CONTROL MÓVIL DE ROBOTS DRONES

ROBOTS MÓVILES - TRABAJO EN GRUPO

Daniel Lozano Moreno, Juan Guardiola San Román, Mario Almela Rubio, Carlos Bellón García

1. INTRODUCCIÓN.

En esta asignatura, se ha aprendido varias estrategias de control para robots móviles como la locomoción, la evitación de obstáculos, mapeado, etc. Sin embargo, se ha profundizado sobre todo en robots terrestres con ruedas de conducción diferencial, ya que las prácticas han ido orientadas a los Turtlebots, robots de dicho tipo. Otra rama de los robots móviles muy importante de la cual no se ha estudiado lo suficiente su control es la de los robots aéreos. En el presente documento se quiere hacer un estudio del control móvil de los drones o UAVs de la misma forma que se ha estudiado la de los Turtlebots, analizando la técnica habitual ya existente de los algoritmos empleados en la actualidad.

Un drone o vehículo aéreo no tripulado (UAV) se trata de una aeronave autónoma perteneciente a la rama de los robots móviles aéreos. Inicialmente fueron desarrollados para el ámbito militar durante la I Guerra Mundial [1], pero actualmente son también utilizados en otros ámbitos como el civil. Podemos encontrarlos tanto en el reconocimiento de personas y combate como en ocio, seguridad, logística, geología, etc.

El objetivo de este trabajo es adquirir la información suficiente como para controlar un drone en ROS de forma básica tal y como se ha hecho en prácticas con los Turtlebots. Debido a que según la locomoción del drone varía la forma de controlarlo, se analizará concretamente el control de un drone quadrotor (Apartado 3), al igual que en prácticas se ha estudiado un robot terrestre específicamente de ruedas diferenciales.

2. LOCOMOCIÓN

Los drones necesitan usar un sistema de locomoción distinto al usado en robots móviles terrestres ya que, debido a que navegan por el aire, en lugar de usar el rozamiento con el suelo para desplazarse usan el rozamiento con el aire. Usan como sistema de locomoción hélices movidas mediante motores y pueden tener o no alas. [2] Podemos encontrar dos formatos de locomoción principales:

- **Ala fija**: similar a los aviones, usan alas para orientarse y un o más motores con hélice para ganar la propulsión necesaria para moverse.
- **Multirotor**: necesitan mínimo tres hélices para propulsarse y para orientarse no usan alas sino un cambio de inclinación en el ejes de las hélices o haciendo que las hélices giren a distintas rpm.

3. POSICIONAMIENTO

El problema que tienen los robots aéreos frente a los terrestres radica en las dimensiones que necesita para posicionarse en su entorno. Los robots terrestres, en su mayoría de los casos, necesitan tan solo dos dimensiones para situarse en un mapa, la latitud y la longitud, pero la capacidad de vuelo de los drones les

permiten usar una dimensión más, la altitud, por lo que necesitarán sistemas de posicionamiento que le ayuden a situarse en entornos 3D.

3.1. POSICIONAMIENTO EN LATITUD Y LONGITUD

Los sensores usados para obtener la posición del drone en latitud y longitud coinciden con algunos usados en robots terrestres. [3] Para entornos exteriores, los drones usan sistemas de posicionamiento satelital como los GPS, permitiendo conocer su posición global respecto a un mapa. En entornos interiores donde los GPS no tienen cobertura necesitan otras alternativas como:

- Balizas: las balizas activas son unas emisoras de señales de radio que pueden ser usadas en drones mediante triangulación de señal instalándose en lugares específicos de un mapa.
- Sensores láser: medidores de distancia láser.
- Sensores visuales: miden la posición en base a variaciones de imágenes captadas por cámaras.

3.2. POSICIONAMIENTO EN ALTITUD

Con esta tercera dimensión, para que el drone se ubique correctamente en el espacio deberá conocer su altitud e inclinación. Para conocer la altitud se usan los barómetros, instrumentos que miden la presión atmosférica del entorno, de manera que, cuanto menos presión detectada, a mayor altitud se encuentra el robot.

La inclinación es el ángulo del robot respecto a la tierra y puede ser medida mediante inclinómetros. Los sensores de inclinación consisten en un cilindro con algo en su interior, un contacto eléctrico en en el centro de la base y la pared constituye en otro contacto eléctrico. Al inclinar lo suficiente el dispositivo, ese algo hará un puente entre ambos contactos cerrando el circuito [4]. Podemos encontrar sensores de inclinación que funcionan mediante una gota de mercurio o mediante dos esferas conductoras.

4. SIMULACIÓN EN ROS DE UN DRONE, OBJETO DE ESTUDIO

ROS ya cuenta con un modelo URDF para un drone multirotor de cuatro hélices (quadrotor) para su simulación llamado *hector_quadrator* (Figura 1) [5]. Su URDF se puede encontrar aquí. Este robot será el objeto de estudio para su control en este trabajo.

Este UAV usa los plugins *hector_quadrotor_gazebo_plugins* que proporciona complementos específicos para su simulación en gazebo. Tiene un controlador que permite el control del UAV usando mensajes *geometry_msgs/Twis*t sobre el topic 'cmd_vel', como se usa en otros robots pero añadiendo la velocidad de ascenso (la tercera dimensión integrada). Los complementos de propulsión y aerodinámica simulan el sistema de propulsión (motores y hélices) y la resistencia, respectivamente tomando los voltajes del motor y el vector del viento como entrada.

Como sensores de posicionamiento, cuenta con:

- **GPS**: utiliza la información verdadera terrestre del gazebo para localizar al robot en entornos al aire libre, simulando la disponibilidad de señales GPS en entornos exteriores.
- **Cámara monocular**: será usada como sensor visual para el posicionamiento en latitud y longitud en entornos interiores.
- **Barómetro**: el plugin *hector_gazebo_ros_baro* incorpora la simulación de un sensor barométrico para conocer la altitud a la que se encuentra [6].

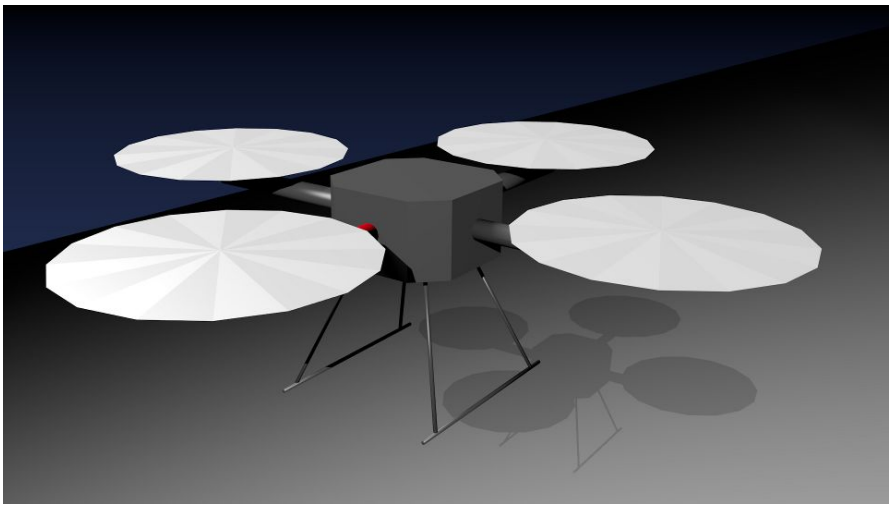

Figura 1: renderizado del URDF del quadrotor que proporciona ROS.

5. ODOMETRÍA

Para el robot es necesario conocer su posición en un momento exacto después de realizar un desplazamiento. Esto, entre otras opciones, se consigue mediante la odometría. La odometría utiliza los sensores de posicionamiento del robot para determinar su posición actual en base a lo que se ha desplazado este robot. En robots terrestres, se usan encoders en las ruedas para calcular en base a las vueltas dadas calcular su posición. En el caso de robots que no tengan su movimiento limitado al suelo como los robots aéreos se deben explorar otros tipos de odometría. Como el robot objeto de estudio posee como sensor una cámara, se usará el método de la odometría visual para solucionar el problema.

La odometría visual [7] consiste en determinar la posición y la orientación de un robot mediante el análisis de las imágenes de las cámara, para ello, se examina los cambios que los movimientos del robot provocan en la imagen de las cámara. La principal ventaja de esta técnica es que ni el diseño del robot ni las perturbaciones en el terreno influyen en las medidas. Su principal contra es que se requiere que la iluminación del entorno sea suficiente para detectar las texturas del entorno y sea capaz de reconocer estos.

El proceso de un sistema de odometría visual consiste en secuenciar las imágenes que toma la cámara del robot, y mediante programas o librerías como OpenCV, se extraen los Features y se emparejan con esos mismos features del frame siguiente. Una vez realizado este matching, se procede a realizar la estimación de movimiento para determinar cuánto se ha desplazado el robot. Esta estimación de movimiento puede realizarse de tres diversas maneras según el modelo a emplear: 2D to 2D, 3D to 3D y 3D to 2D. [8]

Por un lado, existe la técnica de la cámara estéreo basada en tomar fotogramas de manera tridimensional utilizando diversas cámaras configuradas en estéreo. Lo que permite, mediante ciertos detectores de puntos de interés de la imagen como Frostner o Moravec y el cálculo de la estimación de trayectoria, permitan determinar el desplazamiento del robot. A lo que luego se le puede aplicar RANSAC para rechazar los valores atípicos. [8]

Por otro lado, existe la cámara monocular, que como su propio nombre indica funciona solamente mediante una cámara y solo trabaja con la estimación de la pose en 2D. Mediante este tipo de estimación de pose en 2D no se puede conocer la escala del entorno. Pero de manera similar a la cámara en estéreo, se utilizan detectores de puntos de interés en frames sucesivos para determinar el movimiento y lo refinaban mediante el

algoritmo RANSAC [8]. Este es concretamente el tipo de cámara que usa el robot quadrotor que se está estudiando.

Para todos los tipos de odometría visual su implementación [3] es muy similar, solo difiriendo si trabaja con 1 o 2 cámaras, y si trabaja en 2D o 3D. Pero a rasgos generales, la ejecución del programa sigue el mismo orden y de manera cíclica:

1. **Captura de fotograma:** Mediante la cámara se captura una instantánea del entorno.
2. **Obtención de puntos clave:** Mediante la librería OpenCV y un extractor (Harris, Sift, Surf…) se extraen los puntos de interés de la imágen, tanto unos puntos en 2D o una nube de puntos en 3D. Los puntos de interés suelen ser zonas de alto contraste, patrones distinguibles, esquinas…
3. **Emparejamiento:** Se emparejan los puntos de interés de dos frames consecutivos y se obtiene la matriz de homografía.
4. **Transformación de puntos en base a pose:** Este paso mejora la precisión del algoritmo mediante la transformación de la nube de puntos a la perspectiva usando la última pose, pero a consecuencia de este paso el algoritmo pierde velocidad.
5. **Cálculo de desplazamiento:** Finalmente se calculan los desplazamientos y el giro utilizando la matriz de homografía.

Con el algoritmo anterior se obtiene el desplazamiento que hay entre imágenes, con estos datos se puede corregir los errores de medida del desplazamiento mediante la conversión de estos datos que suelen estar en píxeles a medidas del entorno real como por ejemplo metros.

También cabe destacar que hay que convertir el desplazamiento en píxeles a un desplazamiento en una medida del entorno como metros o centímetros, para ello se debe utilizar las funciones de la figura 2, que relacionan los parámetros intrínsecos de la cámara utilizada, siendo F la distancia Focal en metros, y dx y dy son los desplazamiento sobre el eje x e y respectivamente en píxeles:

$$d_X = F \cdot \frac{d_x}{Z} \qquad d_Y = F \cdot \frac{d_y}{Z}$$

Figura 2: fórmula para convertir desplazamiento en píxeles a desplazamiento en metros

Finalmente, se puede observar la implementación de la odometría en este pseudocódigo, donde se observa los pasos, y mediante las funciones se realizan los pasos necesarios para obtener la odometría del robot.

```
odometria (frame, frame_previo):
    keypoints_previo = detector_keypoints (frame_previo)
    keypoints = detector_keypoints (frame)
    correspondencias = matching(keypoints, keypoints_previo)
    homografia = pose(correspondencia)
    deplazamiento = calcular_desplazamiento(homografia)
    desplazamiento = pix2m(desplazamiento)
    return desplazamiento
```

Aunque la odometría retorna resultados bastante buenos, necesita una confirmación en base a las medidas de los sensores externos del drone, ya que una ligera inclinación del robot hace que varíe la imágen y detecte desplazamiento del robot, pero este no se ha movido del sitio.

Para ello se utiliza una combinación de las medidas de desplazamiento tomadas mediante odometría visual, se le aplican los datos tomados con los sensores para obtener unos datos con una mayor verificación, generando una mayor seguridad al drone

6. EVITACIÓN DE OBSTÁCULOS

Un algoritmo de evitación de obstáculos es necesario para que el UAV pueda navegar por el aire de forma autónoma sin chocarse con los objetos. Este algoritmo debe de ser capaz de realizar dos tareas: detección del obstáculo y evitación del obstáculos.

6.1. DETECCIÓN DEL OBSTÁCULO

Los sensores que posee el drone quadrotor que puedan ayudar a realizar esta tarea es la cámara de visión monocular y el barómetro.

Los sensores monoculares permiten realizar una construcción en profundidad de un escenario plasmado en una imagen 2D ya que, según la visualización del entorno a través de la cámara, se puede conocer a cuanta distancia se encuentran los objetos. Cuanto más lejos se encuentre un objeto del drone, más arriba se plasmará el objeto en la imagen captada por la cámara, y cuanto más cerca, más abajo (figura 3). Dicho de una manera muy simplista, el algoritmo de detección de obstáculos compara la imagen capturada por la cámara de visión monocular con sus claves de profundidad pictórica [9].

Figura 3: representación de una pelota captada por una cámara cuando está lejos y cuando está cerca.

6.2. EVITACIÓN DEL OBSTÁCULO

Un ejemplo de algoritmo de evitación de obstáculos son los algoritmos tipo Bug [10], llamados así porque emulan el comportamiento de los insectos cuando encuentran un obstáculo. Esta reacción consiste en bordear el contorno del obstáculo hasta encontrar un camino que se pueda seguir para llegar al objetivo final. De este algoritmo existen diversas variantes

La primera variante, Bug1, está basada en en los algoritmos originales de V. Lumeslsky y A. Stepanov [11]. consiste en caso de obstáculo, el robot almacena la posición del obstáculo detectado previamente. Una vez

detectado el objeto, el robot recorre el contorno del robot y almacena la posición que está más cerca del objetivo. Al acabar de recorrer el objeto, el robot se sitúa en la posición almacenada previamente y se encamina al objetivo donde el robot se dirigía previamente.

Tiempo después, los mismos desarrolladores de Bug1, desarrollarían Bug2. El funcionamiento del algoritmo es similar pero con una gran diferencia. Una vez detecta el obstáculo, se calcula una línea imaginaria entre la posición inicial del robot y la posición objetivo. Seguidamente, el robot comienza a bordear el obstáculo hasta que se sitúe en el punto que intersecta la línea calculada previamente. Cuando llega al punto de intersección el robot retoma su trayectoria hasta el punto objetivo.

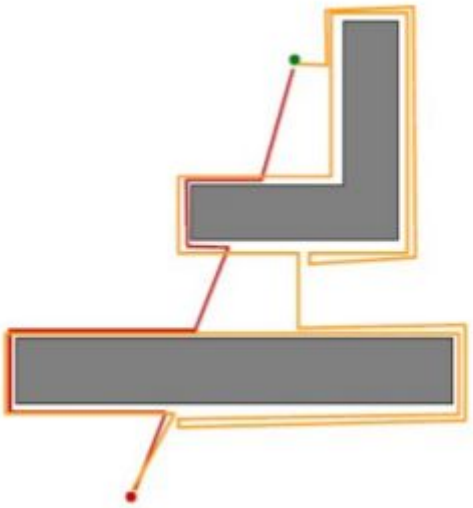

Figura 4: Comparacion Bug1 (amarillo) y Bug2 (rojo)

7. MAPEADO

La generalización de los algoritmos de mapeado al ámbito tridimensional es una tarea importante al tiempo que ardua. En el área de los robots voladores no tripulados existen dos vertientes principales de algoritmos de mapeado: por un lado, los algoritmos de mapeado bidimensional siguen siendo empleables en varias situaciones, como en la generación de mapas de terreno mediante drones [12] o la navegación por interiores donde las paredes tienen la misma altura (en cuyo caso la navegación por el aire es equivalente a la navegación por el suelo siempre que se respete la altura máxima del techo) [13][14]. En el caso de navegación autónoma por terrenos angostos y/o desconocidos, un algoritmo de mapeado en 3D puede ser la única solución viable para impedir que el dron colisione con los obstáculos. En este caso, también existen algoritmos de mapeado válidos basados en sensores de rango 3D [10].

7.1. ALGORITMOS DE MAPEADO 2D

Este tipo de algoritmos se emplea cuando la altura del dron no es un factor importante, como en tareas de mapeado del entorno externo para búsqueda o detección de incendios (lo único que importa en esta tarea es el terreno, el dron se presupone lo suficientemente alto como para no colisionar con nada en el aire), o en navegación en interiores, donde la altitud del dron suele establecerse alrededor de la mitad de la altura de las paredes del entorno o a la altura de los ojos de la persona media.

En el caso concreto del mapeado para búsqueda, se suelen emplear algoritmos basados en imagen. Los sensores de rango no son muy empleados debido a la gran altitud de los drones que realizan esta tarea y a la

necesidad de localizar determinados elementos en el mapa (como personas perdidas, incendios, etc.). Realmente esta tarea no genera un mapa real del entorno, sino imágenes bidimensionales de este que posteriormente se analizan en busca de ciertos elementos.

En este caso, una de las opciones [12] consiste en dividir el entorno en cuadrantes del mismo tamaño y recorrerlos sistemáticamente (Figura 5). A medida que el dron se mueve, se van tomando fotografías del cuadrante que posteriormente se enlazan unas con otras para formar una imagen continua, en un proceso llamado *stitching*.

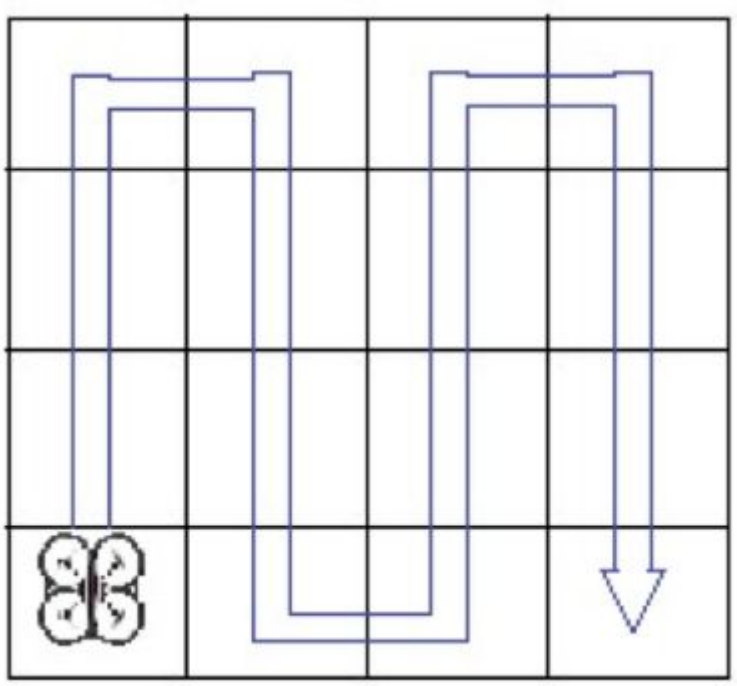

Figura 5: Terreno a mapear dividido en cuadrantes recorridos sistemáticamente [12]

El proceso de *stitching* es complejo, por lo que se suele realizar una vez todas las imágenes han sido tomadas. Suele tener además varias fases, una en la que se unen las imágenes pertenecientes a una misma columna o dirección de movimiento del dron, y otra donde se unen varias columnas o filas. Para unir dos imágenes juntas, se emplea un *pipeline* de visión por computador en el que se extraen características (*keypoints*) en ambas imágenes mediante un algoritmo como SIFT o SURF. A continuación, se asocian los *keypoints* de ambas imágenes en busca de correspondencias. Las correspondencias positivas se emplean para calcular una transformación coherente entre el par de imágenes, que por último se aplica y se guardan las dos imágenes como una sola [12]. Uno de los algoritmos más empleados para *stitching* es el incluido en la librería BoofCV. No obstante, este algoritmo distorsiona las imágenes mediante transformaciones proyectivas y rellena el espacio vacío con un fondo negro que interifiere a la hora de unir una tercera imagen. Es por esto que en muchas aplicaciones de este tipo se opta por diseñar un algoritmo propio, menos robusto en la unión de imágenes pero sin demasiada distorsión [12].

Un posible pseudocódigo para este tipo de algoritmos sería:

```
2d_env_Mapping():
images = []
        while not all_quadrants_visited:
                image = TakePicture()
                images.append(image)
```

```
                MoveToNextQuadrant()
                if MoveToNextQuadrant() == null:
                        all_quadrants_visited = True
finalImage = images[0]
for i in range(1, len(images)):
        finalImage = Stitching(finalImage, images[i])
return finalImage
```

Otro caso donde un mapeo 2D resulta útil es en tareas que involucren desplazamiento en interiores. En este caso, los drones emplean sensores de rango como LIDAR. No es necesario que estos sensores devuelvan información 3D, basta con una "rebanada" del entorno a una determinada altura para determinar obstáculos como paredes o pilares, mientras la información de altitud se extrae directamente de los sensores de altitud como barómetros integrados en el dron [13][14]. El resultado es un mapa bidimensional muy similar al obtenido en sistemas de SLAM 2D estudiados durante la asignatura, como el famoso AMCL (véase sección 8) [13]. Se puede incluir información de altura de cada borde del mapa mediante un código de colores representando diferentes niveles de altitud [13]. A la hora de navegar, se puede fijar un límite de altura para impedir que el dron choque contra el techo. Sensores adicionales de distancia pueden emplearse en entornos desconocidos para determinar la distancia al suelo o al techo e impedir un choque, no obstante esta información no queda registrada en el mapa y se controla en tiempo real mientras el dron vuela (se controla que la distancia al suelo/techo no supere cierto umbral, y se rectifica la altura del dron para corregir malos posicionamientos). El mapa únicamente registra elementos con una altura tal que no pueden ser sobrevolados de forma normal.

En este enlace se puede observar un mapeo 2D realizado por el drone haciendo uso del indoor SLAM, el cual usa el metapackage *hector_slam* que es capaz de crear mapas 2D con un láser LIDAR.

7.2. ALGORITMOS DE MAPEADO 3D

Estos algoritmos ofrecen la máxima calidad a la hora de operar con robots voladores. No están restringidos a espacios cerrados y pueden generar mapas de relieve del terreno y obstáculos con grandes niveles de detalle. Para esto se emplean sobretodo sensores de rango 3D como LIDAR 3D, a veces hasta más de uno al mismo tiempo. El problema de estos algoritmos es la elevada carga computacional que presentan y la ingente cantidad de datos a procesar. Por esta razón, una representación del mapa lo más eficiente posible es esencial.

En este caso se ha considerado un *pipeline* novedoso, caracterizado por su eficiencia y precisión [15]. Este *pipeline* consta de varias fases en la etapa de mapeado, y también soporta localización. El mapeado empieza con varios (al menos 2) sensores de rango 3D con un sistema de referencia común. Las nubes de puntos generadas por estos sensores son propensas a acumular el error local a medida que se unen en nubes de puntos más grandes, problema que precisa solución. En este caso, se emplean mecanismos de correspondencia punto a punto (en las regiones de las nubes de menor densidad) y punto a plano (en las regiones más densas) entre pares de nubes de puntos con una estrategia de correspondencia jerárquica. Para la minimización de la función de correspondencia entre puntos se tiene en cuenta no sólo su posición sino también la orientación relativa de sus sistemas de referencia. Una alternativa a este procedimiento es FaMSA (*Fast Multi-Scan Alignment*), una técnica de alineamiento de nubes de puntos donde el historial de emparejamientos se emplea como conocimiento para acelerar futuros emparejamientos (véase Figura 6) [15][16].

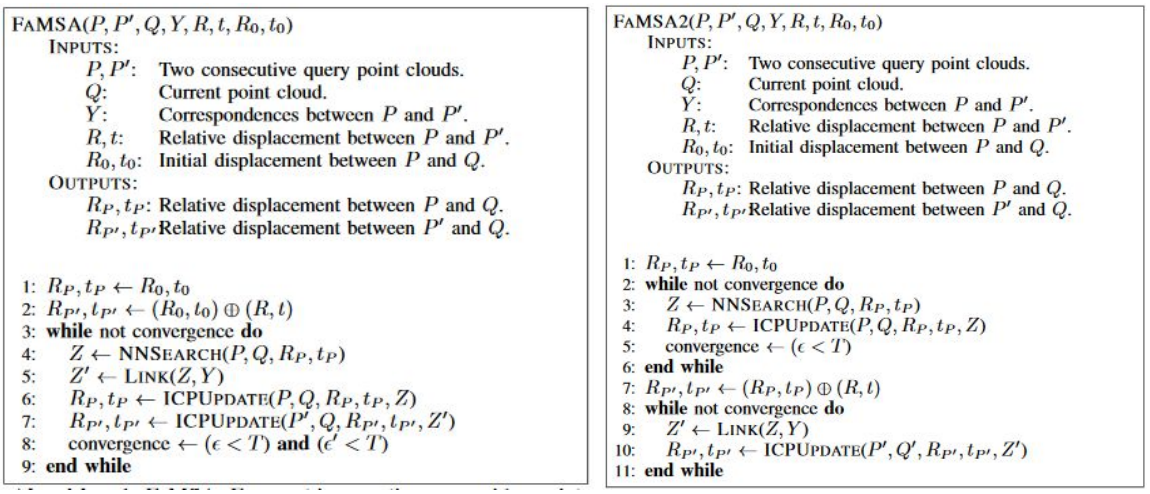

Figura 6. Dos versiones del algoritmo FaMSA, rápida (derecha) y ultra rápida (izquierda) [16]

Una vez elegido el método de unión de nubes de puntos, el siguiente paso en este *pipeline* de mapeado es la generación de un mapa globalmente consistente. Para esto, se considera el alineamiento probabilístico de las nubes de puntos en su conjunto (después de unirlas) para determinar su coherencia global mediante un filtro de información con una ventana temporal de estados (EIF). En este caso, se trata de una adaptación del algoritmo Pose SLAM, un algoritmo de mapeado 3D donde solo se estima la trayectoria del robot y la información sensorial se emplea únicamente para restringir la posición relativa de dos poses sucesivas. Esta técnica produce mapas de alta calidad a costes computacionales aceptables [15].

Por último, una vez generado el mapa, se emplean algoritmos de clasificación avanzados para determinar la transitabilidad del terreno (esto último no es tan necesario en el caso de drones, pero este *pipeline* es aplicable también a robots móviles terrestres). Una vez determinado el terreno transitable, se emplean generadores de trayectorias basados en el famoso A* combinado con políticas de coste híbridas que penalizan la expansión del árbol cerca de obstáculos y tienen en cuenta la cinemática del robot [15].

En este enlace se puede observar un mapeo 3D usando el presente drone. El resultado del mapa se visualiza como un mapa de colores los cuales los más cálidos representan la mayor altura y los fríos la menor.

8. LOCALIZACIÓN

En cuanto a la localización del dron, cabe destacar que no es una tarea sencilla. Debido a que el dron es un robot aéreo sin restricciones de movimiento, habremos de usar avanzados algoritmos y sensores que no dependan de estas restricciones. Muchos de los métodos de mapeado de drones son de hecho usados para la localización de estos mismos, no obstante, también se podrán usar algoritmos de localización en situaciones en las cuales falle el GPS. Cuando nuestro robot busca ser localizado pueden darse principalmente dos situaciones: que el dron sepa su posición inicial respecto al mundo, o el conocido “kidnapped problem”, el cual se puede solucionar mediante el algoritmo AMCL.

El algoritmo AMCL es un algoritmo de localización probabilística basado en Monte Carlo. Primero empieza con una distribución de partículas a lo largo de todo el espacio de configuración. Tras esto comienza la etapa de predicción, en la cual cuando el robot ejecuta un movimiento, desplaza la partícula para predecir el nuevo estado del robot tras el movimiento. Cuando el robot percibe su entorno, las partículas se actualizan usando una estimación Bayesiana que depende de cómo se correlacionan las medidas actuales con el estado predecido. Esto se conoce como etapa de actualización.

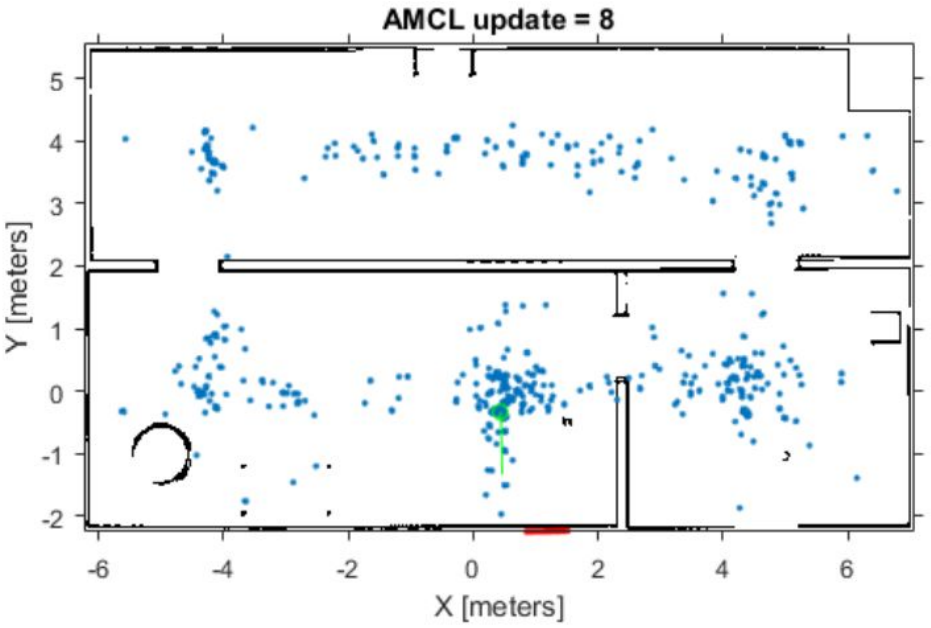

Figura 7: Localización del dron en un mapa a través de AMCL

Para lograr una mejor localización del dron, combinaremos nuestro algoritmo de localización con los datos que nos proporcionan los múltiples sensores del robot. Para ello, es necesario establecer como input las medidas del entorno captadas a través de estos sensores y el mapa 3D del entorno, y como output tras realizar nuestra localización tendremos y la posición estimada de nuestro robot en este entorno. Usaremos el algoritmo AMCL previamente descrito, usando en la etapa de predicción la odometría del robot obtenida a través del GPS, la odometría X-Y obtenida de las imágenes y la odometría del eje Z a través del barómetro, corrigiendo la orientación del robot proporcionada por una IMU, para realizar las predicciones de las partículas en el filtro de partículas. En la etapa de actualización, las partículas serán actualizadas haciendo uso de las medidas del GPS y del mapa 3D cargado inicialmente, que es resultado del mapeado previo del robot [17].

La IMU no es un sensor proporcionado por el drone que se está estudiando, pero como es preciso para esta tarea, puede ser integrado perfectamente. El mismo creador del URDF del drone también tiene un plugin *hector_gazebo_plugin* [18] que otorga numerosos sensores, entre ellos la IMU con el nombre de GazeboRosImu . GazeboRosImu es un reemplazo del complemento *GazeboRosImu* en el paquete *gazebo_plugins*. Simula una unidad de medida inercial afectada por el ruido gaussiano y la deriva aleatoria de baja frecuencia. La orientación devuelta imita un sistema de referencia de actitud y rumbo (AHRS) simple que utiliza velocidades y aceleraciones (erróneas).

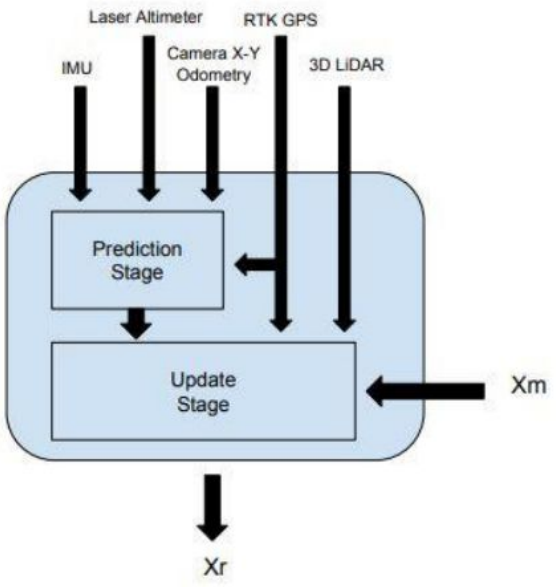

Figura 8: Esquema de la localización del dron

9. VIDEO PRESENTACIÓN

https://drive.google.com/file/d/1nVJromUvHwTRfu6cHFyYDAumiZGCFkQD/view?usp=sharing

10. REFERENCIAS

- [1] REVISIÓN TECNOLÓGICA, NORMATIVA Y APLICACIONES DE LOS VEHÍCULOS AÉREOS NO TRIPULADOS EN LA INGENIERÍA (Parte 1) (2016) *Historia de los drones* https://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado#Clasificaci%C3%B3n
- [2]TIPOS DE DRONES SEGÚN SU LOCOMOCIÓN (2018). *¿Qué diferencia hay entre un cuadricóptero y un dron de ala fija?* https://www.todrone.com/cuadricoptero-dron-ala-fija/
- [3] SISTEMA DE POSICIONAMIENTO PARA UN DRONE (2015): https://riull.ull.es/xmlui/bitstream/handle/915/1137/Sistema+de+posicionamiento+para+un+drone.pdf?sequence=1
- [4] INCLINÓMETROS - SENSORES DE INCLINACIÓN (2015). *Medir inclinación con arduino y sensor TILT SW-520D:* https://www.luisllamas.es/medir-inclinacion-con-arduino-y-sensor-tilt-sw-520d/
- [5] HECTOR QUADROTOR DESCRIPTION (2012) http://mirror.umd.edu/roswiki/hector_quadrotor_description.html
- [6] CONTROL DEL DRONE EN GAZEBO MEDIANTE GEOMETRY_MSGS/TWIST http://mirror.umd.edu/roswiki/hector_quadrotor_gazebo.html
- [7] APLICACIONES DE LA RECONSTRUCCIÓN 3D: ODOMETRÍA VISUAL E INTEGRACIÓN CON LA REALIDAD VIRTUAL (2017) *TFG de la Universidad Politécnica de Madrid* http://oa.upm.es/47889/1/TFG_FRANCISCO_NAVARRO_MERINO.pdf
- [8] ESTUDIO DE LA ESTIMACIÓN DE LA POSICIÓN MEDIANTE LOS ALGORITMOS DE MAPEADO Y LOCALIZACIÓN SIMULTÁNEOS FUSIONADOS CON INFORMACIÓN VISUAL

E INERCIAL DE QUALCOMM SNAPDRAGON FLIGHT (2018) *TFG de la Universidad Politécnica de Madrid*
http://oa.upm.es/52821/1/TFG_JAIME_NIETO_SERRANO.pdf

- [9] TIPOS DE SENSORES DE DRONES PARA EVITACIÓN DE OBSTÁCULOS
https://guiadrones.com/base-de-conocimiento/mejores-drones-para-evitar-colisiones-y-deteccion-de-obstaculos/
- [10] DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS DE EVITACIÓN DE OBSTÁCULOS PARA UN VEHÍCULO AUTÓNOMO SUBMARINO *Tfg de la Universidad Politécnica De Valencia*
https://riunet.upv.es/bitstream/handle/10251/110127/43220496T_TFM_15369251057161086072878319282348.pdf?sequence=1&isAllowed=y
- [11] Vladimir J Lumelsky y Alexander A Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. En: Algorithmica 2.1-4 (1987), págs. 403-430 (vid. pág. 16).
- [12] MAPPING AREAS USING COMPUTER VISION ALGORITHMS AND DRONES (2019) *Documento de la Universidad de Bridgeport:* https://arxiv.org/pdf/1901.00211.pdf
- [13] STACK DE ROS HECTOR QUADROTOR (2012) *Página principal del stack de ROS hector_quadrotor:* http://wiki.ros.org/hector_quadrotor
- [14] DRONE LOCALIZATION AND MAPPING TECHNIQUES (2019) *Técnicas de localización y mapeado desarrolladas por Laserlab:*
https://www.lasernavigation.it/5-years-of-development-of-localization-and-mapping-techniques-in-laser-lab-part-1/
- [15] 3D MAPPING AND PATH PLANNING (2016) *Tesis de la Universidad Politécnica de Cataluña sobre algoritmos de mapeado 3D:* http://hdl.handle.net/10803/392615
- [16] ALGORITMO FAMSA (2016) *Algoritmo de correspondencia de puntos 3D desarrollado por la Universidad Politécnica de Barcelona:*
https://digital.csic.es/bitstream/10261/55825/1/FaMSA_Fast_MultiScan_Alignment.pdf
- [17] MAPEADO Y LOCALIZACIÓN 3D DE UN DRON
https://icarus.csd.auth.gr/wp-content/uploads/2018/08/4-Localization-and-mapping.pdf
- [18] PLUGIN HECTOR GAZEBO PLUGIN
http://wiki.ros.org/hector_gazebo_plugins#GazeboRosImu