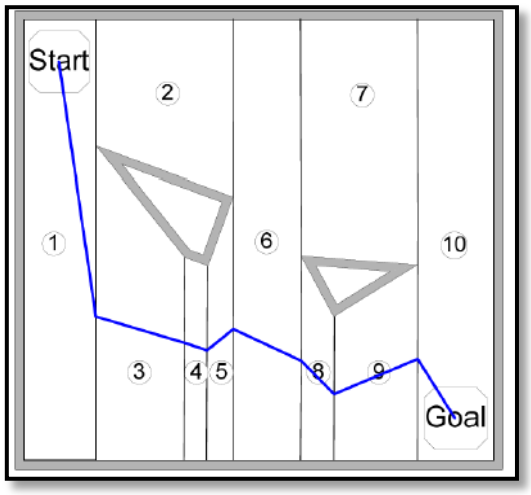

Figura 2-8. Trayectoria definida pasando por el punto medio [10]

2.2.1.2. Descomposición en celdas aproximadas

En este método tendremos celdas que podrán ser etiquetadas como vacías, llenas o mixtas. Una celda vacía será aquella que esté completamente libre de obstáculo, la celda llena la que forma en su totalidad parte del obstáculo y la mixta cualquier celda que no sea ni vacía ni llena.

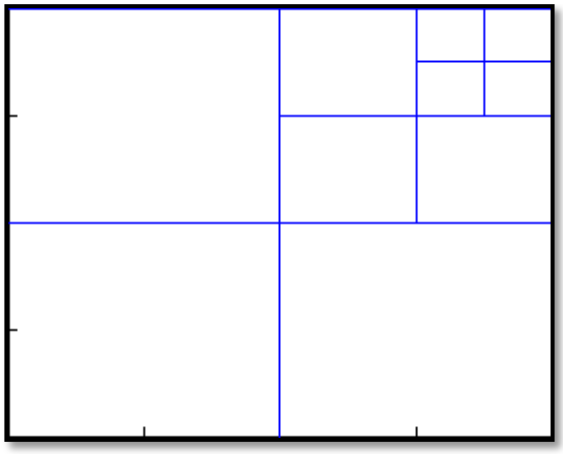

Figura 2-9. Ejemplo de celdas aproximadas

Nuestra meta es conseguir que solo existan celdas llenas y vacías y para esto vamos a dividir las celdas mixtas hasta conseguir que desaparezcan como aparece en la Figura 2-10. Esto hace que mejoren las desventajas de los métodos de celda de tamaño fijo, pero resulta más complicado de implementar y realizar la planificación. Estas complicaciones hacen posible que no encontremos una solución aunque exista.

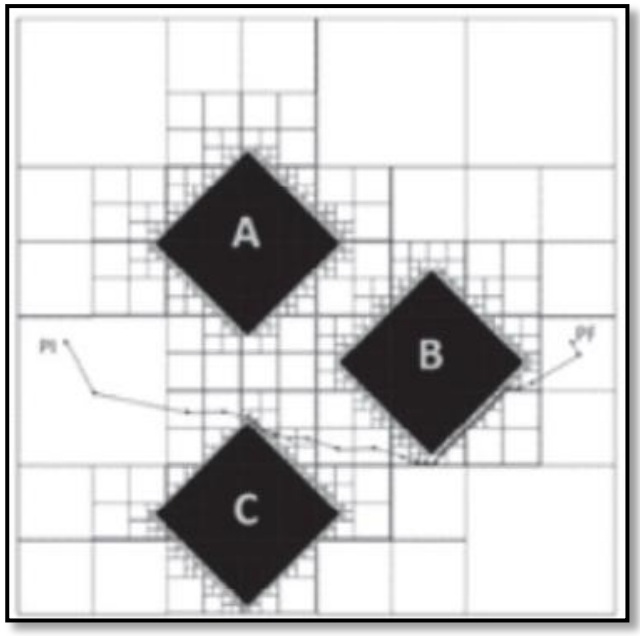

Figura 2-10. Simulación con el método de descomposición de celdas aproximadas [23]

2.2.2. Métodos potenciales

Se basa en la idea de asignar una función de potencial al espacio. Nuestro punto objetivo tendrá el potencial más bajo y esto hace que nuestro vehículo se vea atraído, en cambio, un obstáculo lo repele.

2.2.2.1. Método de los campos potenciales

Esta es la estrategia que vamos a usar en el trabajo. Partimos de la necesidad de discretizar el espacio. Imaginemos que tenemos un escenario de 100 m x 100 m, y hacemos nuestros cálculos con celdas de 1m x 1m. Cada celda tendrá un valor de nuestro campo de fuerzas. Este valor de potencial al que nos referimos será la suma de las fuerzas atractivas y repulsivas a las que se ve sometida nuestra casilla.

Estas pueden ser:

- **Atractivas**: calculamos la distancia entre todos los puntos del mapa y el punto objetivo (*Target*) que contiene la posición en el eje *x, y* y *z* . Se debe multiplicar esta distancia por una constante K que tendrá siempre un valor menor que 1 para moderar estos valores.

$$P(i,j,k) = \sqrt{(Target(1)-i)^2 + (Target(2)-j)^2 + (Target(3)-k)^2} * K$$

Con esto, se ha obtenido la primera parte del campo de fuerzas. Para el ejemplo que vamos a usar, quedaría un campo como el de la Figura 2-11.

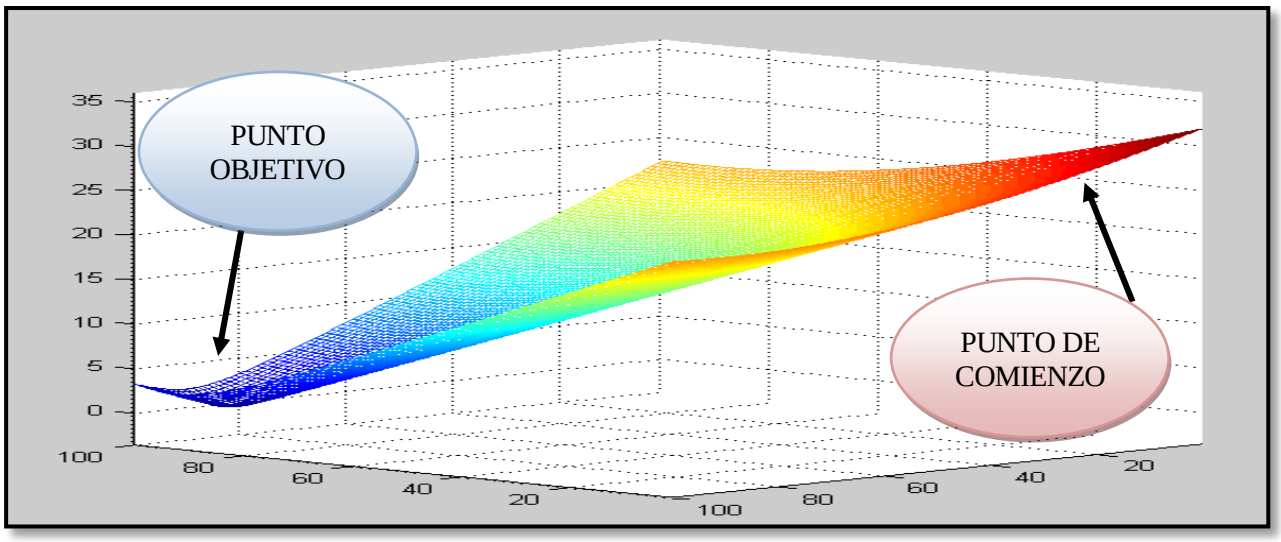

Figura 2-11. Simulación Matlab del campo de fuerzas debida a la distancia

Está claro que a menor valor de fuerza, más cerca del objetivo nos encontramos y más atractivo nos resultará.

- **Repulsivas**: esta fuerza la crean los obstáculos. El valor de mi celda se incrementará un valor proporcional a lo cerca que estemos del obstáculo. Esta influencia dependerá de un parámetro que denominaremos Rsoi o *ratio sensor of influence* que será el radio de influencia de nuestro sensor. Esto significa que una vez hemos detectado obstáculo no solo ponemos una fuerza repulsiva a la casilla donde se encuentra éste, si no que también influirá a su alrededor. La fuerza repulsiva que produce un obstáculo en las celdas contiguas viene dada por la siguiente ecuación:

$$Obs = \frac{Rsoi - Dist}{Dist}$$

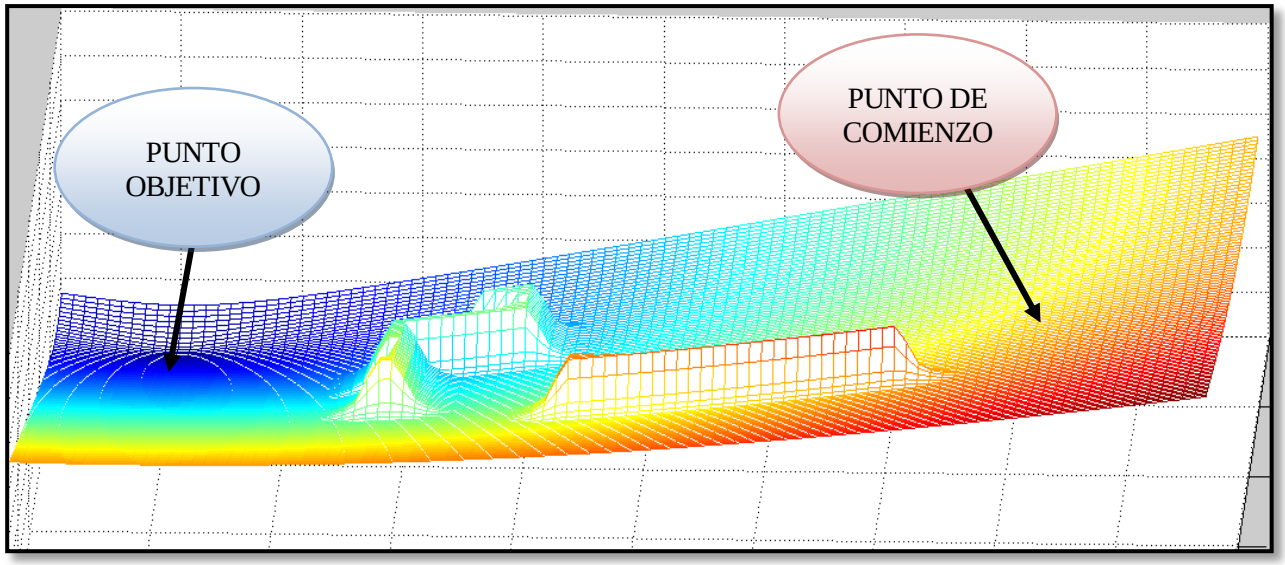

Figura 2-12. Simulación en Matlab del campo de fuerzas de un mapa con obstáculos

La Figura 2-12 representa un campo de fuerzas en Matlab. Aquí es fácil comprobar como el punto objetivo estará en la zona azul oscuro, siendo ésta la zona de menor potencial. La posición inicial estará en la zona roja, que es la de máximo potencial. También es intuitivo relacionar esas "montañas" con los obstáculos del mapa. Se peude observar cómo afectan a nuestro campo de fuerzas, incrementando no solo el valor de potencial en esas celdas si no también las de su alrededor de forma gradual y haciendo así que esa posición sea más desfavorable.

Una vez tenemos las fuerzas atractivas y repulsivas sólo debemos sumar los valores y ya tenemos nuestro campo de fuerzas. A continuación, nuestro algoritmo recorre las posiciones vecinas a nuestra posición actual y finalmente, elige la celda que tenga menor valor, siendo esa la posición más óptima y a la que nos dirigiremos. Una vez estemos en esta posición volveremos a realizar la misma operación hasta que lleguemos al punto objetivo.

¿Cuál es el talón de Aquiles de este método? Los mínimos locales. Un mínimo local se da cuando nos encontramos en una posición y ninguna de las posiciones entre las que buscamos nuestro siguiente *waypoint* es mejor, es decir, que estamos en el menor valor del campo de fuerzas dentro de las posiciones vecinas que contemplamos para realizar el siguiente movimiento. De esta forma se jamás podría salir de este punto y no se alcanzaría la posición deseada.

Se necesita un método para evitar este suceso o para poder salir de él. Se puede atacar el efecto negativo de los mínimos locales de distintas formas, por ejemplo, detectar cuales serán estos mínimos y aplicarles una fuerza repulsiva mayor. También se puede usar un comportamiento con movimientos circulares o el que se ha usado en este trabajo, ya que se considera más efectivo, crear una posición objetivo ficticia.

La solución de crear una posición objetivo ficticia sigue estos pasos:

1. **Comprobar que estamos en un mínimo local**: esto podemos hacerlo mirando el tiempo que llevamos en la misma posición o el número de veces que nos han calculado la posición actual como la mejor posible, si este tiempo o número de veces que hemos obtenido como solución permanecer en la posición actual es mayor que cierto umbral saltaría la alerta de que estamos en un mínimo.

2. **Cambiar el punto objetivo real por uno ficticio**: si cambiamos a un punto objetivo virtual nuestras fuerzas atractivas, es decir, las distancias cambiarán y nuestro robot saldrá del mínimo. El proceso de elegir el nuevo punto objetivo es bastante sencillo, comprobamos dónde está el obstáculo y una vez sabemos su posición el algoritmo nos da dos puntos ficticios posibles para poder salir de la influencia del obstáculo. Elegiremos el que esté más cerca del punto objetivo real.

3. **Reestablecer el punto objetivo real**: debemos volver a cambiar al punto objetivo real pero tenemos que asegurarnos que estamos en una posición a partir de la cual no vamos a volver a entrar en el mínimo local.

2.2.3. Mapas probabilísticos (PRMs)

El uso de este tipo de estrategias está dirigido sobretodo al cálculo de trayectorias de robots con un número elevado de grados de libertad. También resulta muy útil cuando el número de obstáculos móviles y el número de robots aumenta.

Podemos destacar:

- Probabilistic Path Planner (PPP)
- Randomized Path Planner (RPP)
- Exploring Random Tree (RRT).

2.2.3.1. Probabilistic Path Planner (PPP)

Podemos describirlo en términos generales, es decir, sin centrarnos en ningún tipo específico de robot. La base de este método es que mientras se construye el mapa prueba de forma aleatoria las distintas posiciones libres de obstáculos. Finalmente, para realizar la trayectoria los conecta por caminos simples.

Como inconveniente nos encontramos con el tiempo de ejecución.

2.2.3.2. Randomized Path Planner (RPP)

Este método se basa en los campos potenciales, lo que lo diferencia es su forma de salir de los mínimos locales. Usa el movimiento browniano [12], este es el que realizan algunas partículas microscópicas que se hallan en un medio fluido.Por ejemplo, es el que realizan las partículas de polen en el agua, que cambian su posición con movimientos aleatorios sin razón aparente.

Al igual que PPP, el tiempo de ejecución es su mayor inconveniente.

2.2.3.3. Exploring Random Tree (RRT)

La exploración aleatoria de árboles [10] hace una exploración del espacio y determina si hay o no obstáculo en ese camino aleatorio. Por tanto, va generando camino siempre que no tenga obstáculo y finalmente selecciona el mejor de los caminos factibles. Los resultados son buenos aunque el tiempo vuelve a ser el problema.

En las figura 2-13. podemos ver una situación inicial con un punto de partida (Ps) y una meta (Pe).También observamos unos círculos, que simbolizan los obstáculos de ese mapa.

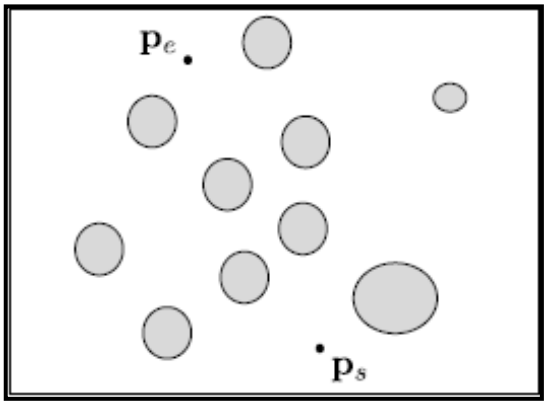

Figura 2-13.Situación inicial [10]

En la figura 2-14. ya es visible el método realizado. Esas ramas que observamos son los caminos posibles que se han ido creando de forma aleatoria, ahora podemos comprender de dónde viene el nombre de este método, pues estos caminos se asemejan a ramas de arboles. Finalmente la línea subrayada sería la trayectoria.

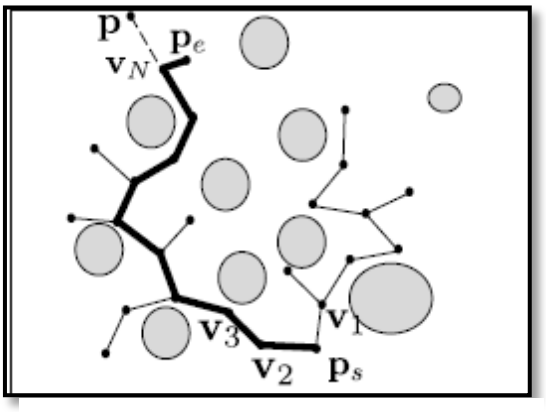

Figura 2-14. Resultado método RRT [10]

3. ROS

Todo arde si le aplicas la chispa adecuada.

- Héroes del Silencio-

El framework que se ha utilizado en este trabajo es ROS[3][4]. Actualmente sólo se ejecuta en las plataformas basadas en Unix. En particular se ha usado Ubuntu. ROS es un sistema operativo *open source* en el que se facilita una serie de librerías y herramientas muy útiles para los desarrolladores de software que estén centrados en el ámbito de la robótica como es el caso.

Como sistema operativo que es, tiene asociado los servicios de abstracción de hardware, control de dispositivos a bajo nivel, gestión de paquetes,etc. Es conocido por los visualizadores tan potentes que utiliza. Estos nos facilitan la realización de simulaciones con una alta fiabilidad y realismo.

3.1. Introducción a ROS

ROS fue creado para dar soporte a proyectos de software dirigidos a la robótica. Lo que lo hace valioso es la gran cantidad de códigos a disposición del usuario para su reutilización y la facilidad que tiene éste de poder compartirlo para su distribución. ROS no es una infraestructura de tiempo real, aunque es posible intregrarlo con el código en tiempo real.

Otro aspecto positivo de este sistema es que se nos permite diseñar ejecutables de forma individual y acoplarlos cuando vayamos a lanzarlos. ROS se ejecuta como una red de procesos *peer-to-peer* o red entre pares. Este tipo de red de computadoras se caracteriza porque todas las que están conectadas se comportan como iguales. Todas funcionan en todos o algunos de los aspectos sin ser clientes ni servidores fijos.

Respecto a la comunicación de ROS, se permite una transmisión síncrona de servicios. También tenemos transmisión síncrona de datos a través de *topics* que nos permite obtenerlos de diferentes nodos y por último su almacenamiento a través de parámetros.

3.2. Conceptos básicos

ROS tiene tres niveles de conceptos [5]:

1. Nivel de sistema de archivos.
2. Nivel de computación gráfica.
3. Nivel de la comunidad.

A continuación detallaremos varios de los conceptos pertenecientes a estos niveles:

- **Paquetes:** es la unidad principal de organización de software de ROS. Este puede contener procesos de ROS en tiempo de ejecución, bibliotecas, datos y archivos de configuración. Son el elemento de construcción más atómica.

- **Repositorios:** es el paquete o conjunto de paquetes que comparten una misma versión y pueden ser liberados juntos.

- **Nodos:** son procesos en los que se realizan cálculos.Utilizan una biblioteca cliente para comunicarse con otros nodos. Los nodos pueden publicar o suscribirse a un *topic*, pueden utilizar o proporcionar algún servicio. Por lo general, un sistema tiene activos muchos nodos, cada uno de ellos con una función.

- **Tópicos o *Topics***: como ya se ha dicho la transferencia de datos se realiza a través del paso de mensajes. Los *topics* serían un medio de comunicación entre nodos. Un *topic* puede ser o suscriptor o publicador. Sus nombres dejan aclaran la función de cada uno, los publicadores pasan la información a estos *topics* y si un nodo necesita dicha información se suscribirá.

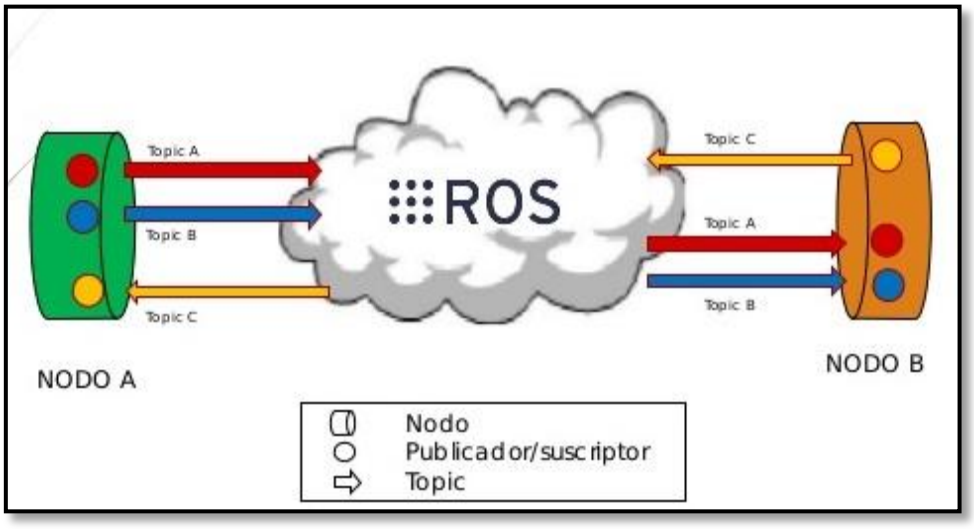

Figura 3-1. Esquema de funcionamiento nodos-*topics*. [13]

- **Maestro o *Master*:** el ROS *Master* es vital para el funcionamiento, sin él los nodos no serían capaces de encontrarse unos a otros, no podríamos invocar servicios.

- **Parámetro Servidor:** el servidor de parámetros permite que los datos sean almacenados en una ubicación central. Actualmente forma parte del *Master*.

- **Mensajes:** si se habla de la comunicación entre nodos, ésta se realiza por el paso de mensajes. Un mensaje es una estructura de datos. Hay una serie de mensajes definidos en ROS y también existe la opción de crear tu propio tipo de mensajes. Estos pueden estar compuestos por enteros, flotantes, vectores, etc.

- **Servicios**: aunque los *topics* resultan muy útiles, para ciertas tareas es necesario un modelo distinto de publicación y suscripción. Se necesita por tanto los servicios, dónde utilizamos las peticiones y respuestas. Se tiene un servidor y un cliente, este último solicita una respuesta a partir de unos datos de entrada.

- **Bolsas o *bags***: es un formato para guardar y poder volver a reproducir datos de mensajes de ROS. Los *bags* resultan muy útiles para la toma de datos de sensores. Han sido vitales para las simulaciones y comprobación del algoritmo.

3.3.Roslaunch

Esta herramienta de ROS facilita el lanzamiento de múltiples nodos simultáneamente, no sólo es útil por eso, si no que también se puede incluir en él una lista de parámetros. La ventaja de tener los parámetros en el archivo .launch y no en el nodo es que si queremos realizar cambios de estos parámetros no se tiene por qué compilar de nuevo todo el código.

A continuación se pone un ejemplo de cómo se incluirían los nodos deseados en nuestro fichero .launch. Basta con escribir <*node/*>, poner el nombre del nodo y el paquete en el que se encuentra.

```
<node name="maptoshadow2" pkg="reaccionando" type="maptoshadow2" output="screen"/>
<node name="gpsinicial2" pkg="reaccionando" type="gpsinicial2" output="screen"/>
<node name="shadowtofcu2" pkg="reaccionando" type="shadowtofcu2" output="screen"/>
```

Figura 3-2. Ejemplo de programación de nodos en el archivo .launch

Igual de sencillo para los parámetros. Se pondría <*param/*>, su nombre, su valor y su tipo.

```
<param name="DSegPlus" value="3" type="string" />
<param name="DSegMin" value="2" type="string"/>
<param name="Rsoi" value="20" type="string" />
```

Figura 3-3. Ejemplo de programación de parámetros en el archivo .launch

Para lanzar nuestro archivo escribimos en nuestra terminal el comando roslaunch seguido del nombre del paquete y del nombre del fichero .launch que queremos lanzar:

```
carmen@carmen-HP-ProBook-4520s:~/catkin_ws$ roslaunch reaccionando planificador_para_bags.launch
```

Figura 3-4. Comando roslaunch

3.4.RViz

Una de las ventajas de ROS es su potente visualizador. RViz es un simulador en 3D, en él se puede representar información de sensores y de los cálculos del algoritmo. No hace falta destacar la importancia que esto supone. Este visualizador facilita la realización del algoritmo, así como la detección de errores.

Además de ser útil, es relativamente sencillo de usar. RViz funciona usando *topics*. Los datos que queremos representar se envían a un *topic* concreto y le asignamos un display que RViz publicará. Un display es una pantalla o indicador numérico utilizado para visualizar una determinada información. Para usar Rviz tan solo se debe escribir en nuestra terminal el comando rviz y si todo es correcto ella nos devolverá lo que podemos ver en la imagen siguiente, donde nos muestra información sobre la versión de RViz que estamos usando.

La interfaz de RViz es muy intuitiva, existen varias zonas que podemos diferenciar:

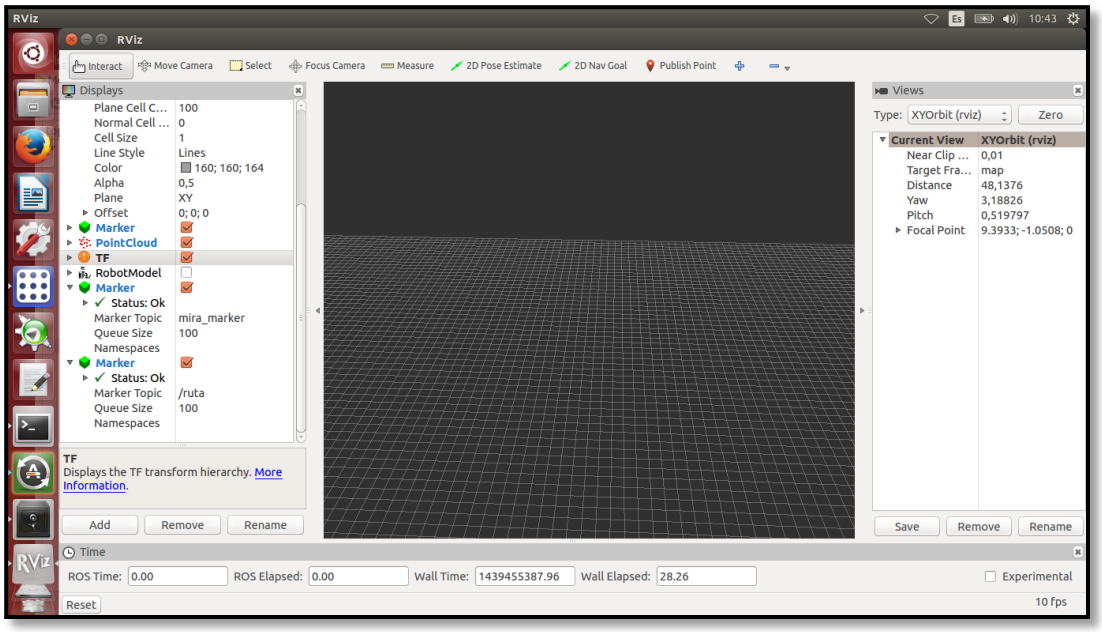

Figura 3-5. Interfaz de RViz

1. **Barra de herramientas de la cámara**: son las distintas opciones para posicionar la cámara.

Figura 3-6. Barra de herramientas de la cámara.

2. **Sección de los displays**: aquí añadimos el display necesario y se le vincula el *topic* que contiene los datos.

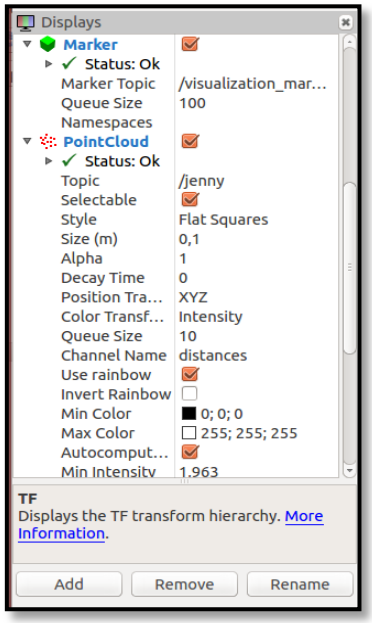

Figura 3-7. Sección de los displays

Por ejemplo, el algoritmo implementado obtiene información de un sensor en forma de nube de puntos en el *topic* /jenny, por tanto, seleccionaríamos el display *PointCloud* y le asociaríamos dicho *topic*. Así estaríamos representando nuestra nube de puntos. Además se puede seleccionar distintas preferencias, el tamaño del punto o incluso que los puntos de la nube sean de diferentes colores dependiendo de la distancia a la que estén.

Esto último facilita mucho la lectura de lo que está sucediendo, en la imagen adjunta podemos ver dos pies de canasta y el suelo. Las canastas se encuentran a mayor distancia, teniendo tonos más frios.

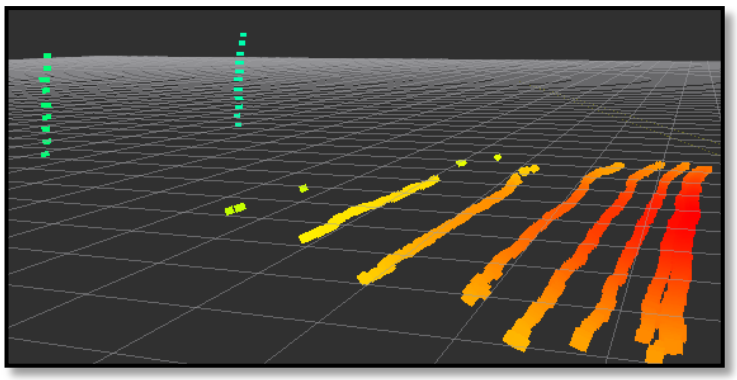

Figura 3-8. Representación nube de puntos. Los puntos a mayor distancia con tonos fríos y los más cercanos con tonos cálidos.

Hay una gran variedad de displays, tenemos también el tipo *Marker*, que puede tener 11 formas distintas. Las que se han utilizado son el cubo, para representar los *waypoints*.

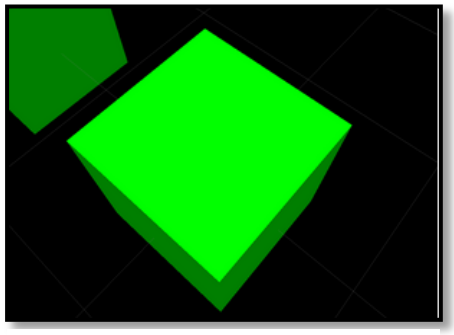

Figura 3-9. Marker tipo cubo [3]

También usamos el tipo lista de cubos, es una recta que une el robot con el *waypoint* intermedio si lo hubiese y con el *waypoint* objetivo. Con esto se obtiene una visión de la trayectoria bastante clara.

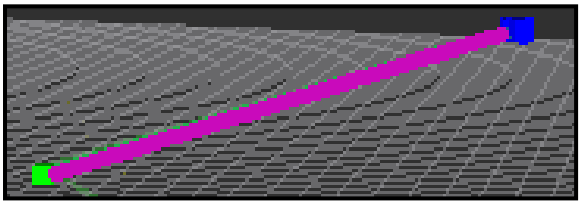

Figura 3-10. Marker tipo lista de cubos

El último *marker* utilizado es la esfera. En el algoritmo se utiliza para indicar el radio dentro del que se considera que el robot ha llegado a la posición objetivo.

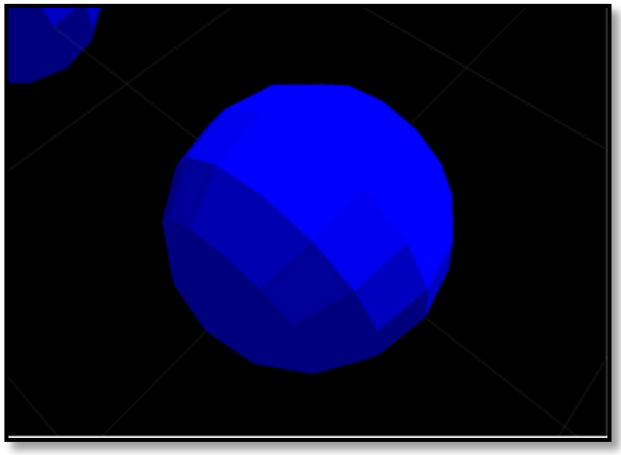

Figura 3-11. Marker tipo esfera

Se ha usado el display tipo *transfer function* (TF) este representa los ejes de coordenadas, dándonos la orientación del mapa: eje *x* (color rojo),eje *y* (color verde) y eje *z* (color azul). En la Figura 3-13 se observan dos TF, uno que representa la posición de la base del láser Hokuyo UTM 30-LX y otro con una traslación en el eje Z que se situa en la base del Servomotor Dynamixel MX-28.

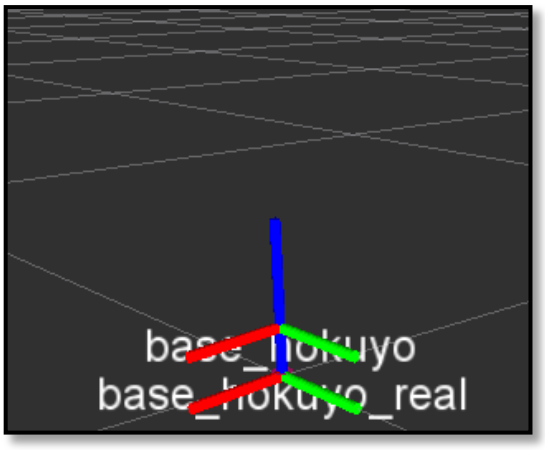

Figura 3-12. Display TF

3. **Pantalla de simulación**: se observa la representación completa con todos los displays seleccionados.

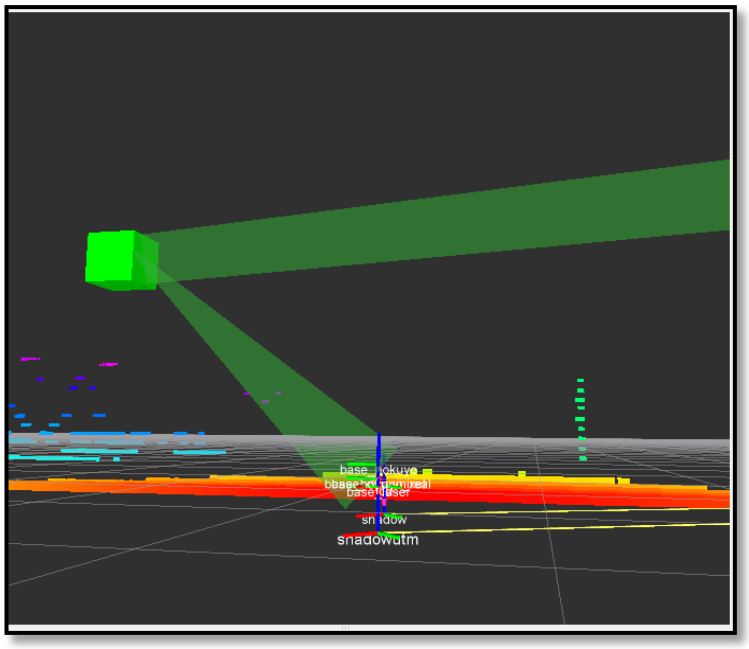

Figura 3-13. Representación RViz de un escenario real. Se aprecia la nube de puntos, el *waypoint* intermedio calculado (cubo verde), la posición del robot (marcador TF) y la trayectoria (lista de cubos verdes).

.

4. DESCRIPCIÓN DEL MÉTODO IMPLEMENTADO

Navegaré por bucles y algoritmos,
recuperando la mejor versión de mí mismo.
- Antílopez -

El objetivo en este trabajo es que un UAV realice un plan de vuelo definido y enviado desde la estación de tierra. Este plan de vuelo se compone de una secuencia de *waypoints* posiciones dadas en coordenadas GPS (Global Positioning System).

El UAV debe llegar a estas posiciones por un camino eficiente y de una forma segura, es decir, libre de colisiones. Todo esto, sin tener conocimiento previo del espacio en el que se va a mover. Nuestra única información del entorno llegará de mano del láser Hokuyo UTM 30.

Una vez obtenido el mapa se verifica si se puede ir a la posición deseada directamente o es necesaria una maniobra, es decir, si debe ejecutarse el método de los campos potenciales para alcanzar dicha posición de forma segura. Esta verificación se hace a partir de la posición del UAV dada por el GPS. Así hasta completar el plan de vuelo.

4.1. Global positioning system (GPS)

El sistema de posicionamiento global [16] o GPS nos permite determinar la posición de un objeto en todo el mundo. Funciona gracias a 24 satélites en órbita a 20200 km de altura. Con estos 24 satélites somos capaces de cubrir toda la Tierra (ver Figura 4-1).

Para obtener la posición se usan mínimo cuatro satélites, estos nos envían un identificador del satélite y la hora del reloj en la que se realiza el envío. Gracias a esto podemos saber el tiempo que ha tardado en llegarnos desde cada posición y obtiene la nuestra mediante el método de trilateración inversa. Conocidas las distancias es relativamente fácil obtener la propia posición. Su precisión puede llegar a los centímetros.

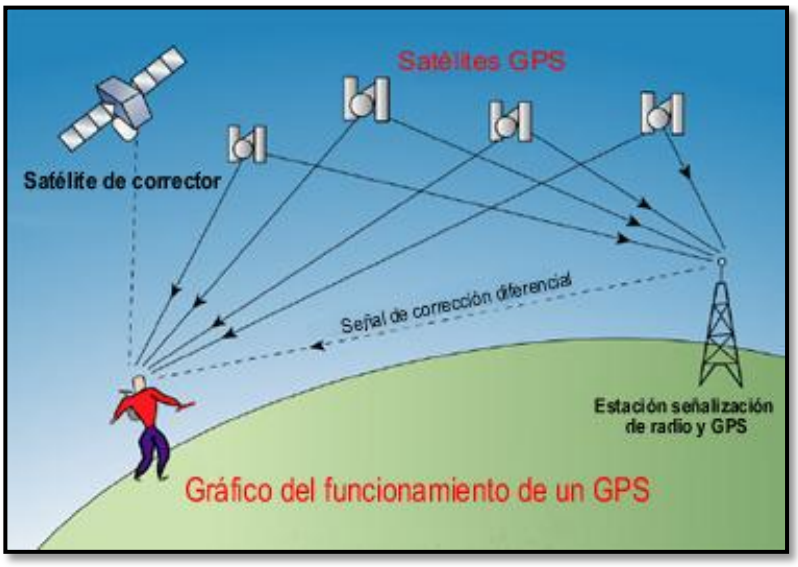

Figura 4-1. Gráfico del funcionamiento de un GPS [21]

La posición GPS se da en tres coordenadas: altitud, latitud y longitud. Deberemos hacer una transformación de las posiciones para situarlas en nuestro mapa y poder trabajar con estos datos.

4.2. Plan de vuelo

Un plan de vuelo es una secuencia de posiciones por las que el UAV debe pasar. En nuestro caso las posiciones son dadas en coordenadas GPS. Para enviar este plan de vuelo tenemos dos opciones. En las simulaciones hemos usado una de las herramientas que nos proporciona ROS, el comando *rqt*. Este nos permite publicar en los *topics* de forma manual mensajes, pudiendo incluir la información que queramos.

Al ejecutar *rqt* aparece una ventana auxiliar donde se tiene que seleccionar *Plugins -> Topics-> Message Publisher*. En nuestro caso vamos a publicar en el *topic: /mavros/misión/waypoints*. Debemos ponerle una frecuencia de 0 Hz, esto es necesario para que realice el envío del *flightplan* (plan de vuelo) tan solo una vez. Si pusiéramos una frecuencia distinta a 0 estaríamos constantemente mandando el plan de vuelo y esto ocasiona errores.

En la Figura 4-2 vemos la pantalla de *rqt* podemos poner el número de componentes que va a tener nuestro vector de *waypoints* siendo este siempre uno más del número de las posiciones deseadas, esto se debe a que el primer *waypoint* se pierde siempre. Finalmente, solo necesitamos completar las variables *x_lat, y_long* y *z_alt* con la latitud, longitud y altura de la posición deseada.

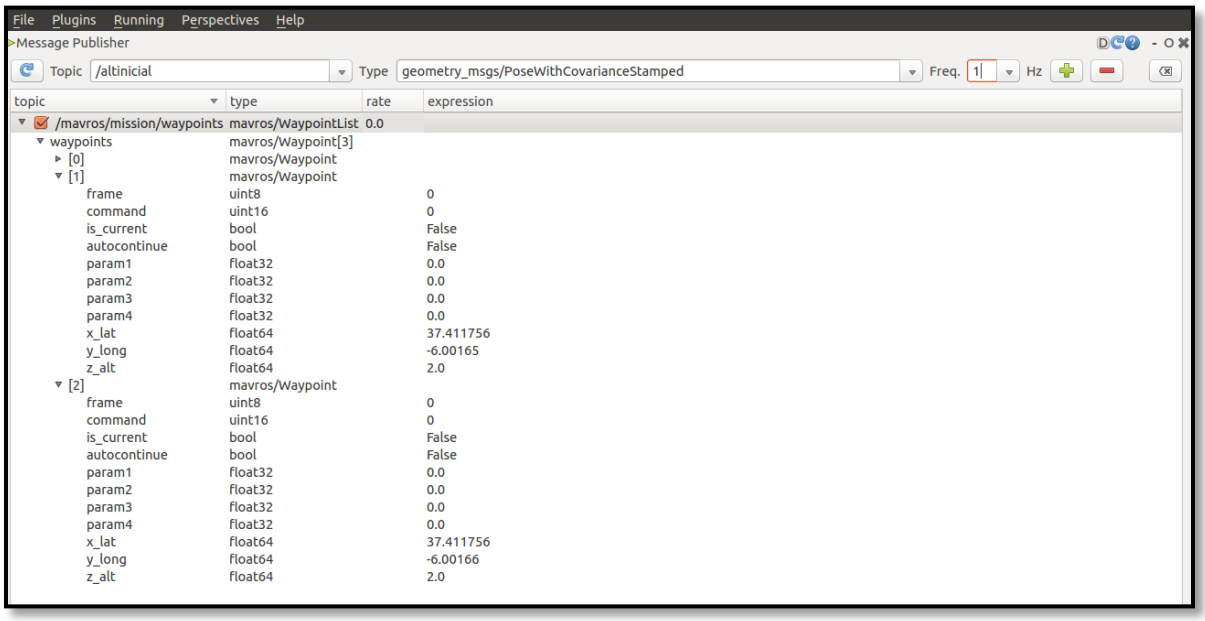

Figura 4-2. Interfaz de herramienta rqt

Una vez hagamos los experimentos en un entorno real usaremos el programa *Mission Planner* , se puede ver su interfaz en la Figura 4-3 [17]. Este nos permite introducir *waypoints* mediante *point-and-click*, es decir, usando Google Maps pinchamos en el punto al que queremos que se dirija nuestro UAV y automáticamente éste se convertirá en un *waypoint*.

Figura 4-3. Captura del programa Mission Planner[17]

4.3.Nube de puntos o *PointCloud*

Como se ha comentado en secciones anteriores, el UAV debe realizar el plan de vuelo siguiendo una trayectoria libre de obstáculos. El mapa de obstáculos se crea a partir de la nube de puntos que el láser Hokuyo UTM 30-LX (ver Figura 4-4) nos proporciona del entorno [19].

Figura 4-4. Láser Hokuyo UTM 30 LX, [11]

Esta nube se realiza con el lanzamiento de un haz de láser con una longitud de onda de 905 nm, escaneando un área semicircular de 270º. La creación de la nube de puntos es más tediosa de lo que parece, pues el láser Hokuyo UTM 30 LX es un láser 2D. Para obtener la nube de puntos 3D se crea un movimiento periódico del servomotor Dynamixel MX-28. Los haces del laser en 2D se van guardando en un búfer y posteriormente se buscarán todos los escáneres realizados desde el principio del barrido hasta el final y se unirán para crear la nube 3D.

La nube de puntos la vamos a recibir cada segundo, este tiempo es un parámetro que podemos modificar dependiendo de la aplicación o el ángulo de barrido que deseemos obtenerUna vez la nube de puntos llega, se trata.

Para ello debemos saber qué contiene el tipo de mensaje de la nube de puntos. Éste lo podemos definir en nuestro algoritmo como *sensor_msgs/PointCloud*. El mensaje está compuesto por tres tipos de mensajes distintos y estos a su vez compuestos por otros:

std_msgs/Header header

geometry_msgs/Point32[] points

sensor_msgs/ChannelFloat32[] channels

Nosotros nos vamos a centrar en *geometry_msgs/Point32[] points*. Éste es un vector de dimensión variable cuyos elementos contienen la distancia que existe desde el láser al obstáculo en el eje *x*, *y* y *z*. La dimensión dependerá de los puntos que detecte. Algo que debemos aclarar es que la nube de puntos siempre llegará, si por ejemplo no detectaramos ningún obstáculo seguiríamos recibiendo la nube y en ella el mensaje *geometry_msgs/Points32.msg points* tendría dimensión cero.

En la Figura 4-5 se ve una fotografía del entorno real y en la Figura 4-6 se representa la nube de puntos asociada.

Figura 4-5. Escenario de prueba para nube de puntos. Cancha de baloncesto de la ETSI.

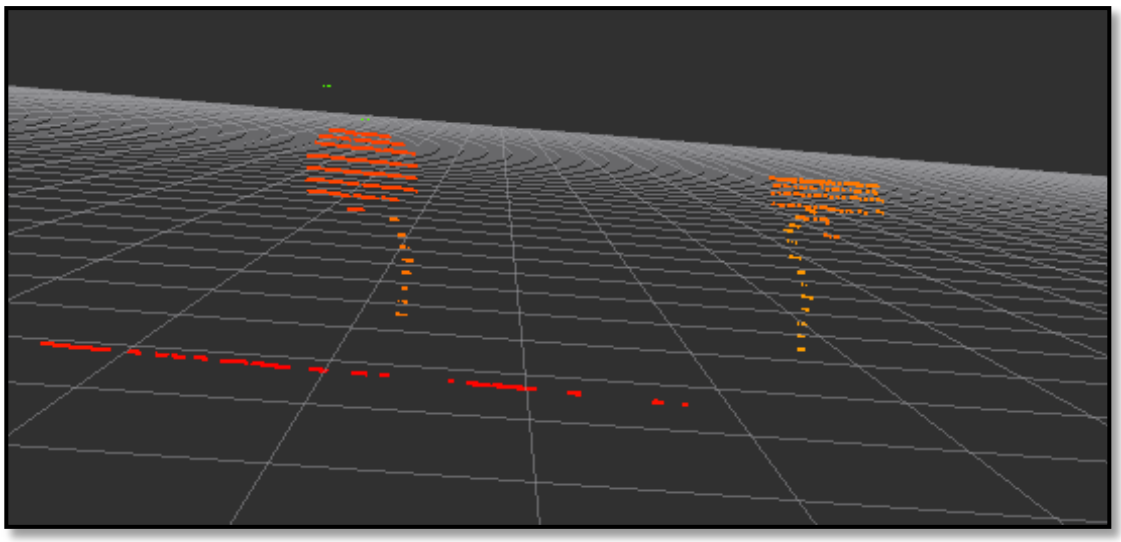

Figura 4-6. Ejemplo de nube de puntos del escenario de prueba.

Se aprecian las dos canastas y parte del suelo

4.4. Funciones de transformación

La Figura 4-9 muestra el árbol de transformadas que tiene nuestro algoritmo. Este árbol se obtiene

ejecutando el siguiente comando de ROS:

rosrun tf view_frames

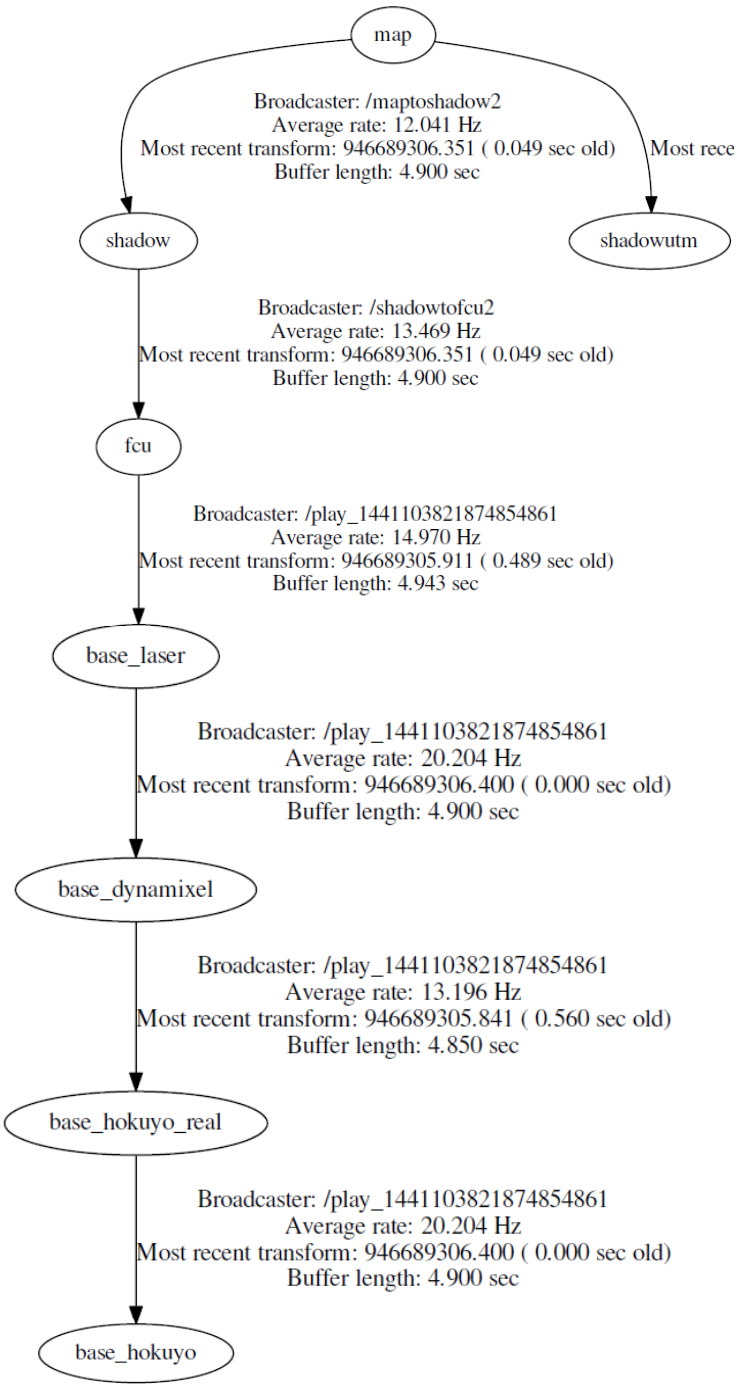

Figura 4-7. Árbol de transformadas de nuestro algoritmo

- **map:** este sería el *frame* que nos posiciona nuestro vehículo en el espacio de trabajo.

- **shadow:** es la proyección en map de los ejes *x* e *y* de nuestro vehículo.

- **shadowutm:** es la proyección en los ejes *x* e *y* de nuestro vehículo en coordenadas UTM. Esta versión existe porque al inicio no se sabía si nos beneficiaba más el uso de estas coordenadas o de las coordenadas GPS. Finalmente, debido a que el *waypoint* que debíamos mandar al autopiloto tenía que ser en coordenadas GPS este *frame* quedó en desuso.

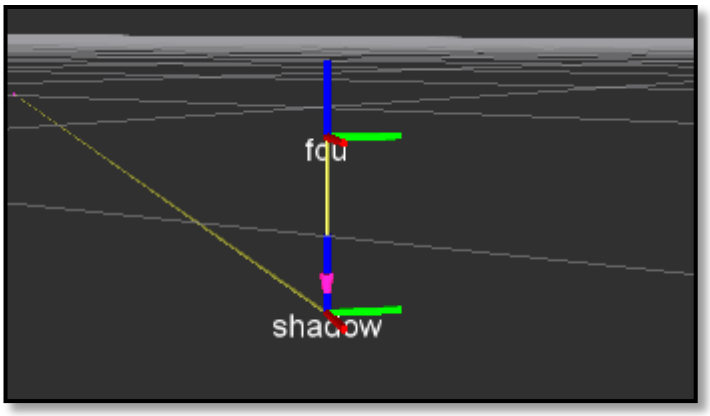

Figura 4-8. Relación entre los frames shadow-fcu.

Ambos ejes son paralelos, presentan una traslación en el eje *z*

- **fcu:** que significa *flight control unit* o autopiloto sería el primer *frame* que encontraríamos situado en nuestro vehículo, en la base donde está este elemento.

- **base_laser:** éste se situa en la base dónde se encuentra el montaje del dynamixel y el láser.

- **base_dynamixel:** éste está localizado en la base del Servomotor Dynamixel MX-28.

- **base_hokuyo_real:** éste coincide con *base_dynamixel.*

- **base_hokuyo:** el último *frame* es el que nos encontramos en la base del láser Hokuyo UTM 30-LX

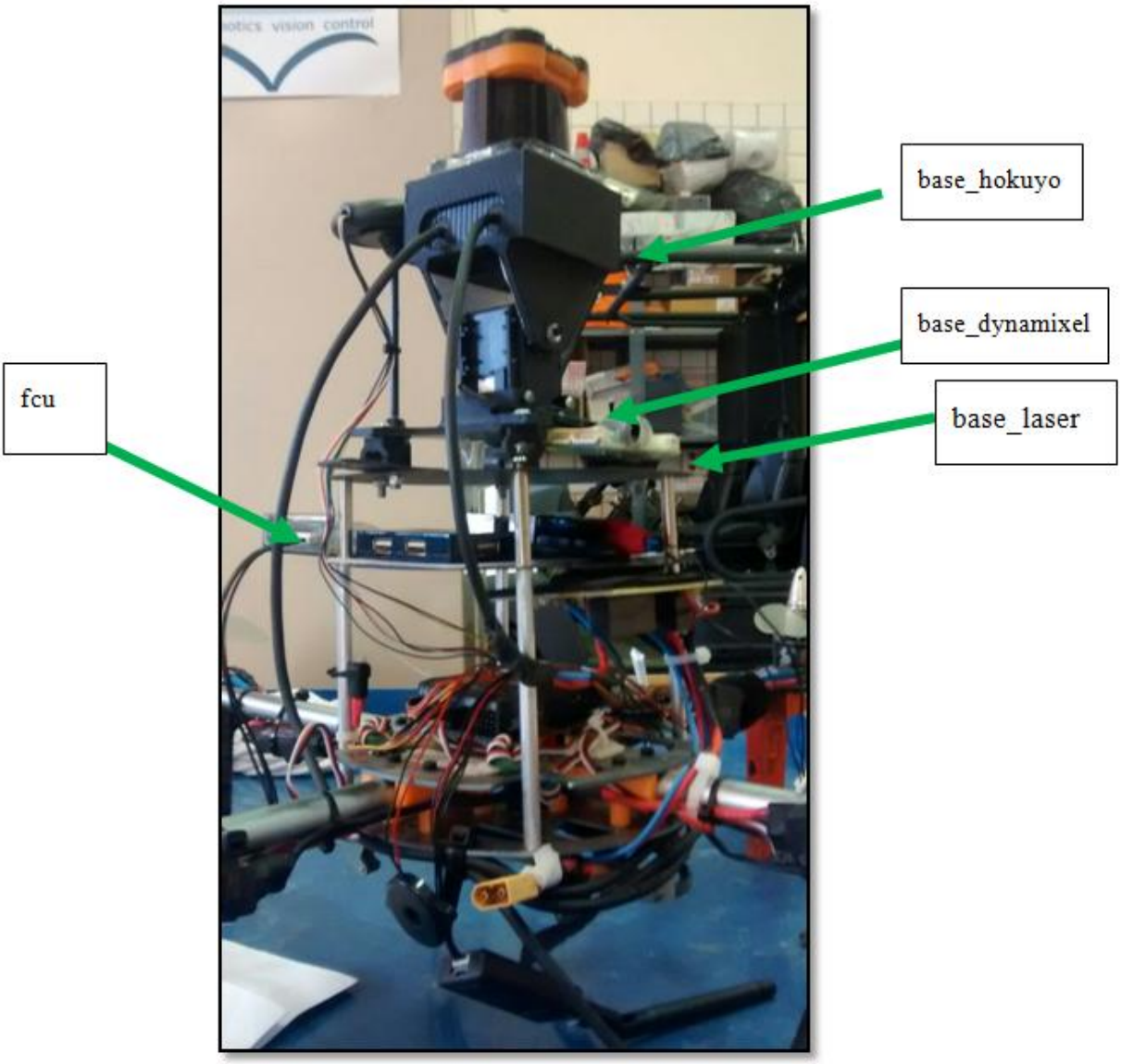

Figura 4-9. Frames de nuestro vehiculo

Como se puede observar en la Figura4-9 la diferencia entre *base_hokuyo, base_hokuyo_real, base_dynamixel, base_laser* y *fcu* es de pequeñas traslaciones en el eje *z*. Debido a esto, no se van a realizar estas transformadas pues supone un pequeño error que podemos tolerar. Por tanto, nos quedamos con la siguiente secuencia de transformadas.

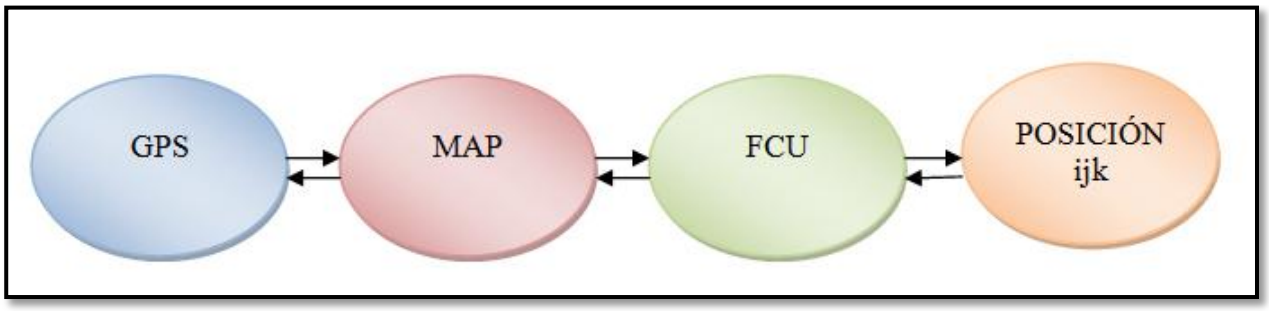

Figura 4-10. Secuencia de transformaciones de nuestro algoritmo

Para realizar la transformada ***GPS-map*** se debe tomar la posición inicial de latitud, longitud y altitud y vincularla a una posición de nuestro mapa, a partir de entonces todas las coordenadas GPS que nos lleguen van a estar relacionadas a las iniciales y, por tanto, nos van a dar un movimiento relativo en el que podamos obtener la distancia a la que nos hemos desplazado.

Respecto a la transformación ***fcu-ijk*** ésta es la que hacemos para posicionar los datos de todas las posiciones del espacio que hayamos tratado en nuestra matriz de cálculo.

4.5.Comprobación camino directo

Antes de calcular un *waypoint* intermedio se debe comprobar si es necesario dicho cálculo o por el contrario podemos ir directamente al *waypoint* objetivo. Para esto hemos usado el algoritmo conocido como *A fast voxel traversal algorithm for ray tracing* [15] de la Universidad de Toronto.

El trazado de rayos es un algoritmo de gran utilidad debido a su simplicidad y elegancia. Éste es un rápido y sencillo método que va a ir atravesando *voxels* (celda o casilla 3D). Su mayor inconveniente es el coste computacional. Este método calcula los rayos de forma muy sencilla, para ir de un *voxel* a otro sólo se requieren hacer dos comparaciones de punto flotante y la suma de otro.

4.6. Método de los campos potenciales

Aunque ya se ha presentado el método de los campos potenciales, esta sección va a profundizar en él comentando varios aspectos del código implementado:

4.6.1. Distancia de los *waypoints* intermedios

Los *waypoints* intermedios son los que el método genera para evitar las colisiones detectadas. Estos *waypoints* deben ser generados a una distancia de la posición actual. Esta es determinada a partir de los errores en la posición GPS recibida y en el seguimiento de *waypoints* del autopiloto.

El método utilizado tiene una matriz P que contiene el campo de fuerzas. Para obtener los *waypoints* se debe tomar una matriz auxiliar que denominaremos NextMove y tiene unas dimensiones que dependen de NM (distancia en metros a la que se generará el *waypoint*). En nuestro caso hemos puesto que los *waypoints* estén a una distancia mínima de 10 metros, este parámetro se puede cambiar fácilmente en el algoritmo.

Por tanto, la matriz *NextMove* tiene una dimensión de NM+1 x 2*NM+1 x NM+1. En nuestro caso particular, será de 11x21x11. El vehículo estará situado en el centro de los ejes *y* y *z* y al inicio del eje *x*.

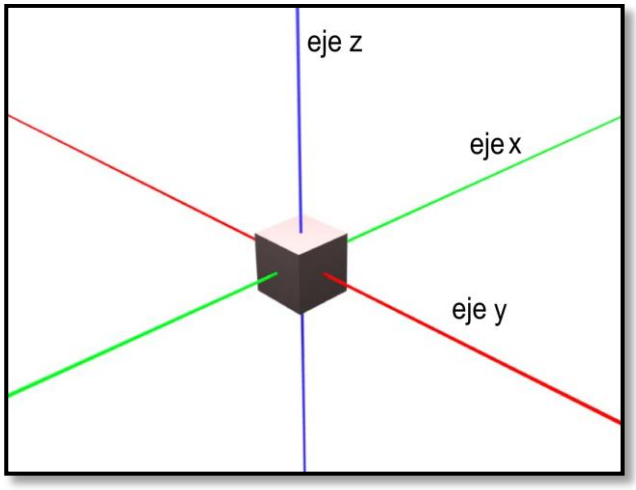

Figura 4-11. Ejes coordenados [22]

Una vez generada la matriz se deben seleccionar aquellas casillas aptas en las que se puede buscar un *waypoint*. Éstas deben estar en los bordes de la matriz, en las imágenes siguientes estas casillas se simbolizan por una recta verde que pasa por ellas.

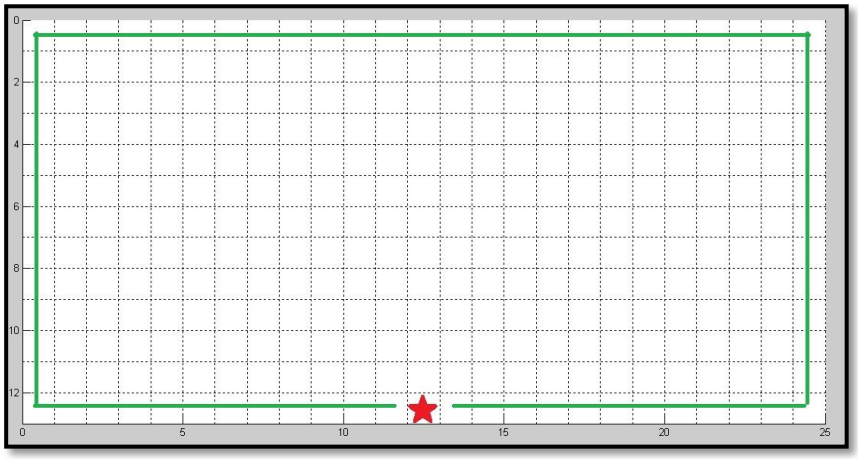

Figura 4-12. Proyección XY de la matriz *NextMove*. Las casillas en las que realizaremos las búsquedas están marcadas con verde

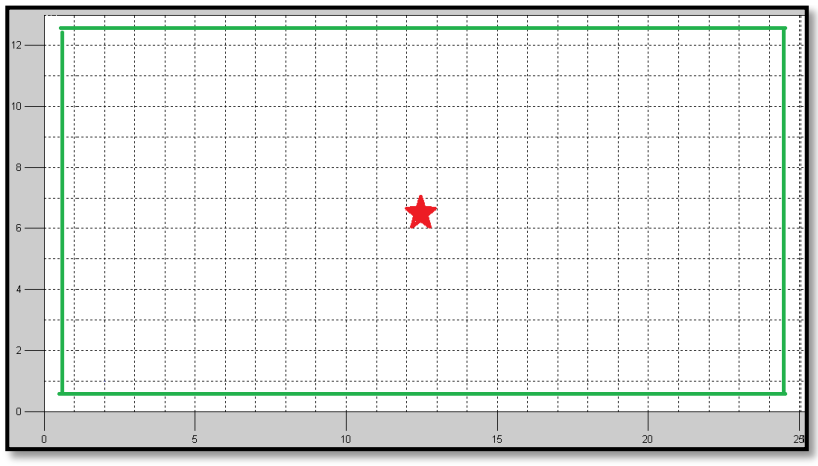

Figura 4-13. Proyección YZ de la matriz *NextMove*.Las casillas en las que realizaremos las búsquedas están marcadas con verde

Todas las casillas que hay entre la posición del vehiculo, señalada con una estrella roja, y esta línea verde, se descartan como *waypoint* (ver Figura 4-13). Para seleccionar una posición como posible *waypoint* antes se debe comprobar si entre la posición del vehículo y ésta existe obstáculo, de ser así ésta no podrá estar entre las posibles elecciones. La Figura 4-14 muestra el caso del que estamos hablando.

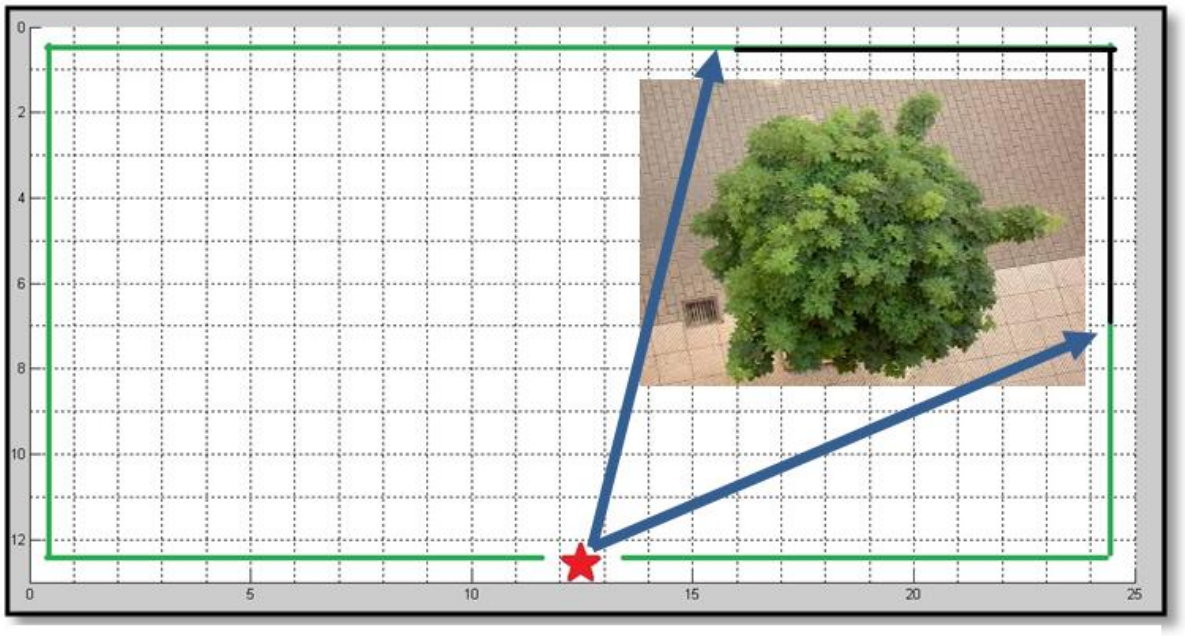

Figura 4-14. Ejemplo de obstáculo dentro de nuestra matriz *NextMove.* Las casillas en las que realizaremos las búsquedas están marcadas con verde, aquellas que antes eran posibles *waypoints* intermedios y ya no de negro.

4.6.2. Restricciones cinemáticas

A pesar de que nuestro vehículo tiene mucha libertad a la hora de realizar los movimientos y no es como un ala fija, hemos pensado en introducir una restricción cinemática para limitar los grados que puede girar y así poder controlar que no se ocasionen giros muy bruscos. Estas restricciones afectan al *yaw*. En nuestro caso hemos puesto ± 60º. Este es un parámetro que se puede modificar en nuestro código dependiendo del vehículo que se use y las necesidades que presente.

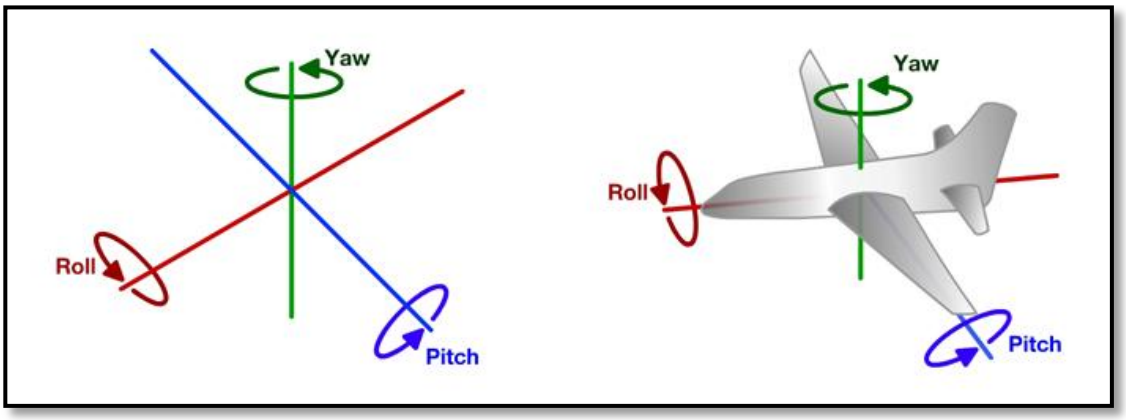

Figura 4-15. Ángulos de navegación [24]

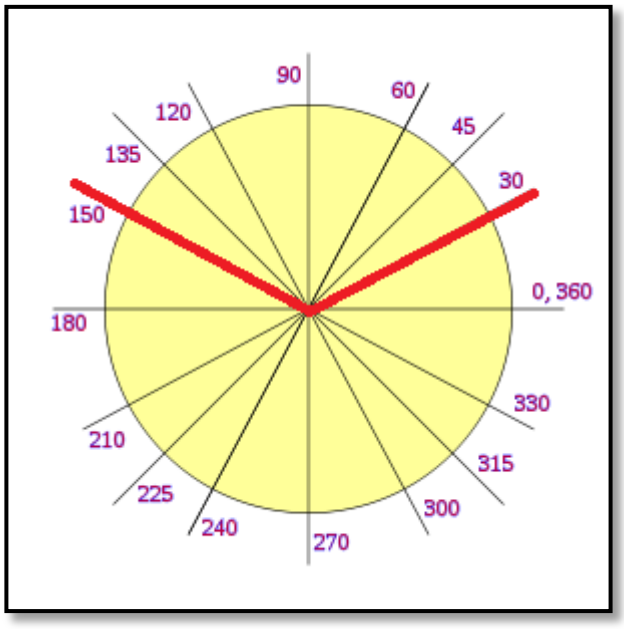

Figura 4-16. Representación de restricción cinemática

Si tenemos en cuenta esta restricción el número de celdas que son aptas va a variar de la situación inicial anterior, quedando como se representa en la Figura 4-17.

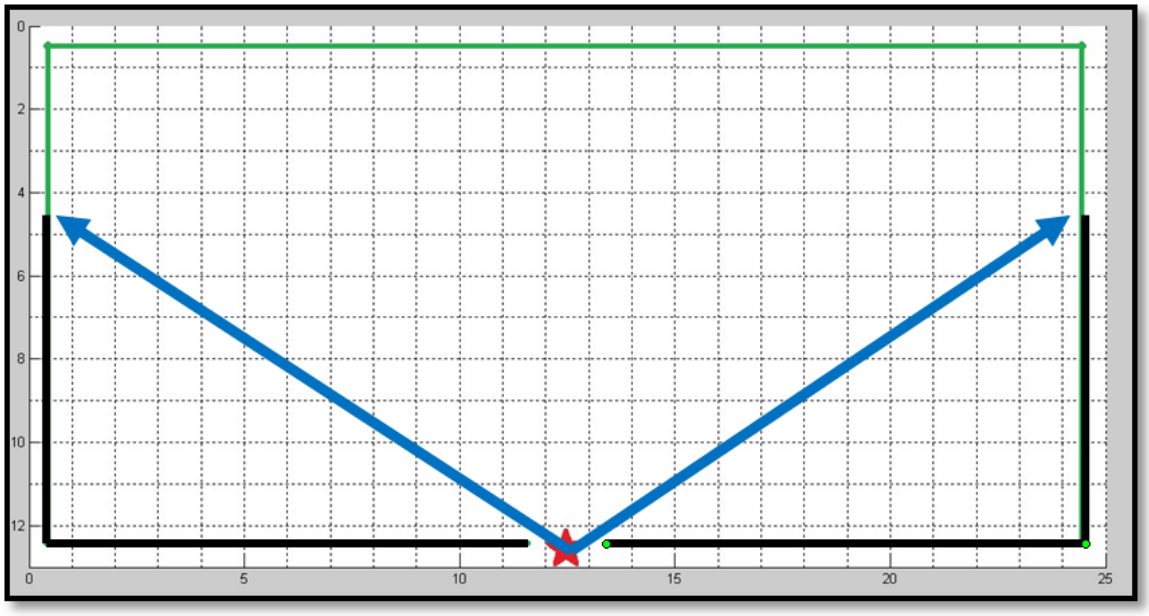

Figura 4-17. Celdas aptas teniendo en cuenta las restricciones cinemáticas. Las casillas en las que realizaremos las búsquedas están marcadas con verde, aquellas que antes eran posibles *waypoints* intermedios y ya no de negro.

4.7. Comprobación de llegada al punto objetivo

¿Cuándo debemos cambiar de punto objetivo? Esto se controla con el módulo de seguimiento de plan de vuelo. Se comprueba cada vez que recibe una nube de puntos, aproximadamente cada segundo, la distancia que existe entre la posición actual del vehículo y la posición o *waypoint* objetivo. Si esta distancia es menor que cierto umbral entonces activamos la condición para cambiar de *waypoint* y dirigirnos al siguiente de la secuencia en el plan de vuelo.

La Figura 4- 18 muestra un cubo verde que representa el *waypoint* intermedio calculado para poder llegar a la posición deseada, representada por un cubo azul. Esta situación es la que tendríamos si el umbral del que hablamos fuera de cero metros, es decir, tendría que llegar exactamente a esa posición. Como ya veremos después eso resulta muy complicado debido al error de las medidas GPS.

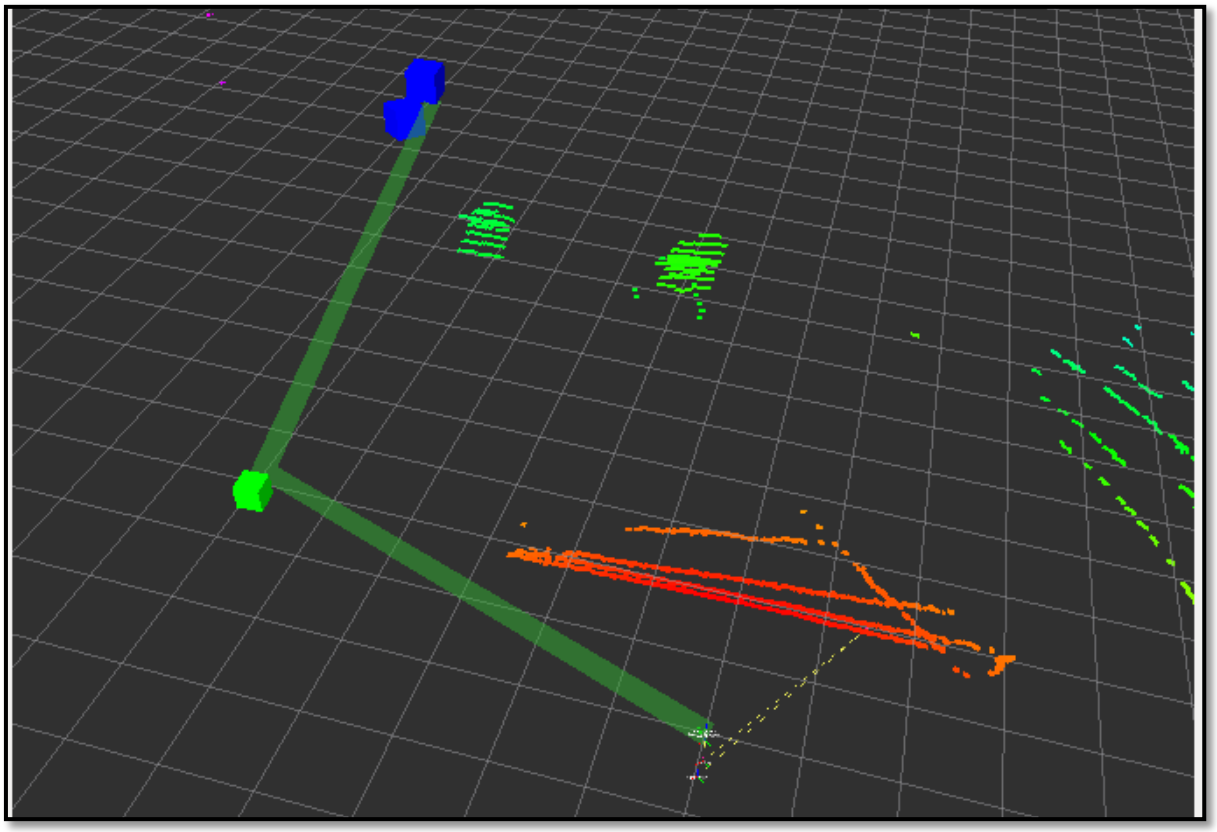

Figura 4-18. Radio cero de distancia al *waypoint*. Escenario con nube de puntos, un *waypoint* intermedio (cubo verde), la posición objetivo (cubo azul) y la trayectoria (lista de cubos verde)

Este umbral es un parámetro que tendremos que concretar dependiendo de la aplicación y el entorno en el que se vaya a desarrollar la actividad. En nuestro caso que vamos a calcular *waypoints* a una distancia de 10 metros, así podemos evitar problemas debido a los errores que introducen las medidas del GPS. La condición será que una vez que la distancia al objetivo sea menor que 4 metros ya consideraremos estar en dicho punto (ver Figura 4-19).

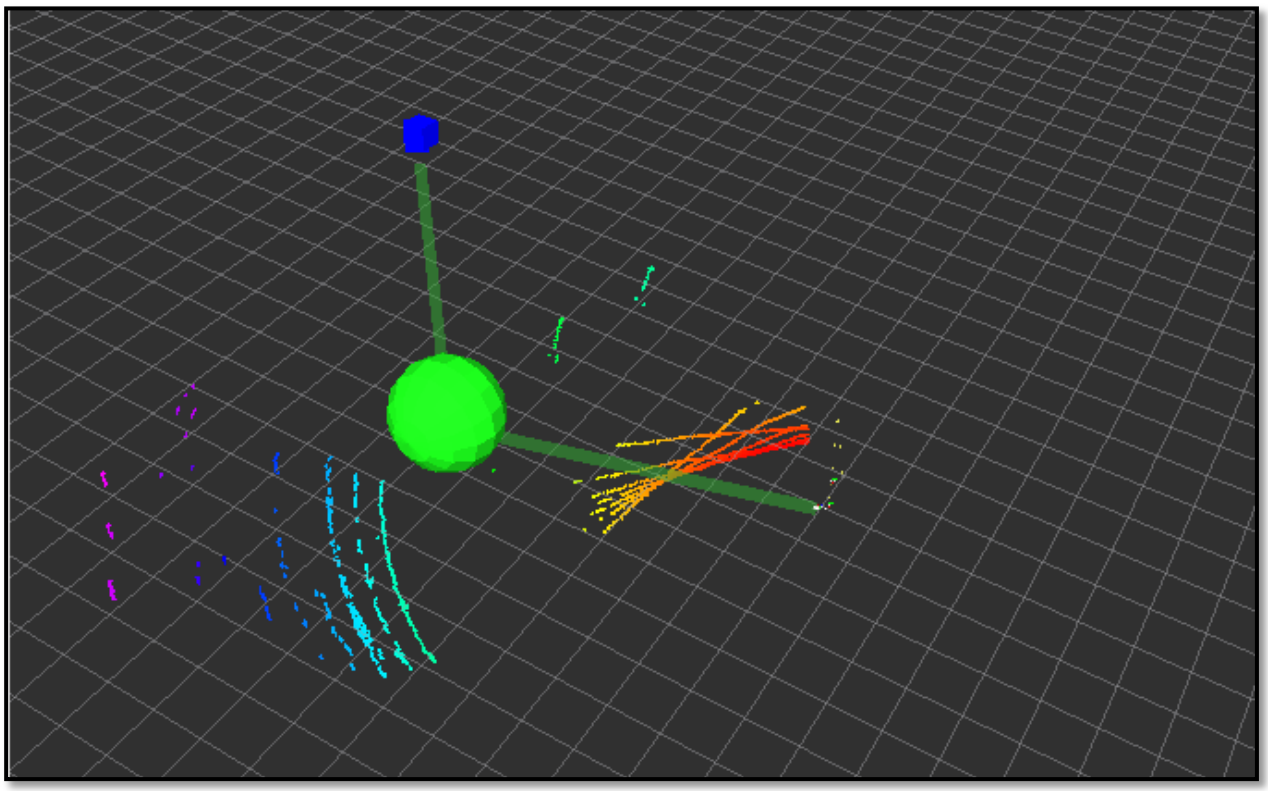

Figura 4-19. Radio cuatro de distancia al *waypoint*. Escenario con nube de puntos, un *waypoint* intermedio (esfera verde), la posición objetivo (cubo azul) y la trayectoria (lista de cubos verde)

Estos 4 metros parecen demasiados, pero tenemos que tener en cuenta aquí el error que introduce las medidas GPS [14]. Este error se debe a que la precisión del GPS es función de la calidad de la señal que recibe, es decir, el número de satélites con los que consigue conectar. Este número varía dependiendo de la zona en la que estemos, no encontraremos buena señal si es interior o está rodeada de muchos edificios, en general, de espacios en los que la visualización del cielo sea reducida.

4.8. Visualización RViz

En nuestro código programamos los publicadores, marcadores y variables necesarias para obtener la visualización más clara y exacta posible de lo que está sucediendo en la simulación.

5.EXPERIMENTOS

La física es un placer.

-Antonio Vega -

Este capítulo escenografia distintas situaciones de vuelo en las que se ha probado el algoritmo implementado, explicar su comportamiento y reflexionar sobre como éste es el acertado. Para ello se presentan simulaciones en RViz.

El escenario está situado en las pistas de baloncesto de la escuela, en ella nos encontramos con estos obstáculos: el suelo, dos canastas y las gradas de césped. También vamos a explicar que significa cada *display*.

En las siguientes figuras aparece un cubo azul, que simboliza el *waypoint* que se ha recibido del plan de vuelo, es decir, la posición por la que debe pasar el UAV. También un cubo verde, que representa el *waypoint* intermedio, que obtenemos con el algoritmo de implementado para evitar las colisiones detectadas. Finalmente, la ruta, señalizada por esas rectas verdes, se compone con la posición del vehiculo, la intermedia si ésta existiera y la deseada.

Caso 1: Trayectoria libre de obstáculos

En la Figura 5-1 la posición intermedia (cubo verde) coincide con la deseada (cubo azul), esto se debe a que entre la posición del vehículo y el *waypoint* objetivo (cubo azul) se cumplen dos condiciones: no existe ninguna influencia de los obstáculos cercanos y no supera la restricción cinemática.

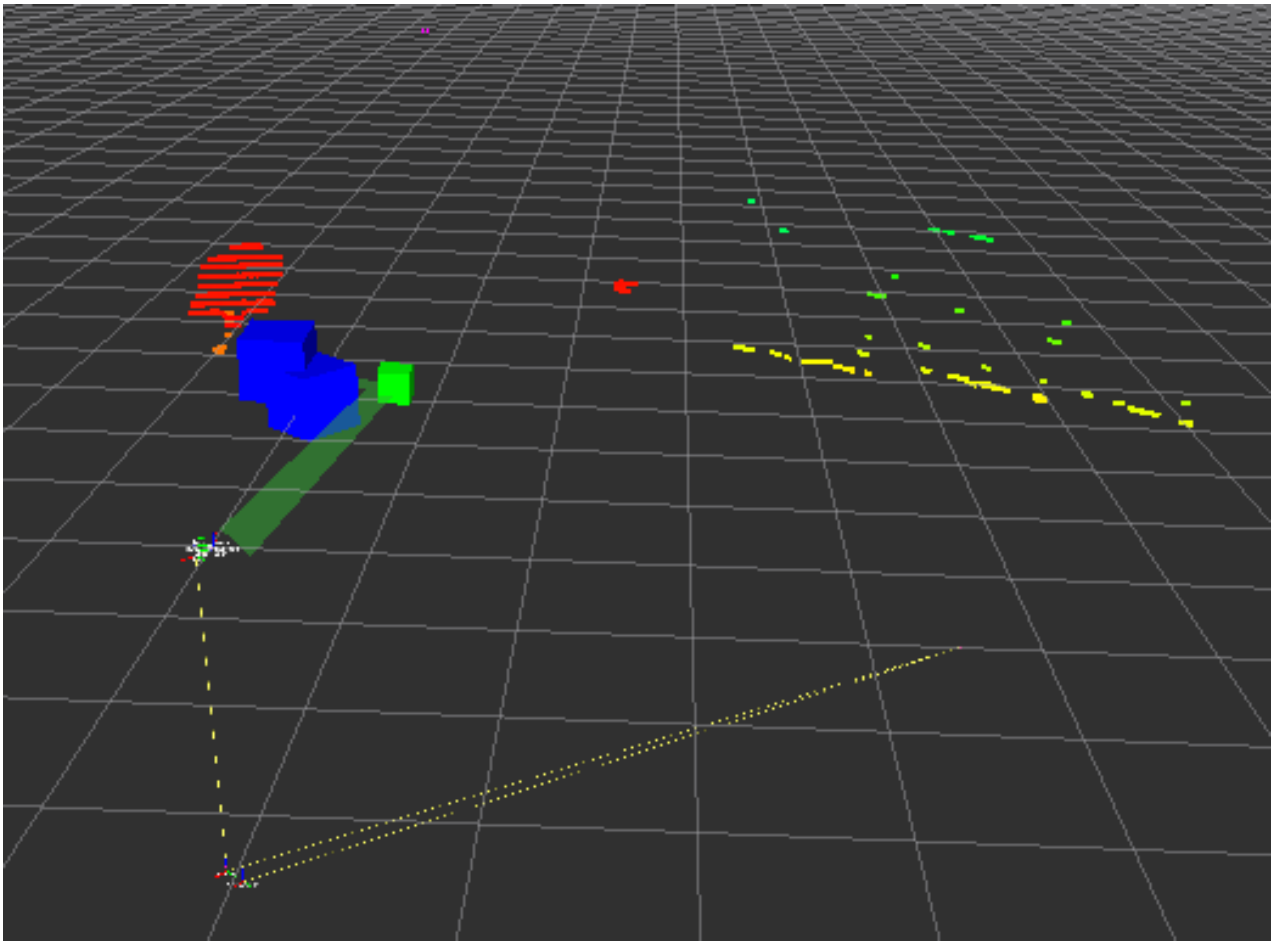

Figura 5-1. Ejemplo Caso 1. Escenario con nube de puntos, un *waypoint* intermedio (cubo verde) que coincide con la posición objetivo (cubo azul) y la trayectoria (lista de cubos verde)

Caso 2: Importancia de la influencia de los obstáculos

Vemos que la ruta (lista de cubos verde) que une nuestra posición actual con la deseada (cubo azul) pasaría entre las dos canastas sin tocar ninguna de ellas, pero una vez que comprobamos si podemos tomar el camino directo, este nos responde que no es posible.

Tenemos que recordar cómo se crea nuestro campo de fuerzas para entender esto. Imaginemos un obstáculo cualquiera, señalaríamos tanto la celda en la que se encuentra como aquellas que estén dentro del radio de seguridad como obstáculo, además tendríamos que obtener la influencia que este obstáculo genera en las cercanías. Podría ocurrir el caso en que dos obstáculos que se encuentren a una cierta distancia sus radios de seguridad se solapen y aunque pensemos que es viable un ruta que pase entre ellos, esta se descartaría.

En la Figura 5-2 se puede observar este efecto debido a la influencia del suelo. El vehículo se encuentra antes de despegar y el *waypoint* intermedio(cubo verde) coincide con la posición del robot, esto se debe a que la posición más favorable que encontramos es la actual. Es fácil percibir que es la influencia del suelo la culpable de que no exista la opción de un camino directo ni un *waypoint* intermedio. La razón es que aunque realmente no se va a encontrar con ningún obstáculo, al haber añadido el radio de seguridad, las celdas más proximas a estos también estarán "marcadas" como obstáculos.

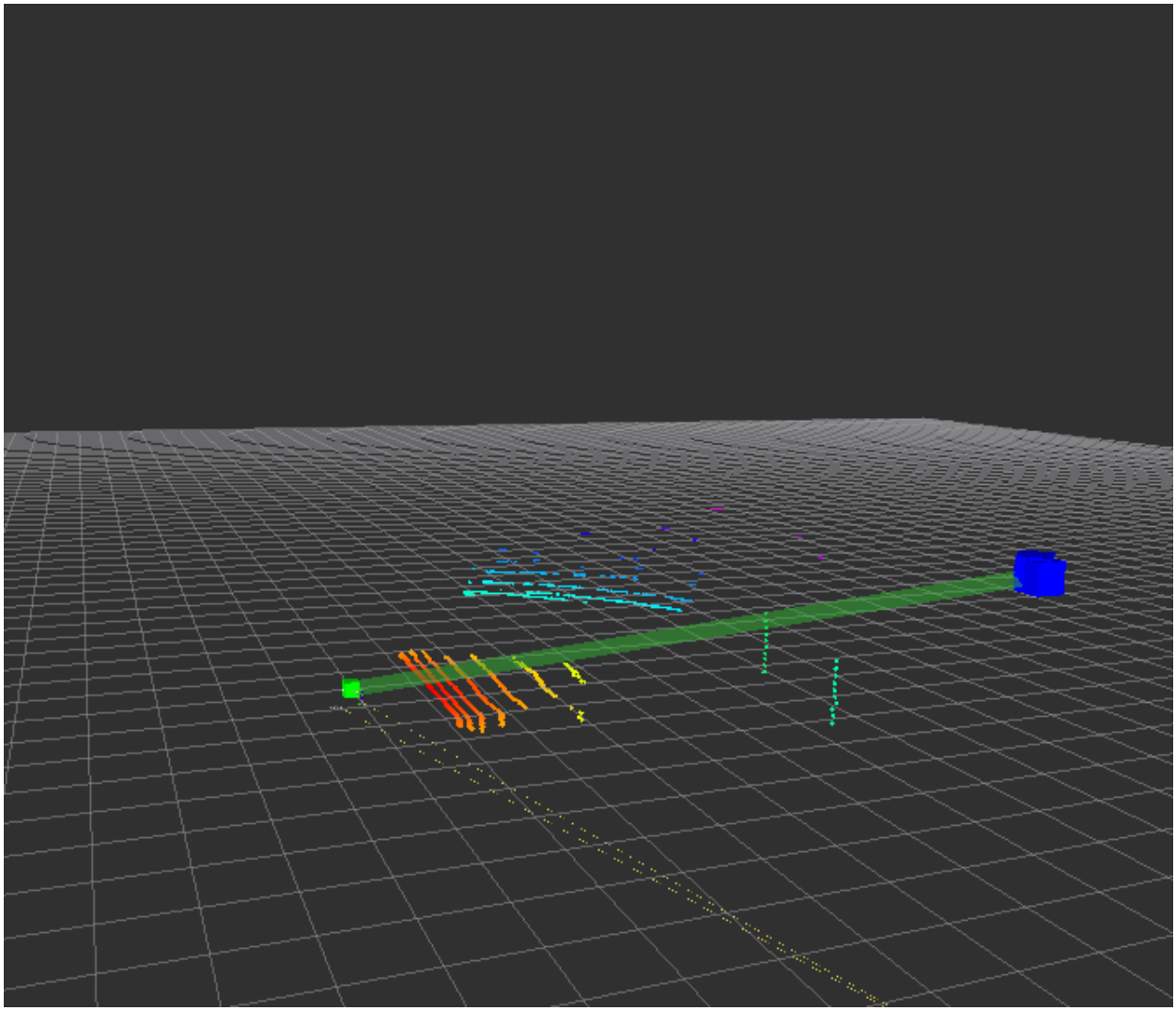

Figura 5-2. Ejemplo de Caso 2. Escenario con nube de puntos, un *waypoint* intermedio (cubo verde), una posición objetivo (cubo azul) y una trayectoria (lista de cubos verde)

Caso 3: Cálculo de un *waypoint* intermedio

La secuencia de acciones que seguiría el algoritmo sería la siguiente: comprobaría en primer lugar si existe un camino directo, esto es fácil de ver en la imagen que no es posible, chocariamos con una de las canastas en esa recta.

Entonces entraría en acción el método de los campos potenciales, obtendriamos la posición más favorable dentro de nuestras celdas a 10 metros y seleccionariamos la más optima. Como podemos ver en la imagen

este *waypoint* (cubo verde) es bastante acertado, pues es el que tiene menor influencia de los obstáculos y también uno de los más próximos a la posición deseada (cubo azul).

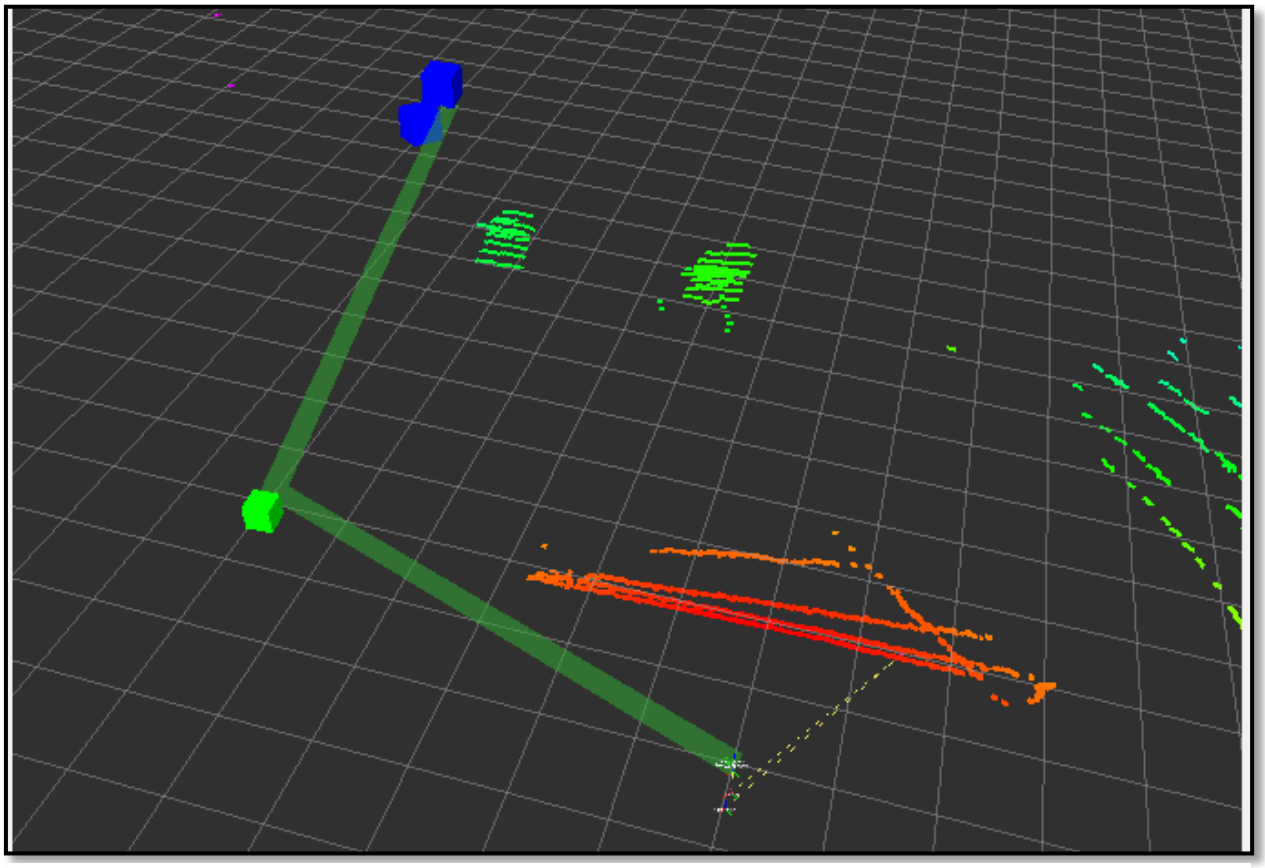

Figura.5-3. Ejemplo 1 de Caso 3. Escenario con nube de puntos, un *waypoint* intermedio (cubo verde), una posición objetivo (cubo azul) y una trayectoria (lista de cubos verde)

Otro ejemplo de cálculo de *waypoint* intermedio (cubo verde), éste con menor influencia de los obstáculos, sigue eligiendo una posición lo más cercana posible a nuestro objetivo (cubo azul).

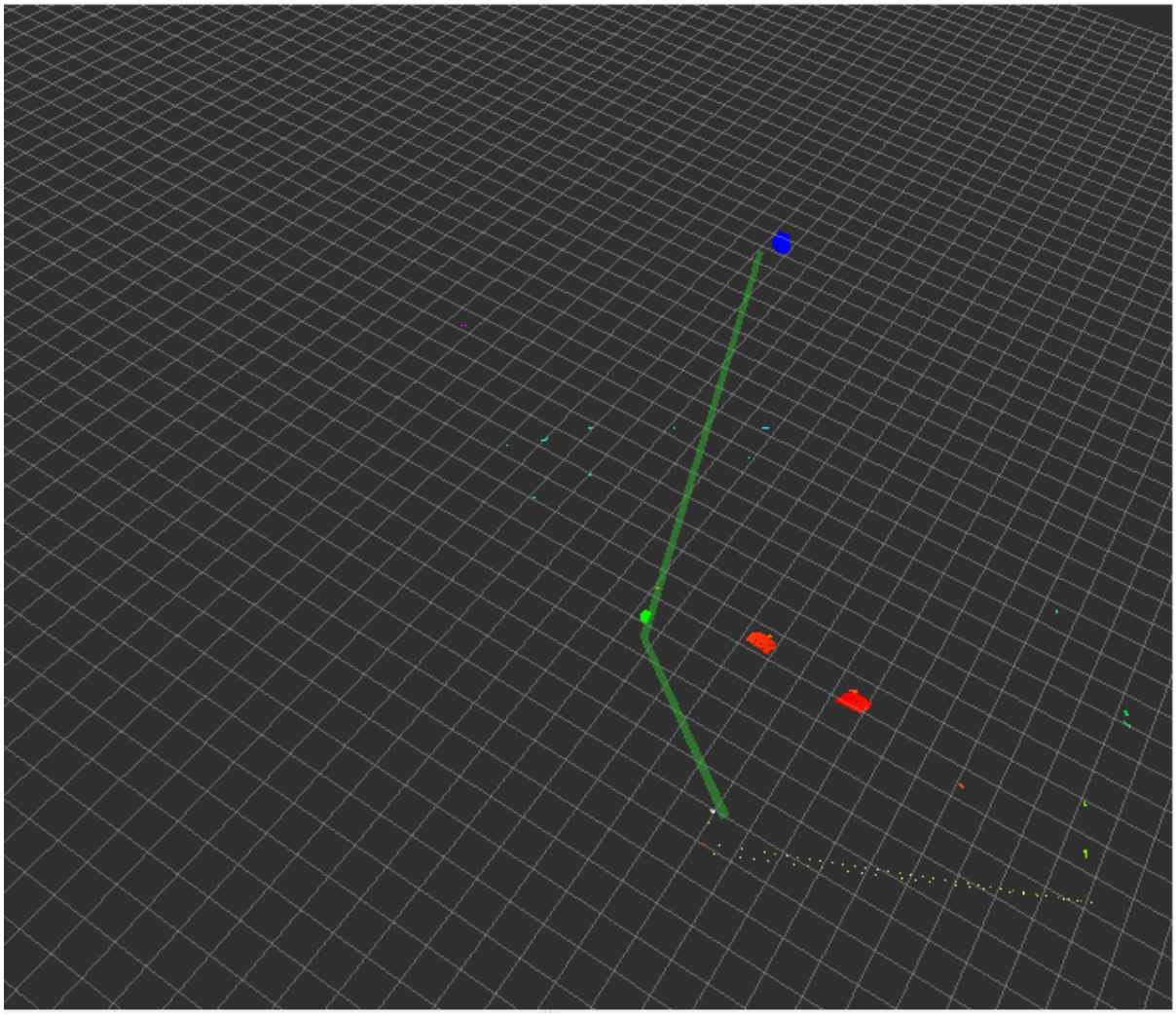

Figura 5-4.Ejemplo 2 de Caso 3. Escenario con nube de puntos, un *waypoint* intermedio (cubo verde), una posición objetivo (cubo azul) y una trayectoria (lista de cubos verde)

6. CONCLUSIONES Y TRABAJO FUTURO

Las tres efes: feo, fuerte y formal.

José María Sanz Beltrán

6.1. Conclusiones

Este trabajo tenía como objetivo implementar un método reactivo basado en el método de los campos potenciales. Se pretende que una vez recibido un plan de vuelo, se puedieran alcanzar las posiciones dadas en el plan de vuelo de forma eficiente y segura.

Para ello se han tratado los datos del entorno obtenidos por el láser y valorado si era necesario activar el método para calcular el *waypoint* intermedio o por el contrario se puede llegar al waypoint de forma directa. Si es necesario este cálculo, esta posición además debe cumplir ciertas restricciones y ya tan solo se debería esperar a que la distancia entre la posición del vehículo y el objetivo fuera lo suficientemente pequeña para pasar al siguiente *waypoint* del plan de vuelo.

El trabajo realizado cumple los objetivos en su mayor medida, siempre dejando una puerta abierta a modificaciones y mejoras.

La importancia de este trabajo la vemos haciendo uso de este algoritmo en cualquier UAV. Este tipo de vehículos tiene ciertas cadenas que frenan su uso, entre ellas, la falta de autonomía. Este código da al vehículo la capacidad de actuar ante obstáculos de forma reactiva siendo una característica que disminuya sus limitaciones y pudiendo así crear aplicaciones nuevas y necesarias.

6.2. Trabajo futuro

Es evidente que existen futuras líneas de investigación en cualquiera de las partes en las que se divide este trabajo. Comenzamos con un código base que se ha ido transformando en lo que tenemos a día de hoy. Siempre intentando mejorar y pulir la mayor cantidad de aspectos posibles. Muchos de estos cambios los podíamos predecir, otros han sido imprevistos. Ahora mismo nuestro proyecto no ha sido probado en un entorno real, es aquí cuando vendrán las complicaciones del directo y tendremos que intervenir.

Quizás uno de los aspectos de estudio más importante y en el cual no hemos indagado mucho es el estudio de los valores de los parámetros, estos modifican notablemente la planificación obtenida. Sería interesante hacer una guía de los valores correctos a elegir dependiendo de la situación a la que nos vayamos a enfrentar.

REFERENCIAS

[1] Ruz Ortiz, J.J. «PLANIFICACION DE TRAYECTORIAS PARA UN ROBOT MOVIL» [En linea] Available: http://eprints.ucm.es/11301/1/MemoriaProyectoSSII.pdf. Universidad Complutense Madrid, 2010.

[2] Pardeiro Blanco, J. «Algoritmos de planificación de trayectorias basados en fast marching square» [En linea] Available:http://javiervgomez.com/files/works/theses/jpardeiro_msc_report.pdf. Universidad Carlos III de Madrid,2015.

[3] «ROS» [En línea]. Available: http://www.ros.org/. [Último acceso: 06 09 2015].

[4] «Robot Operating System NXT» [En línea]. Available:http://blog.electricbricks.com/2011/04/ros-robot-operating-system-nxt-1/. [Último acceso: 06 09 2015].

[5] «ROS: Conceptos » [En linea] Available: http://erlerobotics.gitbooks.io/erlerobot/content/es/ros/ROS-concepts.html [Último acceso: 06 09 2015]

[6] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous uav guidance," *Journal of Intelligent and Robotic Systems, vol. 57, no. 1, pp. 65-100,* 2010.

[7] Sánchez Miralles, A. « Modelos de entorno y planificación de trayectorias » [En linea] Available: http://www.iit.upcomillas.es/~alvaro/teaching/Clases/Programacion/Modelado.pdf. [Último acceso: 06 09 2015]

[8] Muñoz Cueva, A. « Generación global de trayectorias para robots móviles, basada en curvas betaspline» [En linea] Available: http://bibing.us.es/proyectos/abreproy/90085. Universidad de Sevilla, 2014.

[9] Rodríguez Días, L. «Planificación local eficiente de caminos para UAVs en entornos desconocidos» [En linea] Available: http://bibing.us.es/proyectos/abreproy/90015. Universidad de Sevilla, 2014.

[10] Ollero,A. «Planificación de movimientos basada en modelo».Universidad de Sevilla.

[11] « RobotShop» [En linea] Available: http://www.robotshop.com/en/hokuyo-utm-03lx-laser-scanning-rangefinder.html [Último acceso: 06 09 2015]

[12] «Wikipedia» [En linea] Available: https://es.wikipedia.org/wiki/Movimiento_browniano. [Último acceso: 06 09 2015]

[13] Romero Gandul, A.A. «Integración de ROS con Arduino y Raspberry PI» [En linea] Available: http://es.slideshare.net/AlvaroAngelRomeroGan/presentacinpresentationintegracin-de-ros-robot-operating-system-con-las-plataformas-arduino-y-raspberry-pi-y-diseo-de-control-para-robot-mvil

[14] «Wikipedia» [En linea] Available: http://fjalejandre.blogspot.com.es/2014/03/erroresmedicionesGPS.html. [Último acceso: 06 09 2015]

[15] J. Amanatides, A. Woo. "A fast voxel traversal algorithm for ray tracing". En linea. Available: http://www.cse.chalmers.se/edu/year/2011/course/TDA361_Computer_Graphics/grid.pdf. University of Toronto.

[16] «Wikipedia» [En linea] Available: https://es.wikipedia.org/wiki/Sistema_de_posicionamiento_global. [Último acceso: 06 09 2015]

[17] «Mission Planner» [En linea] Available: http://planner.ardupilot.com/. [Último acceso: 06 09 2015]

[18] «ARCAS» [En linea] Available: http://www.arcas-project.eu/. [Último acceso: 06 09 2015]

[19] Delgado Rich, J. «Sistema emebebido en aeronave VTOL para detección y evitación de obstáculos»

[20] «SINTELWEB» [En linea] Available: http://sintelweb.blogspot.com.es/2012_01_01_archive.html [Último acceso: 19 09 2015]

[21] «Carpeta Pedagógica» [En linea] Available: http://cienciasnaturales.carpetapedagogica.com/2009/10/como-funciona-el-gps.html [Último acceso: 19 09 2015]

[22] «Marduk Astronomía» [En linea] Available: http://www.pawean.com/MVM/Coordenadas%20Cartesianas3D.html [Último acceso: 19 09 2015]

[23] K. D. Ramírez Benavides. «Mapeo».Universidad De Costa Rica.

[24] «Theboredengineers» [En linea] Available: http://theboredengineers.com/2012/05/the-quadcopter-basics/. [Último acceso: 20 09 2015]

Glosario

ARCAS: Aerial Robotics Cooperative Assembly Sistem 11

GPS:Global Positioning System 25

PPP: Probabilistic Path Planner 14

PRM:Probabilistic Roadmap 14

ROS: Robot Operating System 11

RPP: Randomized Path Planner 14

RRT: Exploring Random Tree 14

UAV : Unmanned Aerial Vehicle 11

UTM:Universal Transverse Mercator 30