PROYECTO MÓVILES:

INTRODUCCIÓN A LOS DRONES EN ROS

David Mataix Borrell
Victoria Frutos Navarro
Nuria Martínez Berná
Esther Vera Moreno

ÍNDICE

1. Introducción en los drones

La aviación no tripulada comenzó como modelos de pruebas para el desarrollo de estos prototipos a mayor tamaño para tener pilotos a bordo. Aunque los inicios de los drones fueron los torpedos aéreos que derivaron en las bombas guiadas, ambos usados principalmente en las guerras.

Un dron es un vehículo aéreo que vuela sin tripulación y que se pueden manejar mediante un control remoto. Aunque todo son básicamente lo mismo hay tres palabras para referirse a estos vehículos no tripulados: RPAS (Remotely Piloted Aircraft System) haciendo referencia al control remoto de estos vehículos, Dron el cual es el nombre más famoso con respecto a estos vehículos, y, UAV (Unmanned Aerial Vehicle) este nombre hace referencia a la ausencia de piloto en la nave.

Existen una amplia variedad de drones, desde algunos más pequeños hasta más grandes. Se pueden clasificar con diferentes metodologías, dependiendo de la sustentación en el aire o por el número de alas.

Según el tipo de sustentación hay dos tipos:

Drones de ala fija: estos tipo de UAV necesitan una velocidad inicial para el despegue, además de disminuir la velocidad en el aterrizaje, es decir, como los aviones de las aerolíneas (aviones normales). Pero cuando están volando tienen muy buena aerodinámica y puede estar volando por horas siendo muy óptimos para el mapeado de grandes superficies.

Drones de ala rotatoria o multirrotores: son las UAV más famosos y son parecidos a los helicópteros normales, se mantienen en el aire por las hélices y pueden despegar en vertical. Cada hélice cuenta con un motor lo que le da una gran estabilidad a este tipo de drones. Dentro de este tipo se pueden encontrar distintos tipos de drones dependiendo del número de hélices.

Cuadricóptero: se componen de cuatro brazos, cada uno contiene un motor y son los más vendidos.

Hexacópteros: son muy seguros porque se puede aterrizar aunque le falle un motor y son los más utilizados para realizar fotografías o vídeos aéreos.

Octocópteros: cuenta con ocho brazos y ocho motores, esto le ofrece más estabilidad y potencia pero lo hace más pesado, esto hará que no pueda operar correctamente en espacios pequeños.

Coaxiales: en este tipo de drones se observa que hay dos motores por cada brazo, lo que los hace capaces de aguantar más peso que los normales. Son buenos para trabajos profesionales (logistica, repartir paquetes, etc)

Se acaba de explicar los tipos de drones aéreos, pero además de los UAVs existen los UUVs (*Unmannned Underwater Vehicles*). Los UUVs son vehículos no tripulados, como los UAVs, que operan bajo el agua. La tecnología para desarrollar este tipo de drones está poco desarrollada respecto a los drones aéreos, aunque se puede observar distintos tipos de drones submarinos.

La principal clasificación de los UUVs es mediante el tipo de conexión:

ROUV son los vehículos submarinos que son controlados por un operador humano de forma teleoperada. El control de este tipo de dron se suele realizar con un cable, la ventaja de esta conexión es la facilidad para transmitir la energía y los datos ya que la ondas de radio tienen dificultad en el agua, pero el propio peso del cable requiere energía para moverlos. Una de la clasificación de los ROUV es por el tamaño, pueden ser pequeños o de observación (peso máximo de 10 Kg) o mediano (peso de 100 o 200 Kg)

AUV son son los vehículos submarinos que trabajan de manera completamente autónoma. Se eliminó la comunicación por cable lo que le daba autonomía para poder operar mejor, pudiendo utilizarse para investigación de los océanos. Como la ondas de radio no se propagan bien por el agua y no se dispone de cable, se incorporó moduladores acústicos para la comunicación de información.

Dos de las principales aplicaciones de estos drones son la defensa, habiendo un desarrollo armamentístico para una posible guerra, y, oceanografía. En la oceanografía se utiliza para poder descubrir y mapear partes del océano que el ser humano no puede llegar.

Aunque la aplicación que se va a desarrollar en ROS va a ser con vehículos no tripulados aéreos, se puede implementar y desarrollar aplicaciones en drones submarinos con el paquete que tiene ROS. Ambos tipos de vehículos tienen software abierto en ROS para el desarrollo de aplicaciones.

2. Aplicaciones

Como se ha comentado anteriormente, un drone es un pequeño vehículo aéreo no tripulado el cual es capaz de desplazarse rápidamente sobre cualquier terreno y superar cualquier tipo de obstáculo, ofreciendo imágenes o capturando otro tipo de datos. Los drones están equipados con tecnología de todo tipo, como cámaras de infrarrojos, GPS, láser, etc, que permiten un control remoto del dispositivo.

Debido a sus características los drones se han introducido en una gran cantidad de sectores, de los cuales destacan:

- **Logística:** el uso de drones en estos sectores acelera las dinámicas de los almacenes ya que es capaz de desplazarse entre ellos realizando tareas a su vez.

- **Agricultura:** el uso de drones en este sector se basa en la teledetección, es decir, el dron se encarga de recorrer el terreno aportando información del campo gracias a las cámaras. Aunque también son muy utilizados en aplicaciones fitosanitarias, es decir, en la detección de plagas y malas hierbas.

- **Seguridad:** en este sector se emplean drones con videovigilancia los cuales permiten proteger y vigilar grandes espacios gracias al amplio campo de visión que ofrecen, a la elevada velocidad a la que pueden desplazarse y a su versatilidad.

- **Periodismo y fotografía:** en este sector se emplean los drones como herramienta para poder llegar a aquellos lugares donde los profesionales no pueden o es peligroso para su bienestar.

También cabe destacar que los drones son muy empleados en aplicaciones militares y son conocidos como UCAV (Vehículos aéreos de combate no tripulados). El uso de este tipo de drones permite operar de forma remota en las zonas más peligrosas sin ningún tipo de limitación y sin el riesgo de perder vidas humanas.

A continuación se van a explicar detalladamente varias posibles aplicaciones para los drones.

Los incendios forestales provocan la desaparición de miles de hectáreas de bosques y reservas naturales. Esta situación pone en peligro vidas humanas, produce pérdidas materiales y provoca daños ambientales muy graves. Para hacer frente a este problema ha sido necesario adoptar nuevas técnicas y medidas que permitan elevar el nivel de eficiencia en los trabajos de prevención, gestión y control de incendios, para ello se han integrado los UAV en tareas de prevención y control de incendios forestales, más específicamente, se han empleado drones para la búsqueda de puntos calientes tras la extinción de un incendio.

El empleo de vehículos aéreos no tripulados como herramienta en cada una de las etapas de un incendio (antes, durante y después) es una manera de elevar la eficiencia, ya que éstos permiten disponer de información fiable en tiempo real y a bajo coste.

Una vez que se ha extinguido el incendio, el personal de tierra debe de recorrer el terreno en busca de los posibles puntos calientes restantes, esto es una tarea lenta y poco eficiente por lo que al usar un dron se aligera el trabajo haciéndolo más eficiente. El dron monitoriza el terreno y localiza los puntos calientes restantes de manera que los servicios forestales pueden acceder a ese punto y apagarlo antes de que se reaviva la llama.

Otra de las posibles aplicaciones en las que se puede implementar en un UAV es un sistema colaborativo entre el mismo, y un UGV (vehículo no tripulado terrestre) o una plataforma terrestre. Ambos estarían unidos por un cable, pero cada uno puede moverse independientemente por lo que el problema se basa en que se muevan de manera coordinada.

El movimiento coordinado entre ambos dispositivos y la unión de esto mejora la percepción del espacio para diversas actividades. Se introduce un robot primario (UGV) siendo el UAV un asistente con un punto de vista distinto. Para saber la localización del UAV con respecto al otro sistema se puede utilizar la posición del cable, aunque este es un sistema pasivo el cual no tiene control sobre su longitud u orientación. Uno de los problemas que se obtiene el de la movilidad coordinada del sistema es el tiempo de reacción del dron con respecto a fuerza externas que se ejerzan sobre él.

Existen desarrollos, principalmente comerciales, en el desarrollo de este tipos de sistemas robóticos. El dron está conectado con el cable a una plataforma terrestre fija que lo alimenta eléctricamente. Esto hace que las horas de vuelo aumenten debido a que están conectados a la luz, pero limita el espacio de vuelo al estar conectado a ese cable.

3. Instalación en ROS

Se ha nombrado anteriormente que ROS cuenta con paquetes para implementar vehículos no tripulados aéreos y submarinos. Aunque se desarrollará el control de UAV, se va a introducir la instalación tanto del paquete de UAV, como el de UUV.

3.1 Instalación UUV

La instalación de `uvv_simulator` es un paquete que contiene los complementos necesarios para simular un vehículo no tripulado submarino. La instalación es muy simple, primero hay que introducir estos comando en la terminal:

```
$sudo apt-get update
$sudo apt-get install ros-kinetic-uuv-simulator
```

Una vez ejecutado estos comandos, la biblioteca ya está instalada. Se copia el paquete en el espacio de trabajo con los siguiente comandos:

```
$mkdir ~/uuv_simulator
$cd ~/uuv_simulator
$cd ~/uuv_simulator/src
$git clone https://github.com/uuvsimulator/uuv_simulator.git
```

Cuando se ha hecho todas las instalaciones anteriores se introduce estos comandos para actualizar ros y las terminales:

```
$source devel/setup.bash
$source /opt/ros/kinetic/setup.bash
```

Cuando se ha hecho todo lo anterior, ya se puede trabajar con el simulador de UUV, si se ejecuta el siguiente comando se puede observar un UUV en un mundo de Gazebo:

```
$roslaunch uuv_gazebo start_mb_smc_demo.launch
```

Al introducir el comando se puede observar el dron en el mundo de Gazebo y la lectura de los diferentes sensores en RViz, se muestra en las siguientes imágenes.

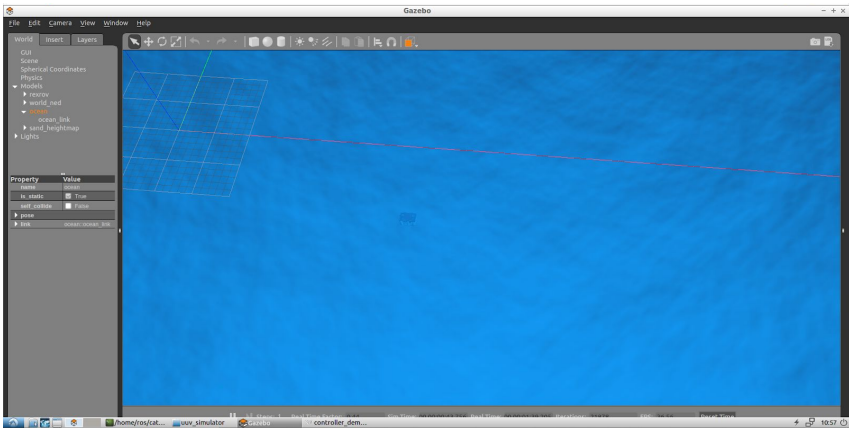


Figura 1: UUV en la simulación de Gazebo

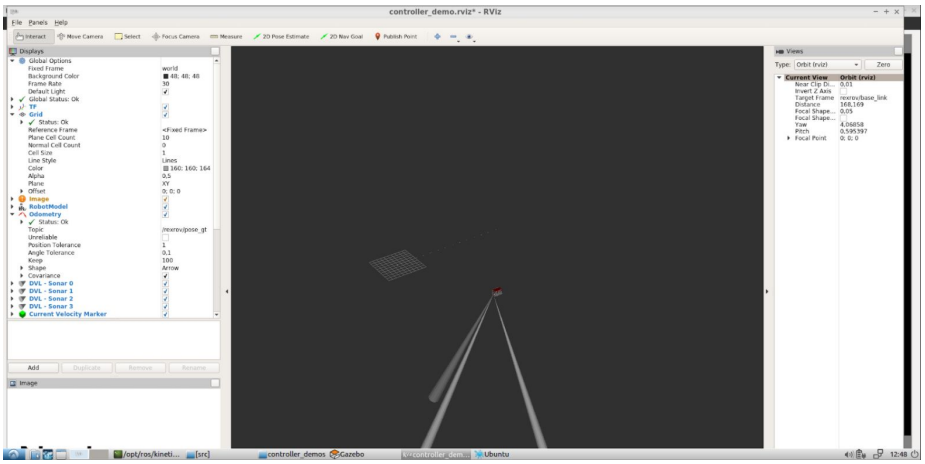


Figura 2: Lectura de los sensorse del UUV en RViz

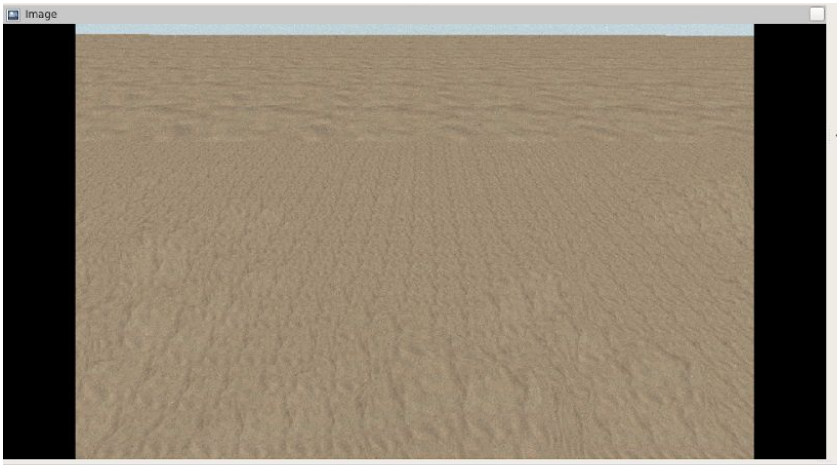


Figura 3: Imagen de la cámara del UUV en RViz

3.1 Instalación UAV

Como se ha comentado antes, hay una amplia variación de distintos drones. El que se va a utilizar en el desarrollo de la aplicación va a ser del tipo Quadcopter. Existe un paquete en ros para este tipo de UAV y se puede instalar de dos maneras diferentes.

La primera manera es poniendo en el terminal los siguientes comandos:

```
$sudo apt-get update
$sudo apt-get install ros-kinetic-hector-quadrotor-demo
```

Si al poner estos comando no funciona porque no encuentra el paquete para ser instalado se puede instalar del directorio de github directamente. Para comenzar con la instalación primero se crea un espacio de trabajo en Ros:

```
mkdir ~/hector_quadrotor_tutorial
```

```
cd ~/hector_quadrotor_tutorial
wstool init src
https://raw.github.com/tu-darmstadt-ros-pkg/hector_quadrotor/kinetic
-devel/tutorials.rosinstall
```

Una vez instalado hay que instalar distintos paquetes para que se pueda ejecutar la simulación del dron. Hay que instalar el paquete *geographic_msgs* que contiene mensajes de puntos en los mapas o segmentos. También hay que instalar el paquete de *ros control* y *ros gazebo control* para poder ejecutar los controladores de los motores, las transmisiones y el hardware del dron.

```
sudo apt-get install ros-kinetic-geographic
sudo apt-get install ros-kinetic-ros-control
sudo apt-get install ros-kinetic-gazebo-ros-control
```

Una vez instalado todo lo necesario, primero se ejecuta el *catkin_make* y después ya se puede ejecutar el dron en gazebo y comprobar su funcionamiento con el siguiente comando:

```
roslaunch hector_quadrotor_demo outdoor_flight_gazebo.launch
```

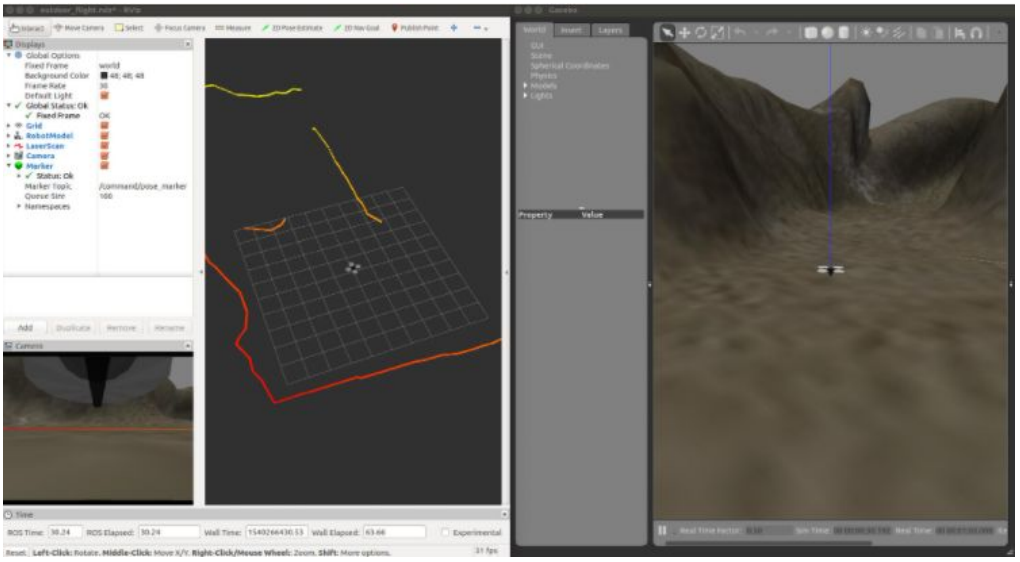


Figura 4: Lectura de los sensores del UAV en RViz y simulación en Gazebo

Cuando se ejecuta se abre RViz con el robot y la información de los obstáculos que detecta, además, en la esquina inferior izquierda se puede observar lo que la cámara está viendo. En gazebo se puede observar el dron en el mundo, que es una montaña.

Para poder controlar el dron y hacer que vuele se puede hacer de dos maneras: mediante teclado o por una mando de XBox. La manera que se va utilizar es por teclado, para ello primero hay que instalar el paquete de *teleop_twist_keyboard* con el siguiente comando:

```
sudo apt-get install ros-kinetic-teleop-twist-keyboard
```

Una vez se ha instalado el paquete ya se puede ejecutar el archivo launch para ejecutar la teleoperación con el siguiente comando:

```
roslaunch teleop_twist_keyboard teleop_twist_keyboard.py
```

Pero al ejecutar este comando se observará que al intentar mover el dron, este no se moverá. Esto se debe a que no se han activado los motores del dron, por lo que hay que ejecutar el siguiente comando antes del anterior:

```
rosservice call /enable_motors "enable: true"
```

Una vez hecho todo esto, se puede mover el dron alrededor del mundo y observar como varía la cámara en RViz. Para volar el dron se pulsa la tecla "t" y para seguir manteniendo la altura se debe pulsar ese botón sino el dron irá descendiendo paulatinamente. Si se quiere aterrizar basta con soltar la tecla o pulsar la tecla "b".

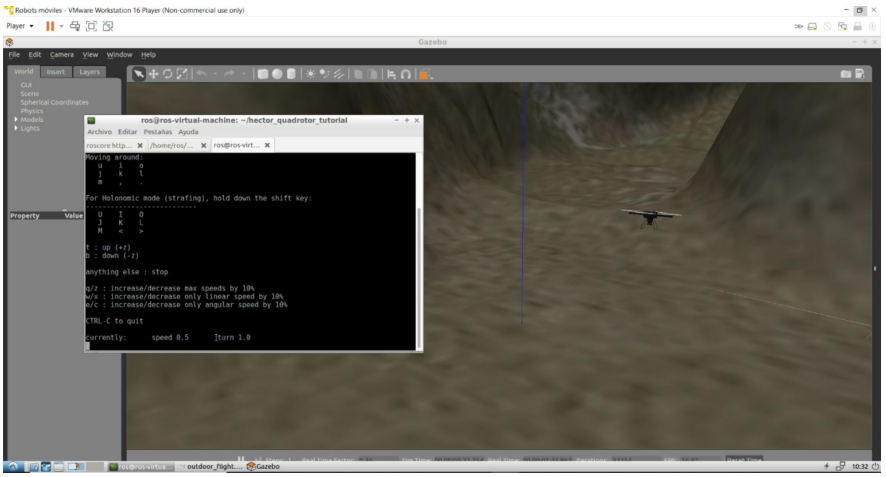


Figura 5: Teleoperación del dron en Gazebo

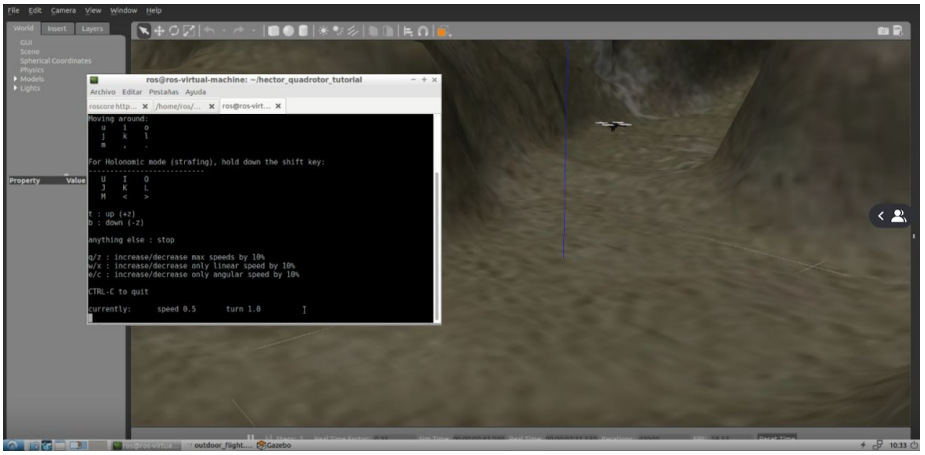


Figura 6: Teleoperación del dron en Gazebo

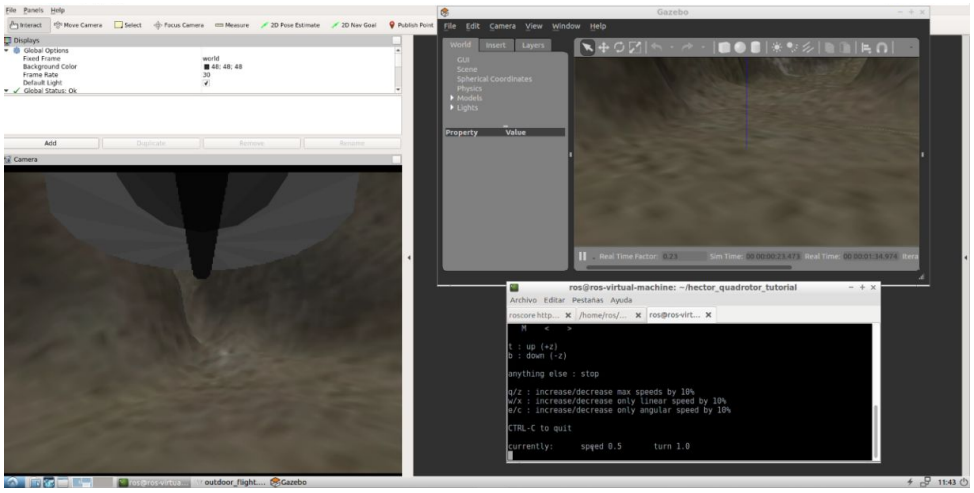


Figura 7: Cámara del dron en RViz

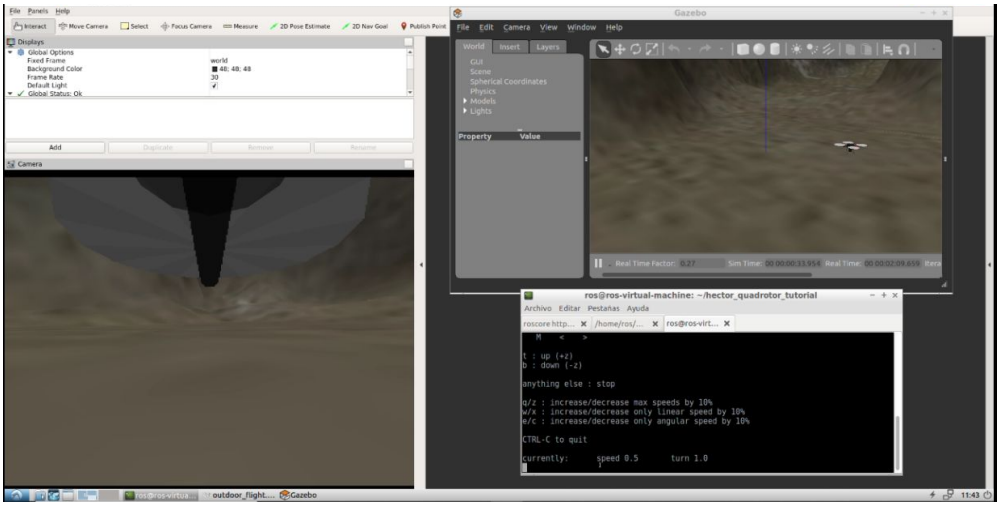


Figura 8: Cámara del dron en RViz y simulación en Gazebo

Se ha grabado una prueba del movimiento del dron en el entorno online de ROS ya que su funcionamiento es mejor y el robot no desciende aunque ya no se esté pulsando la "t", como ocurre en la simulación de la máquina virtual. En este caso tampoco hace falta encender los motores. Esto se puede ver en el enlace.

4. Programación del dron en ROS

4.1 Topics

Una vez que se ha lanzado el dron y está en funcionamiento en el mundo de simulación se miraron los topics en los que el robot es capaz de publicar y recibir información. Para ello se empleó:

```
rostopic list
```

En las siguientes imágenes se pueden observar todos los topics que hay disponible para el Quadcopter.

```
rros@ros-virtual-machine:~/hector_quadrotor_tutorial$ rostopic li
/action/landing/cancel
/action/landing/feedback
/action/landing/goal
/action/landing/result
/action/landing/status
/action/pose/cancel
/action/pose/feedback
/action/pose/goal
/action/pose/result
/action/pose/status
/action/takeoff/cancel
/action/takeoff/feedback
/action/takeoff/goal
/action/takeoff/result
/action/takeoff/status
/aerodynamics/wrench
/altimeter
/altimeter/parameter_descriptions
/altimeter/parameter_updates
/clicked_point
/clock
/cmd_vel
/command/attitude
/command/motor
/command/pose
/command/pose_marker
/command/pose_marker_array
/command/thrust
/command/twist
/command/twist_limit
/command/wrench
/command/yawrate
/controller/attitude/pitch/parameter_descriptions
/controller/attitude/pitch/parameter_updates
/controller/attitude/pitch/state
/controller/attitude/roll/parameter_descriptions
/controller/attitude/roll/parameter_updates
/controller/attitude/roll/state
/controller/attitude/yawrate/parameter_descriptions
```

Figura 9: topics del dron

```
/controller/attitude/roll/parameter_descriptions
/controller/attitude/roll/parameter_updates
/controller/attitude/roll/state
/controller/attitude/yawrate/parameter_descriptions
/controller/attitude/yawrate/parameter_updates
/controller/attitude/yawrate/state
/controller/position/x/parameter_descriptions
/controller/position/x/parameter_updates
/controller/position/y/parameter_descriptions
/controller/position/y/parameter_updates
/controller/position/yaw/parameter_descriptions
/controller/position/yaw/parameter_updates
/controller/position/z/parameter_descriptions
/controller/position/z/parameter_updates
/controller/velocity/x/parameter_descriptions
/controller/velocity/x/parameter_updates
/controller/velocity/x/state
/controller/velocity/y/parameter_descriptions
/controller/velocity/y/parameter_updates
/controller/velocity/y/state
/controller/velocity/z/parameter_descriptions
/controller/velocity/z/parameter_updates
/controller/velocity/z/state
/diagnostics
/estop
/fix
/fix/position/parameter_descriptions
/fix/position/parameter_updates
/fix/status/parameter_descriptions
/fix/status/parameter_updates
/fix/velocity/parameter_descriptions
/fix/velocity/parameter_updates
/fix_velocity
/front_cam/camera/camera_info
/front_cam/camera/image
/front_cam/camera/image/compressed
/front_cam/camera/image/compressed/parameter_descriptions
/front_cam/camera/image/compressed/parameter_updates
/front_cam/camera/image/compressedDepth
/front_cam/camera/image/compressedDepth/parameter_descriptions
```

Figura 10: topics del dron

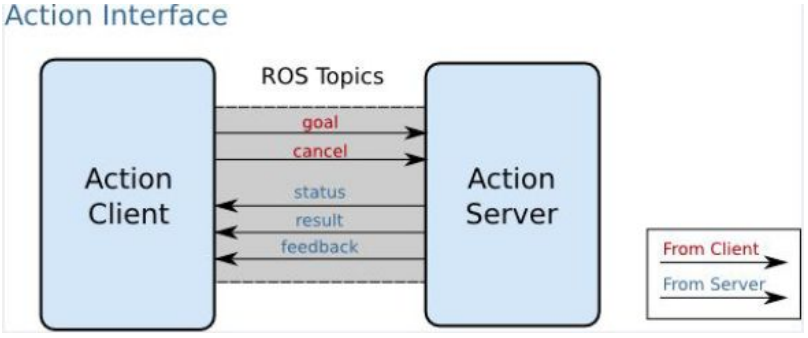
```
/front_cam/camera/image/compressedDepth/parameter_updates
/front_cam/camera/image/theora
/front_cam/camera/image/theora/parameter_descriptions
/front_cam/camera/image/theora/parameter_updates
/front_cam/parameter_descriptions
/front_cam/parameter_updates
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/gazebo_gui/parameter_descriptions
/gazebo_gui/parameter_updates
/ground_truth/state
/ground_truth_to_tf/euler
/ground_truth_to_tf/pose
/initialpose
/joint_states
/joy
/joy/set_feedback
/magnetic
/magnetic/parameter_descriptions
/magnetic/parameter_updates
/move_base_simple/goal
/pressure_height
/raw_imu
/raw_imu/accel/parameter_descriptions
/raw_imu/accel/parameter_updates
/raw_imu/bias
/raw_imu/rate/parameter_descriptions
/raw_imu/rate/parameter_updates
/raw_imu/yaw/parameter_descriptions
/raw_imu/yaw/parameter_updates
/rosout
/rosout_agg
/scan
/sonar_height
/sonar_height/parameter_descriptions
/sonar_height/parameter_updates
/tf
/tf_static
/wind
ros@ros-virtual-machine:~/hector_quadrotor_tutorial$
```

Figura 11: topics del dron

Pero no se van a comentar todos sino que tal solo se comentarán aquellos que tienen una mayor relevancia para la aplicación elegida, la cual se explicará más adelante.

- `/action:` este topic se emplea para controlar el estado del dron, es decir, el aterrizaje, el despegue y la posición actual. Cada uno de estos parámetros contiene diferentes topics:
    - `cancel`
    - `feedback`
    - `goal`
    - `status`
    - `results`

  Los cuales se puede observar el funcionamiento en el siguiente esquema:



- `/cmd_vel`: este topic se encarga de publicar la velocidad lineal y angular.

```
ros@ros-virtual-machine:~/hector_quadrotor_tutorial$ rostopic info /cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_twist_keyboard (http://ros-virtual-machine:37043/)

Subscribers:
 * /gazebo (http://ros-virtual-machine:41383/)
```

Figura 12: topic /cmd_vel

- `/controller`: este topic se encarga de controlar el robot independientemente del mundo que lo rodea.
  - `/controller/attitude`: detecta la rotación tridimensional del dron.
  - `/controller/position`: detecta la posición en la que se encuentra el dron.
  - `/controller/velocity`: detecta la velocidad a la que se mueve el dron.

- `/front_cam`: este topic es el encargado de establecer los parámetros de la cámara RGB así como se mostrar las imágenes que capta.

```
ros@ros-virtual-machine:~/hector_quadrotor_tutorial$ rostopic info /front_cam/camera/image
Type: sensor_msgs/Image

Publishers:
 * /gazebo (http://ros-virtual-machine:41383/)

Subscribers: None
```

Figura 13: topic /front_cam

- `/scan`: este topic se encarga de mostrar las medidas tomadas por el láser del dron. El láser se emplea para comprobar si hay objetos cerca y evitar colisiones.

```
ros@ros-virtual-machine:~/hector_quadrotor_tutorial$ rostopic info /scan
Type: sensor_msgs/LaserScan

Publishers:
 * /gazebo (http://ros-virtual-machine:41383/)

Subscribers:
 * /rviz (http://ros-virtual-machine:33083/)
```

Figura 14: topic /scan

- `/sonar_height`: este topic contiene la información relativa al sensor sonar que tiene el dron.

```
ros@ros-virtual-machine:~/hector_quadrotor_tutorial$ rostopic info /sonar_height
Type: sensor_msgs/Range

Publishers:
 * /gazebo (http://ros-virtual-machine:41383/)

Subscribers: None
```

Figura 15: topic /sonar_height

- `/ground_truth/state`: proporciona la odometría del dron.

```
david@david-Lenovo-ideapad-700-15ISK:~/hector_quadrator/src/aplicacion/src$ rost
opic info /ground_truth/state
Type: nav_msgs/Odometry

Publishers:
 * /gazebo (http://david-Lenovo-ideapad-700-15ISK:34377/)

Subscribers:
 * /ground_truth_to_tf (http://david-Lenovo-ideapad-700-15ISK:45117/)
```

Figura 16: topic /ground_truth

4.2 Sensores

El robot que se ha empleado para la simulación, el `hector_quadrotor`, contiene varios tipos de sensores diferentes que se emplean a la hora de moverse por el mundo simulado.

- **Sonar:** son detectores de proximidad que detectan objetos a distancias que van desde pocos centímetros hasta varios metros. El sensor emite un sonido y mide el tiempo que la señal tarda en regresar. El robot cuenta con el *maxsonar ez4* que es capaz de detectar objetos en un rango de 6 a 254 pulgadas, tiene una velocidad de lectura de 20 Hz y posee varias opciones de salida: ancho de pulso, voltaje analógico y serie RS232.
- **Láser:** cuenta con un sensor LiDAR, el cual es un dispositivo que permite determinar la distancia desde el robot a un objeto o superficie utilizando un haz láser pulsado. La distancia al objeto se determina midiendo el tiempo de retraso entre la emisión del pulso y su detección a través de la señal reflejada.
- **Cámara RGB:** la cámara capta las imágenes del mundo simulado, tiene un sensor que mide la capacidad de luz dentro del espectro visible, es decir, el espectro que el ojo humano es capaz de ver. Con este tipo de cámaras solamente se puede captar e interpretar los colores tal y como nosotros los vemos.

5. Experimentos en simulación

En este apartado se va a describir distintas pruebas con el paquete de dron aéreo `hector_quadrotor` que se han podido realizar para obtener una idea más completa de las funcionalidades que ofrece y las aplicaciones a hacia las que se puede orientar.

5.1 Topic /takeoff y /land

Como se ha visto anteriormente, el paquete ofrece una cantidad bastante grande de topics. Sin embargo, para poder probar más fácilmente la función de despegue y aterrizaje del dron, se decidió utilizar un código adicional. Utilizando los archivos del proyecto [5] se ha podido interactuar con esta funcionalidad. Este se adjunta en la tarea en la carpeta topics */takeoff* y */land*.

El hecho de tener estas funciones es bastante útil a la hora de la navegación pudiendo iniciar el despegue hacia una altura concreta y aterrizar solo con una función de forma automática. Además al moverlo en la simulación de la máquina virtual se tiene que estar pulsando continuamente la tecla "t" para que el dron no se caiga. En el entorno virtual se queda suspendido en el aire solo a partir de que se pulsa "k" (parada en seco) o algún otro movimiento. Utilizando el topic */takeoff* podemos definir que suba durante un tiempo y ya no baje ni siga subiendo publicando como:

```
rostopic pub /takeoff std_msgs/Empty "{}"
```

De la misma manera, cuando se necesite aterrizar el dron solo se deberá escribir:

```
rostopic pub /land std_msgs/Empty "{}"
```

Sin embargo, para usarse en la máquina virtual se deben hacer ciertas modificaciones para que el robot suba y baje continuamente, por lo que el programa proporcionado solo sirve en la versión online. Si se prueba en la máquina virtual, el dron subirá y al poco volverá a bajar.

El funcionamiento de estos topics se puede ver en el vídeo del siguiente enlace y se pueden notar las diferencias con el anterior vídeo sobre la demostración del *teleop*.

Para su utilización en la máquina virtual solo hace falta ejecutar el código como:

```
python takeoff_land_code.py
```

aunque también se puede añadir al archivo *.launch* del mundo para que los *topics* se inicien junto al *roslaunch*:

```
<include file="$(find takeoff_land)/launch/takeoff_land.launch"/>
```

Si se usa en el ROS online con el proyecto proporcionado [5], con el lanzamiento del roslaunch es suficiente aunque habrá que hacer ciertas modificaciones (catkin_make, etc) para que funcione el entorno debido a la última actualización.

5.2 Mapeado, localización y navegación

Para llevar a cabo el mapeado de un entorno conocido, se probaron diferentes métodos de mapeado 3d, pero en ninguno, se consiguió un buen resultado, por lo que se optó por realizar el mapa en 2d, siguiendo los mismos pasos que se realizan cuando se lleva a cabo el mapeado con el Turtlebot.

Para realizar esta tarea se siguieron los tutoriales de las fuentes bibliográficas[6][7][8].

Primero se copio del paquete de navegación del Turtlebot, el archivo gmapping.launch.xml, la lectura del láser se obtiene del topic /scan y el *base_frame* corresponde con *base_footprint* y el odom_frame a *world*:

```
<arg name="scan_topic"  default="/scan" />
<arg name="base_frame"  default="base_footprint"/>
<arg name="odom_frame"  default="world"/>
```

La estructura de los frames se puede obtener tras ejecutar en el terminal:

```
rosrun tf view_frames
```

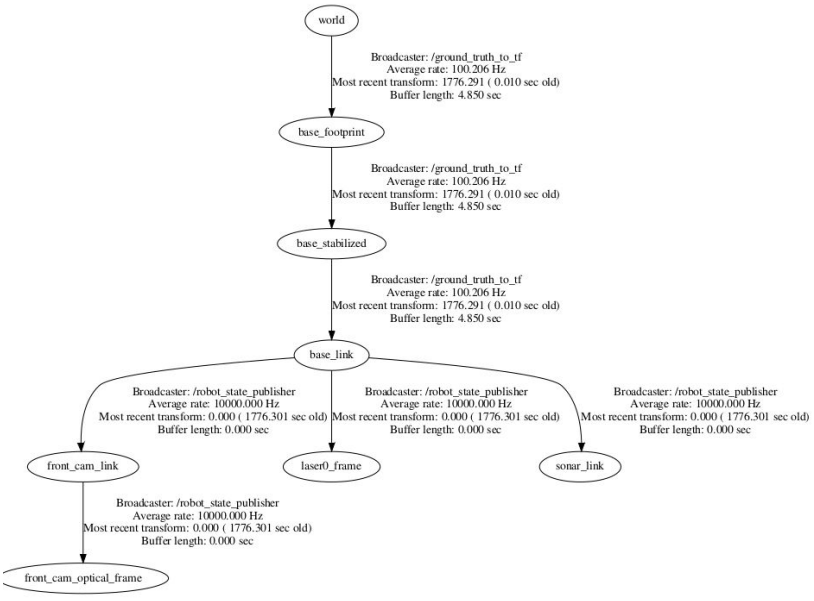


Figura 17: Frames `hector_quadrotor`.

Para poder realizar el mapa del entorno hay que abrir el mundo creado en gazebo, rviz, y ejecutar el archivo gmapping.launch que en este caso se ha llamado quadrotor_mapping.launch:

```
roslaunch hector_quadrotor_demo mundo.launch
rosrun rviz rviz
roslaunch quadrotor_navigation quadrotor_mapping.launch
```

Una vez configurado correctamente rviz:

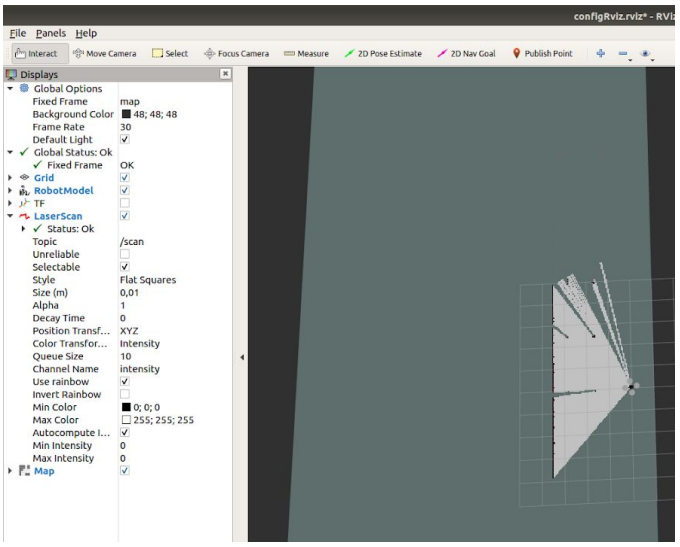


Figura 18: Configuración rviz

Se puede comenzar a realizar el mapa moviendo el dron mediante línea de comandos, antes de poder mover el dron hay que cargar los motores:

```
rosservice call enable_motors "enable: true"
roslaunch teleop_twist_keyboard teleop_twist_keyboard.py
```

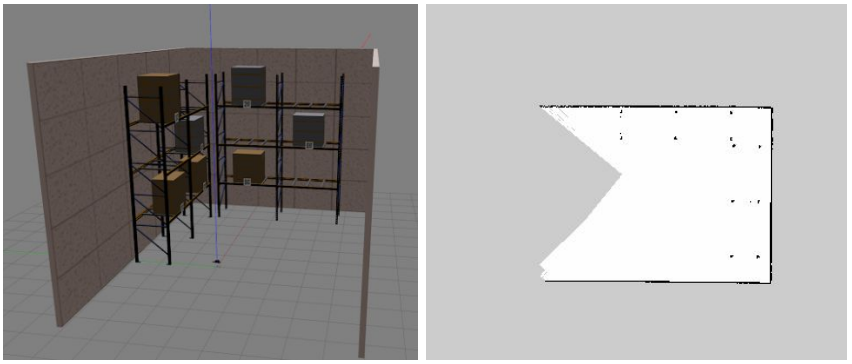


Figura 19: Mapa 2d del entorno simulado

Para llevar a cabo la localización de `hector_quadrotor`, hay que copiar del paquete de navegación del Turtlebot el archivo amcl_demo.launch de donde se obtendrá la forma de cargar el mapa y también hay que copiar el archivo amcl.launch.xml que será el que permita la localización del dron. Hay que introducir los frames de `hector_quadrotor` y el topic */scan.*

```
<arg name="map_file" default="$(findquadrotor_navigation)/maps/escena.yaml"/>
<node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />
<arg name="use_map_topic"   default="false"/>
<arg name="scan_topic"      default="/scan"/>
<arg name="initial_pose_x"  default="0.0"/>
<arg name="initial_pose_y"  default="0.0"/>
```

```
<arg name="initial_pose_a"  default="0.0"/>
<arg name="odom_frame_id"   default="world"/>
<arg name="base_frame_id"   default="base_footprint"/>
<arg name="global_frame_id" default="map"/>
```

Para probar la localización hay que abrir el mundo en gazebo y el mapa en rviz, ejecutar el archivo creado, que en este caso se ha llamado quadrator_localization.launch:

```
roslaunch hector_quadrotor_demo mundo.launch
roslaunch quadrotor_navigation quadrator_localization.launch
rosrun rviz rviz
```

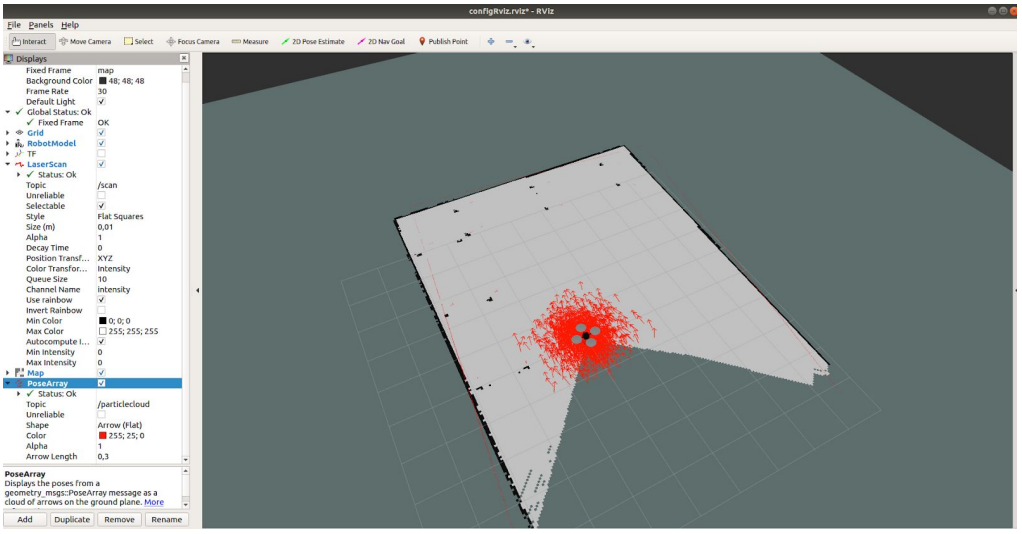


Figura 20: Localización de hector_quadrotor.

Para la navegación del dron, hay que copiar del paquete de navegación del Turtlebot, el archivo move_base.launch.xml y algunos de los archivos de configuración que emplea, para posteriormente ser usados por el dron. Hay que cambiar el topic de odometría que para el drone es */groud_truth/state*, el topic del sensor láser es */scan* y también hay que cambiar el *odom_frame_id* por *world*. Hay que añadir que abra el archivo de localización previamente creado llamado quadrator_localization.launch.

```
<include file="$(find quadrotor_navigation)/launch/quadrator_localization.launch"/>

<arg name="odom_frame_id"   default="world"/>
<arg name="base_frame_id"   default="base_footprint"/>
<arg name="global_frame_id" default="map"/>
<arg name="odom_topic" default="/ground_truth/state" />
<arg name="laser_topic" default="/scan" />
<arg name="custom_param_file" default="$(find quadrotor_navigation)/param/dummy.yaml"/>

<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
  <rosparam file="$(find quadrotor_navigation)/param/costmap_common_params.yaml" command="load"
ns="global_costmap" />
  <rosparam file="$(find quadrotor_navigation)/param/costmap_common_params.yaml" command="load"
ns="local_costmap" />
  <rosparam file="$(find quadrotor_navigation)/param/local_costmap_params.yaml" command="load" />
  <rosparam file="$(find quadrotor_navigation)/param/global_costmap_params.yaml" command="load" />
  <rosparam file="$(find quadrotor_navigation)/param/dwa_local_planner_params.yaml" command="load" />
  <rosparam file="$(find quadrotor_navigation)/param/move_base_params.yaml" command="load" />
  <rosparam file="$(find quadrotor_navigation)/param/global_planner_params.yaml" command="load" />
```

```
  <rosparam file="$(find quadrotor_navigation)/param/navfn_global_planner_params.yaml" command="load" />
  <!-- external params file that could be loaded into the move_base namespace -->
  <rosparam file="$(arg custom_param_file)" command="load" />

  <!-- reset frame_id parameters using user input data -->
  <param name="global_costmap/global_frame" value="$(arg global_frame_id)"/>
  <param name="global_costmap/robot_base_frame" value="$(arg base_frame_id)"/>
  <param name="local_costmap/global_frame" value="$(arg odom_frame_id)"/>
  <param name="local_costmap/robot_base_frame" value="$(arg base_frame_id)"/>
  <param name="DWAPlannerROS/global_frame_id" value="$(arg odom_frame_id)"/>

  <remap from="cmd_vel" to="/cmd_vel"/>
  <remap from="odom" to="$(arg odom_topic)"/>
  <remap from="scan" to="$(arg laser_topic)"/>
</node>
```

Para los diferentes archivos de configuración que emplea, hay que cambiar, en los que sea necesario, el topic del sensor para que sea /scan, y cambiar también el *global_frame* por *map*.

Para proceder a la navegación hay que cargar el mundo en gazebo, el mapa en rviz, lanzar el archivo de navegación llamado quadrotor_move_base.launch, publicar en el topic /takeoff para elevar el dron a una cierta altura y indicando la posición destino con la flecha 2d Nav Goal a la que se tiene que desplazar el dron, igual que se hacía con el Turtlebot.

```
roslaunch hector_quadrotor_demo mundo.launch
rosrun rviz rviz
roslaunch quadrotor_navigation quadrotor_move_base.launch
```

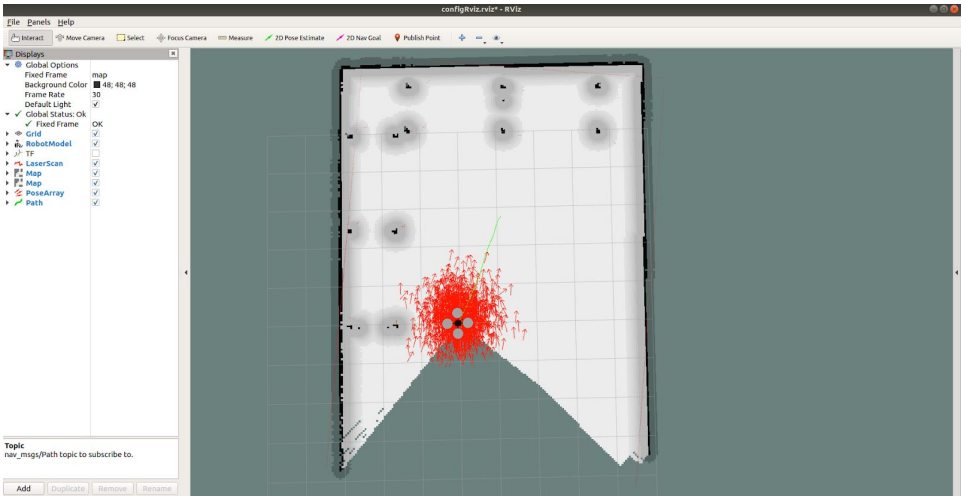


Figura 21: Navegación hector_quadrotor.

En el tutorial seguido si que consigue navegar por el mapa creado con el `hector_quadrotor`, como se muestra en este enlace, pero en nuestro caso no lo hemos conseguido hacer funcionar.

5.3 Control de varios drones

En este proyecto también se ha querido probar el manejo de varios drones al mismo tiempo dentro del mismo entorno. En este caso se ha obtenido el código necesario de las referencias [9] y [10].

Primero de todo se lanzará el mundo de gazebo y se cargarán los motores de cada dron, en este caso se ha probado para 6. A continuación al ejecutarse el código, los robots empiezan a moverse cada uno a una serie de posiciones y por la terminal se van mostrando sus posiciones actuales y las deseadas. Para distinguir cada quadrotor, dentro del *.launch*, están clasificados con el nombre */quadN*, donde *N* es un número, y al hacer `rostopic list` cada uno tiene sus *topics* asociados con su nombre delante:

```
/quad1/controller/velocity/y/parameter_updates
/quad1/controller/velocity/y/state
/quad1/controller/velocity/z/parameter_descriptions
/quad1/controller/velocity/z/parameter_updates
/quad1/controller/velocity/z/state
/quad1/estop
/quad1/fix
/quad1/fix/position/parameter_descriptions
/quad1/fix/position/parameter_updates
/quad1/fix/status/parameter_descriptions
/quad1/fix/status/parameter_updates
/quad1/fix/velocity/parameter_descriptions
/quad1/fix/velocity/parameter_updates
/quad1/fix_velocity
/quad1/ground_truth/state
/quad1/ground_truth_to_tf/euler
/quad1/ground_truth_to_tf/pose
/quad1/joint_states
/quad1/magnetic
/quad1/magnetic/parameter_descriptions
/quad1/magnetic/parameter_updates
/quad1/pressure_height
/quad1/raw_imu
/quad1/raw_imu/accel/parameter_descriptions
/quad1/raw_imu/accel/parameter_updates
/quad1/raw_imu/bias
/quad1/raw_imu/rate/parameter_descriptions
/quad1/raw_imu/rate/parameter_updates
/quad1/raw_imu/yaw/parameter_descriptions
/quad1/raw_imu/yaw/parameter_updates
/quad2/action/landing/cancel
/quad2/action/landing/feedback
/quad2/action/landing/goal
/quad2/action/landing/result
/quad2/action/landing/status
/quad2/action/pose/cancel
/quad2/action/pose/feedback
/quad2/action/pose/goal
/quad2/action/pose/result
/quad2/action/pose/status
/quad2/action/takeoff/cancel
/quad2/action/takeoff/feedback
/quad2/action/takeoff/goal
/quad2/action/takeoff/result
/quad2/action/takeoff/status
/quad2/aerodynamics/wrench
/quad2/altimeter
/quad2/altimeter/parameter_descriptions
/quad2/altimeter/parameter_updates
/quad2/cmd_vel
/quad2/command/attitude
/quad2/command/motor
/quad2/command/pose
/quad2/command/pose_marker
```

Figura 22: Rostopic list con varios drones

El código lee un archivo .mat donde están definidas en una lista las posiciones X, Y, Z de cada dron. Obteniendo la posición actual del dron con */ground_truth/state* y la deseada del *.mat*, mediante el uso del PID y cambiando las velocidades, los drones se acercan a estos puntos definidos.

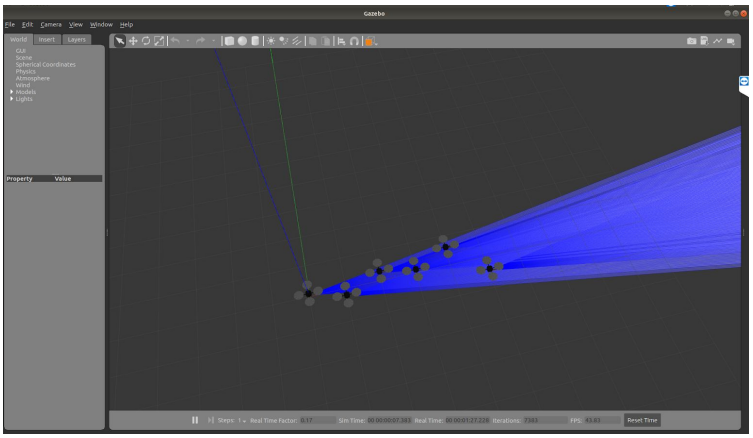


Figura 23: Drones volando simultáneamente 2

Así según las posiciones descritas se podría hacer hasta que los drones formasen figuras entre ellos. Sin embargo, el programa obtenido no es tan avanzado y los drones no están coordinados entre sí, por lo que los choques entre ellos ocurren.

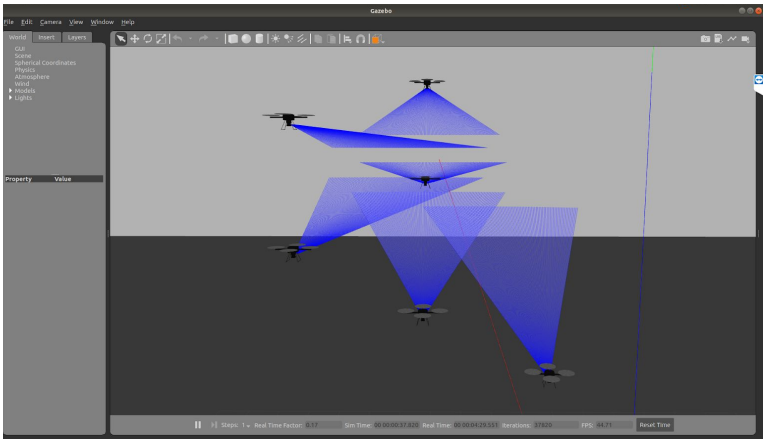


Figura 24: Drones volando simultáneamente 2

La prueba de este código se puede ver en el siguiente enlace. El funcionamiento es correcto dentro de las limitaciones del código obtenido y los drones se mueven a las posiciones deseadas del programa.

5.4 Aplicación propuesta : recuento de cajas

La tarea propuesta trata de la utilización de un dron para hacer un recuento de las cajas de unas estanterías leyendo sus QR. La idea fue obtenida de un proyecto visto en github [11] del cual se obtuvieron las cajas y estanterías.
En los anteriores apartados se ha comentado la dificultad para grabar el mapa y la imposibilidad de poder navegar en el con `hector_quadrotor`. Es por ello que este primer mapa inicial propuesto se ha modificado para hacerlo más sencillo, por lo que el entorno final usado es el siguiente:

Figura 25: Mundo propuesto para la tarea

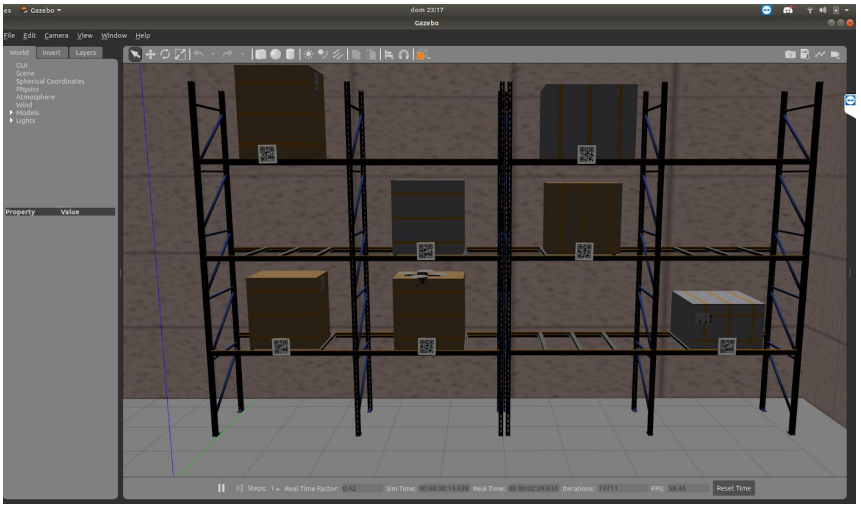


Figura 26: Dron volando en el mundo

Por tanto, para poder llevar al dron a una posición determinada se ha hecho uso del topic */ground_truth/state* como en el código del control de varios drones, aunque el funcionamiento final de movimiento se ha implementado de forma diferente. Este nos proporciona información sobre la posición del dron respecto al mundo. Si se obtienen las posiciones del mundo de gazebo donde se quiere llevar al dron para que compruebe si hay caja o no en un hueco determinado de la estantería, solo se tiene que ver si esa posición deseada coincide con la actual. Para guiar al robot, se irán mandando velocidades lineales y angulares según la diferencia de estas dos posiciones. Primero de todo, el dron llegará a la posición deseada y a continuación girará si es necesario. Cuando entre dentro del rango definido de la posición final, las velocidades serán mucho más pequeñas y no se moverá apenas de este punto. Cuando se quiera comprobar la siguiente estantería, girará en la misma posición y avanzará hacia la siguiente en línea recta. El número de cajas se irá incrementando y finalmente acabará aterrizando al final de la última estantería.
Al usar esta función, cuando el robot se va moviendo en el entorno y en RVIZ se puede ver la cámara y el recorrido que va dejando:

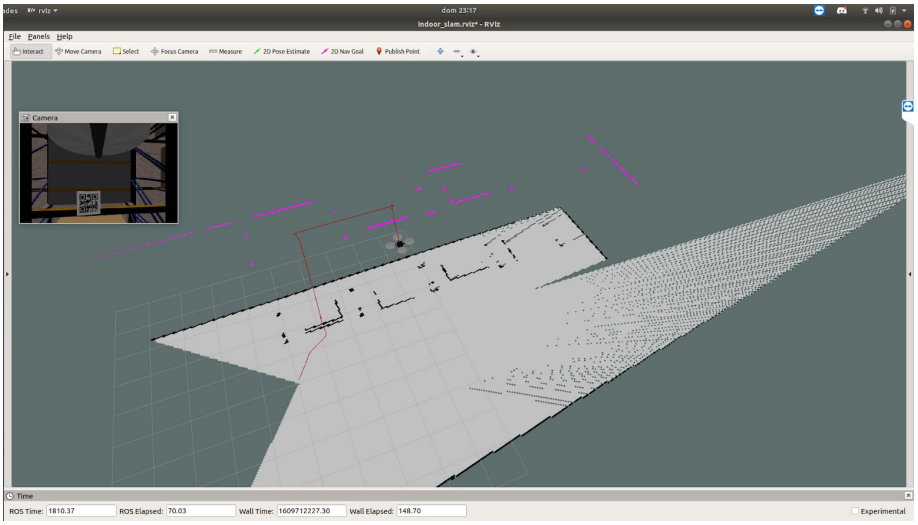


Figura 27: Entorno RVIZ durante la ejecución

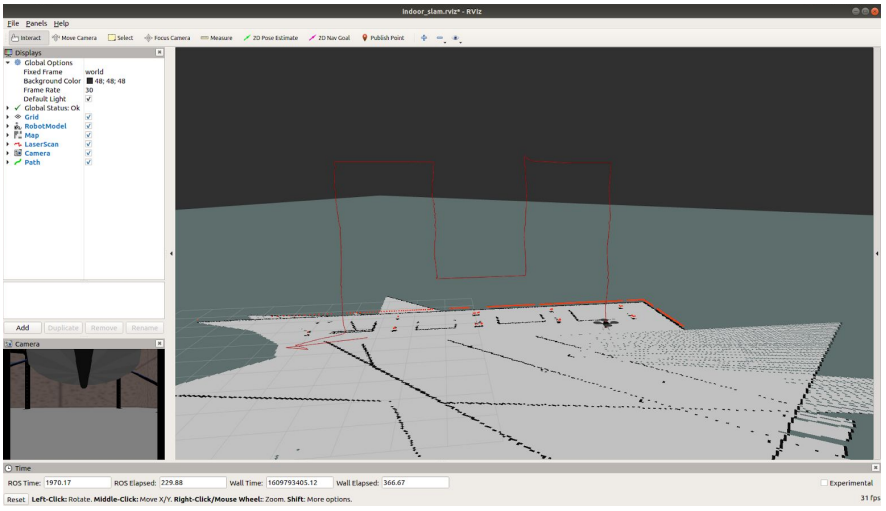


Figura 28: Entorno RVIZ al final de la ejecución

Los códigos desarrollados se llaman *detectar_qr.py* y *contarCajas.py*, ambos se encuentran comentados.

En cuanto al funcionamiento de la tarea, se puede ver en el siguiente enlace que el dron ejecuta los movimientos y el recuento correctamente. Para en cada posición y distingue bien si hay un código QR o no y lo muestra por la terminal. Sin embargo, hay algunos casos, en el que si la aplicación de gazebo o RVIZ va con retardo, el dron puede haberse movido y coger el frame en el que QR no aparece o esté descuadrado y no consiga identificarlo correctamente. Generalmente sí detecta todas las cajas pero puede haber algún caso que cuente de menos y este número final de cajas sea incorrecto.

7. Guía de ejecución

7.1 Control de varios drones

Para la prueba de este código primero se deberá clonar el repositorio en una carpeta:

```
git clone github.com/MaxChernyaev/gazebo_quadcopter
```

Dentro de la carpeta en el terminal se ejecutará:

```
catkin_make
```

Se cambia dentro del archivo *Trajectory_ref_my.py*, dentro de la carpeta scripts de *hector_Astar*, la ruta de ejecución en la línea 103 y el nombre del *.mat* a *my001.mat*.

A continuación, se abre el mundo en gazebo con los 6 drones ejecutando el comando:

```
roslaunch hector_Astar hector_formation_6_quads.launch
```

Una vez abierto el mundo, se activan los motores de los drones. Esto se puede hacer de forma independiente para cada uno ejecutando las órdenes:

```
rosservice call /quad1/enable_motors "enable: true"
rosservice call /quad2/enable_motors "enable: true"
rosservice call /quad3/enable_motors "enable: true"
rosservice call /quad4/enable_motors "enable: true"
rosservice call /quad5/enable_motors "enable: true"
rosservice call /quad6/enable_motors "enable: true"
```

Pero, en el paquete descargado también incluye una opción que introduciendo el número de drones los activaba al mismo tiempo.

```
./motor ON
6
```

Para proceder a la simulación hay que ejecutar el comando:

```
roslaunch hector_Astar hector_formation_6_quads_2.launch
```

7.2 Aplicación propuesta: recuento de cajas

Los modelos necesarios además de los QR creados se encuentran dentro de esta carpeta. Los archivos dentro deben ser adjuntados al directorio `/home/<user>/.gazebo/models` para que funcione el mundo. Aparte de esto, dentro del archivo *escena_entrega.world* se

debe modificar el inicio las rutas donde aparece `home/…`, donde se define el *quadrotor*. Los mundos de Gazebo creados se encuentran en este enlace.

Para poder abrir el mundo de gazebo *escena_entrega.world*, se ha introducido en el fichero *willow_garage.launch*, que esté a su vez es lanzado por los archivos *indoor_slam_gazebo.launch* y por *mundo.launch*. La única diferencia entre estos dos archivos es que uno abre rviz, con la configuración guardada para el *hector_quadrotor* y el otro no.

Para abrir la simulación en gazebo y si se desea, abrir la visualización en rviz hay que ejecutar:

```
roslaunch hector_quadrotor_demo indoor_slam_gazebo.launch
roslaunch hector_quadrotor_demo mundo.launch
```

Después se activan los motores y se lanza cada código en una terminal:

```
rosservice call enable_motors "enable: true"
python detectar_qr.py
python contarCajas.py
```

8. Dificultades encontradas

La primera dificultad del proyecto fue instalar el paquete `hector_quadrator`. Existen varios repositorios y tutoriales pero dependiendo de la máquina virtual usada o la versión online se obtenían diferentes errores. La instalación propuesta es la que al final proporcionaba la ayuda para instalar todos los paquetes necesarios para el funcionamiento del paquete.

En cuanto al mapeado, como se ha comentado, se quiso hacer primero de un entorno 3D. Esto no se consiguió y se optó por grabar el mundo 2D. Sin embargo a la hora de abrirlo, varios paquetes tuvieron que ser modificados para poder ser utilizado como en los mundos de turtlebot, evitando cualquier error al modificarlos. En un principio RVIZ reconocía el RobotModel del dron al abrir el mapa pero finalmente se solventaron estos problemas y se consiguió abrir el mapa 2D.

Por otro lado, la localización y navegación se probó en varias máquinas virtuales y en el entorno online de ROS, y aunque se calculaba la trayectoria, el robot no llegaba a moverse por lo que esta idea de navegación se tuvo que descartar.

Se pensó que el problema podía ser el topic */takeoff* el cual no funcionaba correctamente en la máquina virtual pero en la versión online tampoco se consiguió mover como en el tutorial [6]. Además también este código para el */takeoff* también se modificó para poder usarlo pero al final se descartó su uso en la aplicación final.

Para el control de varios drones distintos archivos .mat y .py fueron probados. El propuesto es el que mejor se podía ver su funcionamiento. Algunos .mat tenían como posición inicial del dron el punto (0,0,0) pero dentro del mapa aparecían en otros puntos por lo que al ejecutarlos estos se iban arrastrando por el suelo en vez de despegar. Se solventó probando y eligiendo un archivo de posiciones correctas puesto que es imposible colocar todos los drones en esa posición.

En cuanto a la detección de las cajas. En un primer momento, el mundo estaba hecho con las estanterías en forma de "L". Esto creaba un problema al mover el dron de una estantería a la otra. Al guiarlo mediante velocidades, había momentos en los que este tenía problemas para girar y acababa dando vueltas de más o chocando. Esto es debido a su vez al no conocimiento del mapa y, al no tener calculados los *costmaps*, los choques no se evitan. De forma que para evitar estos problemas, debido a las limitaciones del movimiento, se modificó el mapa y se ajustó la velocidad publicada. Es por ello que existen posiciones intermedias en el recorrido con giros del robot y se establece un orden de ejecución, intentando no perder la naturalidad en el vuelo.

Finalmente, es posible que en algunas ejecuciones del código de las cajas, alguna caja no sea detectada. Al final se han ajustado lo máximo posible estas posiciones para que ocurra lo menos posible y normalmente funciona correctamente.

En general, ha sido un proyecto lleno de dificultades y de transformaciones dinámicas sobre la marcha para poder adaptarlo a las limitaciones del paquete, el entorno y de las propias máquinas virtuales.

9. Conclusión

Como conclusión, cabe decir que durante la realización de este proyecto se ha aprendido más sobre los tipos de drones y cómo se puede trabajar en simulación con ellos. En concreto se han probado dos paquetes para ROS, el de drones submarinos `uvv_simulator` y el `hector_quadrotor`. Finalmente, el trabajo se ha centrado en el análisis de este último, sobre las ventajas y funcionalidades que ofrece. Las dificultades encontradas se han podido superar de la mejor manera posible y gracias a ello, varios entornos y aplicaciones con el dron aéreo han podido ser probadas, destacando entre ellas: el control de varios drones de forma simultánea y la aplicación propuesta del recuento de cajas.

NOTA: la presentación del proyecto se puede ver en el enlace.

10. Bibliografía

[1] Propuesta de sistema multi-UAV para aplicaciones de cobertura de área
https://www.tdx.cat/bitstream/handle/10803/456309/eesc1de1.pdf?sequence=1&isAllowed=y

[2] Cooperación Cordada entre Robot Terrestre(UGV) y Robot Aéreo (UAV)
http://oa.upm.es/52971/1/TFM_Daniel_Aldalur_Leal_1.pdf

[3] Aplicaciones de drones
https://www.imnovation-hub.com/es/sociedad/aplicaciones-de-los-drones/

[4] Tutorial de instalación de ROS
https://darienmt.com/autonomous-flight/2018/10/20/flying-ros-and-hector.html

[5] Entorno online del dron con topics /takeoff, /land y mapas obtenido del tutorial
http://www.rosject.io/l/c5f313b/

[6] Mapeado con el dron
https://www.youtube.com/watch?v=dND4oCMqmRs&t=614s

[7] Localización del dron
https://www.youtube.com/watch?v=n6RjVbh3Vgc&t=491s

[8] Planeador de trayectorias del dron
https://www.youtube.com/watch?v=JZqVPgu0KIw&t=409s

[9] Control de varios drones
https://www.youtube.com/watch?v=-I1mkmI-s00&t=183s

[10] Github para el control de varios drones
github.com/MaxChernyaev/gazebo_quadcopter

[11] Modelo de las cajas y estanterías
https://github.com/webvenky/qr_scanning_drones