Juan Carlos Escalante Diaz Del Campo

Reporte Técnico Final

“Diseño y modelado de redes resistivas para la planeación de Trayectorias Robóticas libres de colisiones”

Instituto Nacional de Astrofísica Óptica y Electrónica

30 de Mayo del 2017 - 20 Julio 2017

# Objetivos

* Principal: Aplicación de método “redes resistivas” para la generación de trayectorias en un Vehículo aéreo no tripulado.
* Secundario: Programación en Python para trayectorias predeterminadas.
* Secundario: Creación de archivos .launch en sistema operativo: “Ubuntu”, en Robot Operating System.
* Secundario: Implementación de simulación de UAV: Cuadrimotor utilizando Gazebo simulator y TUM\_simulator.

# Metas

Se debe lograr una programación de rutinas en lenguaje de alto nivel en este caso para el dron, definiendo una trayectoria y por ende una solución a una trayectoria libre de colisiones. Además de lograr una comunicación entre ROS y el dron obteniendo así resultados favorables tanto en simulación como en implementación real.

**Especificaciones técnicas**

* Sistema operativo Ubuntu 14.04.05 Trusty
  + Se utilizó este sistema operativo por las dependencias de ROS y el soporte que éste mismo tiene para ese sistema operativo, en gran mayoría, todo el soporte está para esta plataforma.
* ROS Indigo
  + Este sistema operativo Robótico, se utiliza para manejar, crear y utilizar distintas librerías, repositorios o archivos, relacionados con la robótica en sí, en este caso se eligió éste porque es el que soporta la versión 14.04 de Ubuntu y es en el que hay más soporte.
* AR. Drone Autonomy 2.0
  + Ardrone\_autonomy es un driver de ROS para Parrot AR-drone 1.0 y 2.0 cuadrimotor. Este driver está basado en el oficial AR-Drone SDK versión 2.0.1. El paquete fue desarrollado en el laboratorio “Autonomy” de Simon Fraser University por Mani Monajjemi y otros contribuyentes.
* Gazebo Simulator, TUM\_simulator, TUM\_ardrone
  + Gazebo es un ambiente de simulación robótico, no solamente se utiliza para drones, también se utiliza para brazos robóticos, robots diferenciales y bases.
  + TUM\_simulator es una librería o repositorio que permite la simulación del cuadrirotParrot Ardrone 2.0, basado en las especificaciones reales y situaciones en un ambiente real, simulando una dirección IP que es la que el dron ficticio está mandando al adaptador de red de la computadora.
  + TUM\_ardrone es una librería o repositorio que permite visualizar las camaras del drone para una navegación autónoma basada en la cámara.
* Parrot AR.Drone 2.0
  + Dron utilizado para las pruebas físicas que también se utilizó en las pruebas de simulación.
  + Especificaciones físicas:
  + Dimensiones: Con casco: 52,5 x 51,5 cm | Sin casco: 45 x 29 cm
  + Peso: 380 g con la cubierta del casco para exteriores. | 420 g con la cubierta del casco para interiores.
  + Velocidad de marcha (crucero): 5 m/s, 18 km/h. Autonomía de vuelo promedio: 12 minutos.
  + Sistema informático integrado:
  + Microprocesador ARM9 RISC de 32 bits @ 468 MHz
  + Memoria DDR SDRAM de 128 MB @ 200 MHz
  + Sistema operativo con núcleo Linux
  + Módem Wi-Fi b / g
  + Conector USB de alta velocidad
  + Sistemas de guía inercial:
  + Acelerómetro de 3 ejes construido con tecnología MEMS.
  + Giroscopio de 3 ejes, Magnetómetro de 3 ejes y sensor de presión.
  + Motores y energía:
  + 4 motores sin escobillas, funcionando a 3.500 rpm con una potencia de 15 W
  + Batería de ion de litio de 3 celdas, capaz de entregar 1000 mA/hora con un voltaje nominal de 11,1 V, y que le permite una autonomía de vuelo de entre 12 y 15 minutos.
  + Capacidad de descarga: 10C. Tiempo de carga: 90 minutos.
  + Cámara frontal, ventral y altímetro:
  + Cámara frontal con sensor CMOS 93° amplitud, ventral con sensor CMOS de lente diagonal 64° de amplitud, dos ultrasonidos de 40 khz para alcanzar gran estabilidad a grandes alturas.

**Justificación del Proyecto**

El uso de aviones no tripulados o, más técnicamente llamados, de vehículos autónomos no tripulados (UAV) ha aumentado en los últimos años. Mientras que el uso de UAV solía ser sólo militar, hoy podemos encontrar UAV para la investigación, exploración, fotografía, patrullaje e incluso recreación.

Por eso, hoy es más importante que nunca aprender a programar los drones. Aunque pueden ser controlados en tiempo real por el usuario, la implementación de rutas y trayectorias automáticas, junto con sensores y sistemas de control, puede ayudar con el uso de drones en varias tareas como patrullaje o misiones de ayuda y rescate, siendo totalmente autónomo.

El proyecto presentado en este trabajo se realizó por las razones descritas anteriormente. Con el objetivo final de crear varias rutas y trayectorias para crear posteriormente rutas óptimas por medio de redes resistivas.

**Normativa aplicable**

En mayo de 2016, el SCT y la dirección general de la aviación civil publicaron los requisitos para operar un UAV. CO AV-23/10 R2 se aplica a cualquier persona física que opera o tiene la intención de operar una nave aérea no tripulada. Las autoridades de aviación clasificaron a los UAV por su peso y uso:

|  |  |  |
| --- | --- | --- |
| Peso | Categoría | Uso |
| 2 kg o menos | Micro | Recreacional (privado) |
| No comercial (privado) |
| Comercial |
| 2kg a 25 kg | Pequeño | Recracional (privado) |
| No comercial (privado) |
| Comercial |
| Mas de 25 kg | Grande | Recreacional (privado) |
| No comercial (privado) |
| Comercial |

Algunos de los requisitos y limitaciones generales para todas las categorías son:

● Ningún piloto debe arrojar o tirar objetos

● Ningún piloto debe operar el drone en áreas prohibidas o restringidas

● Ningún piloto debe operar el drone donde no sea visible

● El piloto debe mantener el control de la trayectoria de vuelo en todo momento

● Los UAV deben ser operados en horas oficiales

● El UAV no debe ser operado desde vehículos en movimiento

● El piloto del drone no debe operar más de un drone a la vez

Dependiendo de la categoría y el uso, los requisitos y limitaciones cambiarán, el Parrot AR. DRONE 2.0 entra en la categoría micro, todos los UAV de esta categoría no requieren autorización de operación, siempre y cuando los pilotos respeten los siguientes requisitos y limitaciones:

● El propietario del UAV debe registrar todos los drones que pesen más de 250 g en el sitio web de la SCT / DGAC

● El piloto debe operar el zumbido hasta 122 metros

● El drone no debe estar a más de 450 metros de distancia horizontal del piloto

● Debe ser operado a una distancia mínima de 9,2 km de un aeródromo

● El piloto debe operar el UAV a una altura máxima de 100 metros

● El UAV debe ser operado al menos a una distancia de 0,9 km de un helipuerto

● La velocidad dependerá del peso máximo de despegue

**Análisis de factibilidad**

Un análisis de factibilidad ayudó a comparar el proyecto con las tecnologías actuales y los productos, servicios y requerimientos actuales y decidir si los objetivos podían obtenerse o no.

Se consideraron los siguientes aspectos:

● Papeles o Patentes Relacionados

Martin Lefebure tiene la patente WO 2010061099 A2, publicada el 3 de enero de 2010, que se refiere al control de un drone con un dispositivo para realizar órdenes básicas de movimiento.

La invención se refiere a un dispositivo para pilotear un zángano, que incluye un alojamiento provisto de un detector de inclinación de la carcasa y una pantalla táctil, que muestra una pluralidad de zonas sensibles al tacto. El dispositivo incluye un medio controlado por un área táctil que forma un botón de activación / desactivación para conmutar alternativamente el modo de pilotaje de aviones no tripulados entre un modo de activación del sistema de estabilización autónomo del zángano, en el que las órdenes de pilotaje enviadas al zumbido son El resultado de la transformación de señales emitidas por las zonas sensibles al tacto y un modo de desactivación del sistema autónomo de estabilización del zángano, en el que los comandos de pilotaje enviados al zumbido son el resultado de la transformación de señales emitidas por el detector de inclinación del alojamiento.

El método de control en las funciones de pilotaje elementales son funciones del grupo que comprende el despegue, la tierra, arriba, abajo, girar a la derecha, girar hacia la izquierda, hacia adelante, hacia atrás, hacia la derecha, panorámica hacia la izquierda.

● Tecnologías Disponibles

Para el proyecto proyecto se utilizaron diversas bibliotecas y software para conectar con el drone y realmente volarlo. Como se hizo antes, encontramos varias páginas con ayuda sobre los problemas que encontramos en el desarrollo. Entre los programas utilizados, encontramos:

➔ Linux / Ubuntu

➔ ROS Kinetic

➔ Controladores AR Drone

➔ Administrador de texto sublime

➔ Nodo para JavaScript

Además, debemos mencionar que hemos revisado varias guías y bibliotecas de terceros con el fin de ayudarnos a resolver ciertos problemas que ocurrieron durante el desarrollo.

● Productos y servicios actuales

Hoy en día, hay muchos UAV en el mercado, desde los utilizados para fotografías y recreaciones hasta los militares. Por lo general, los "drones" recreativos no tienen opciones para configurarlos como vehículos autónomos autónomos, porque parte de la experiencia ofrecida al usuario es impulsarlos.

Sin embargo, los que se utilizan para las tareas militares de incluso concursos científicos o aficionados pueden incluir la opción de instalar controladores para permitir la codificación de auto-enrutamiento o autónomo volar. Las UA más usadas para esto son la serie Drone AR de Parrot.

● Requisitos y regulaciones legales y éticas

El uso de vehículos aéreos no tripulados está regulado por las respectivas organizaciones aeronáuticas. En México la Dirección General de Aeronáutica Civil había establecido las normas y regulaciones sobre el funcionamiento de estos dispositivos en el correo CO AV-23/10 R2 que determina las limitaciones de uso para evitar accidentes en áreas urbanas o naturales y dicta las respectivas Normas en el espacio aéreo.

● Mejoras identificadas

La principal mejora del proyecto proyecto da al zumbido existente es la capacidad de crear trayectorias autónomas y controlar utilizando el ordenador, al tiempo que recibe el estado actual de los motores y la navegación general; Mientras que el AR Drone 2.0 usualmente sólo usa el teléfono celular del usuario para el control.

**Desarrollo del proyecto**

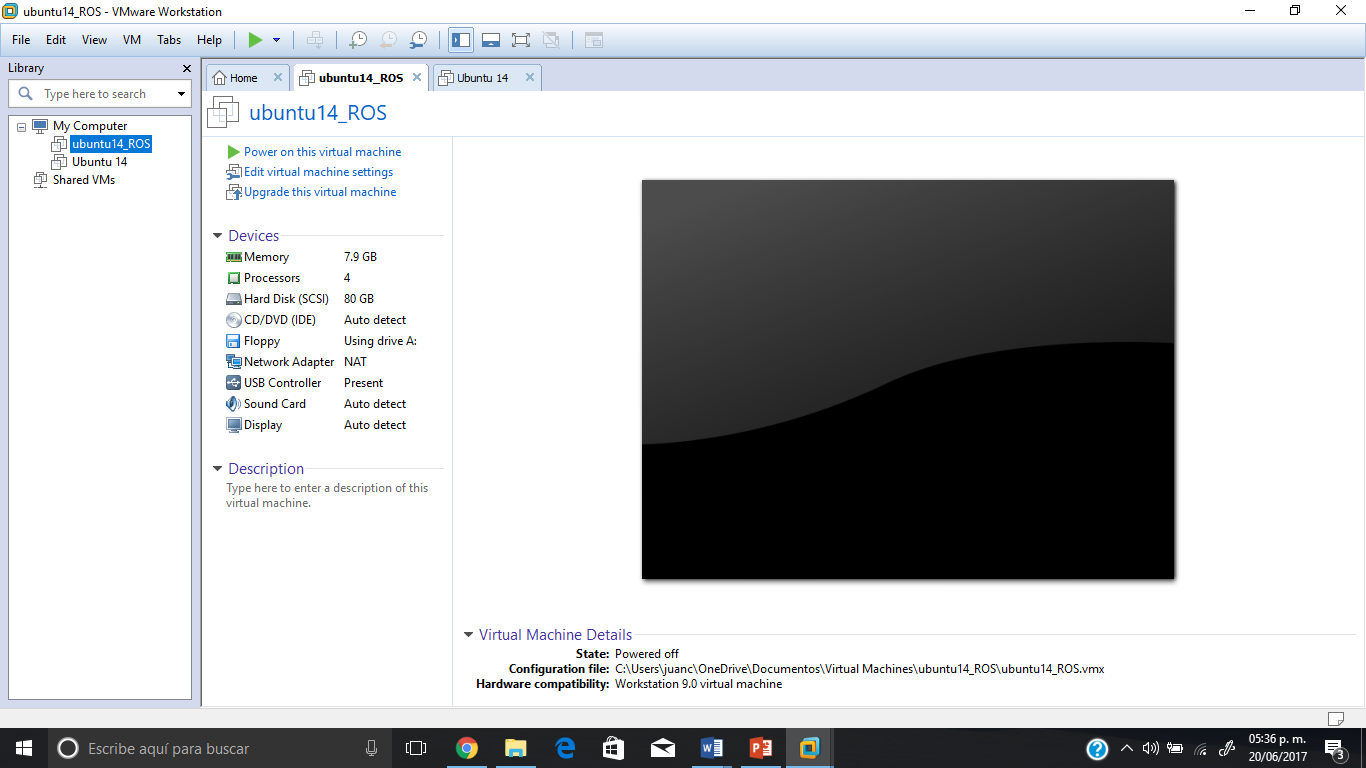
Para comenzar, después de que el proyecto fue asignado, se platicaron los alcances de esta práctica de verano y los objetivos, el proyecto asignado fue: “Diseño y modelado de redes resistivas para la planeación de Trayectorias Robóticas libres de colisiones”. El planteamiento y la resolución todo debe de estar en el sistema operativo ROS. Para la primera semana se realizó una presentación de PowerPoint, que enunciaba las actividades que se realizaron al tomar el curso de drones.

Ya que el curso de drones es totalmente relevante para el tema y la elaboración del proyecto, se realizó la presentación y se presentó, para ver qué tan avanzado podría estar en el uso y manejo de ROS.

Posteriormente se realizó una propuesta de trabajo para realizar en estas prácticas, en la que se proponía la simulación de un dron en un ambiente con obstáculos, generar una trayectoria, ya sea vectorial o con redes resistivas, para llegar de un punto a otro, o con una figura geométrica.

Para este proyecto se utilizó el simulador Gazebo, Ubuntu 14, se utiliza Ubuntu 14 porque es el que tiene más soporte de ROS para realizar actividades relacionadas, y ROS Indigo que es el que tiene más soporte en la página de ROS.

Para realizar esto, primero se descargó VMware, que es un software utilizado para máquinas virtuales, y se utilizó una máquina virtualizada por el Doctor Gerardo, que es el encargado del proyecto; se muestra una imagen a continuación:

Imagen 1: Inicio de máquina virtualizada

Para una utilización óptima de la máquina virtual, se cambió el uso de memoria RAM a aproximadamente 8 gb, además de que se aumentaron los procesadores de 2 a 4.

Al iniciar la máquina virtual, se tenía que abrir una terminal o CMD para ingersar los siguientes comandos para hacer todo el proceso de la simulación del Ar.drone Parrot:

* Instalacion de Ar.drone automomy (por instalación binaria utilizando debian):
  + apt-get install ros-Indigo-ardrone-autonomy
* Instalación del simulador TUM
  + Crear una carpeta de trabajo para el simulador en home:
    - mkdir -p ~/tum\_simulator\_ws/src
    - cd ~/tum\_simulator\_ws/src
    - catkin\_init\_workspace
  + Descargar dependencias:
    - git clone https://github.com/AutonomyLab/ardrone\_autonomy.git # The AR.Drone ROS driver
    - git clone https://github.com/occomco/tum\_simulator.git
    - cd ..
    - rosdep install --from-paths src --ignore-src --rosdistro indigo -y
  + Construir el simulador
    - Catkin make
  + Setear el ambiente
    - source devel/setup.bash
* Instalación de simulador Gazebo (para realizar mapa o ambiente de simulación)
  + Instalación por Debian preconstruidos:
    - sudo apt-get install ros-indigo-gazebo-ros-pkgs ros-indigo-gazebo-ros-control.
* Creación de paquete catkin para carpeta de aplicación dron
  + - cd ~/catkin\_ws/src
    - catkin\_create\_pkg drone\_application std\_msgs rospy roscpp
    - Luego, se crea un archivo .launch en el editor de texto con lo siguiente:
* <launch>
* <include file="$(find gazebo\_ros)/launch/empty\_world.launch">
* <arg name="world\_name" value="$(find cvg\_sim\_gazebo)/worlds/ardrone\_testworld.world"/>
* </include>
* <include file="$(find cvg\_sim\_gazebo)/launch/spawn\_quadrotor.launch" >
* <arg name="model" value="$(find cvg\_sim\_gazebo)/urdf/quadrotor\_sensors.urdf.xacro"/>
* </include>
* </launch>

Ejecutando así los nodos necesarios en paralelo, para mostrar así lo siguiente:

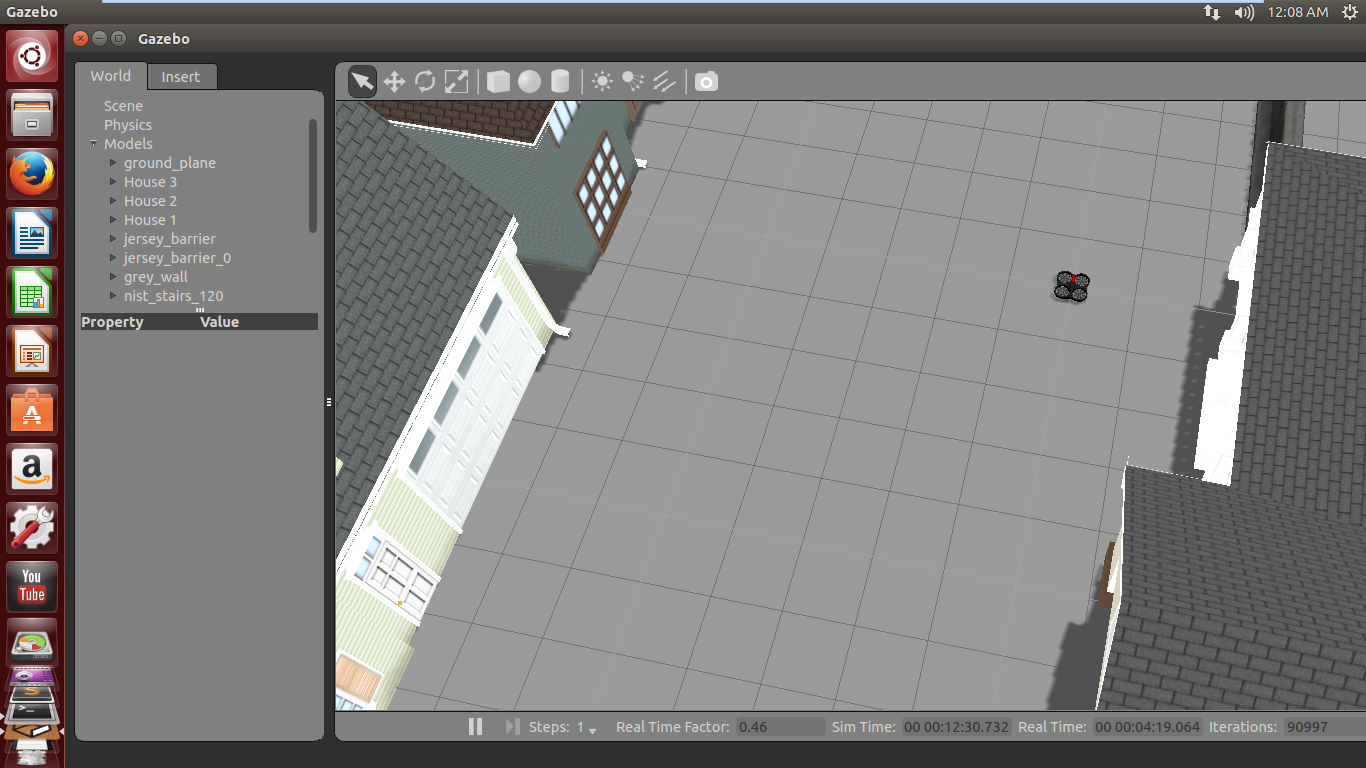


Imagen 2. Simulador Gazebo con cuadrimotor.

Entonces, al publicar mensajes en cmd con los topicos y mensaje adecuado, se pudo realizar el despegue del dron como se muestra a continuación:

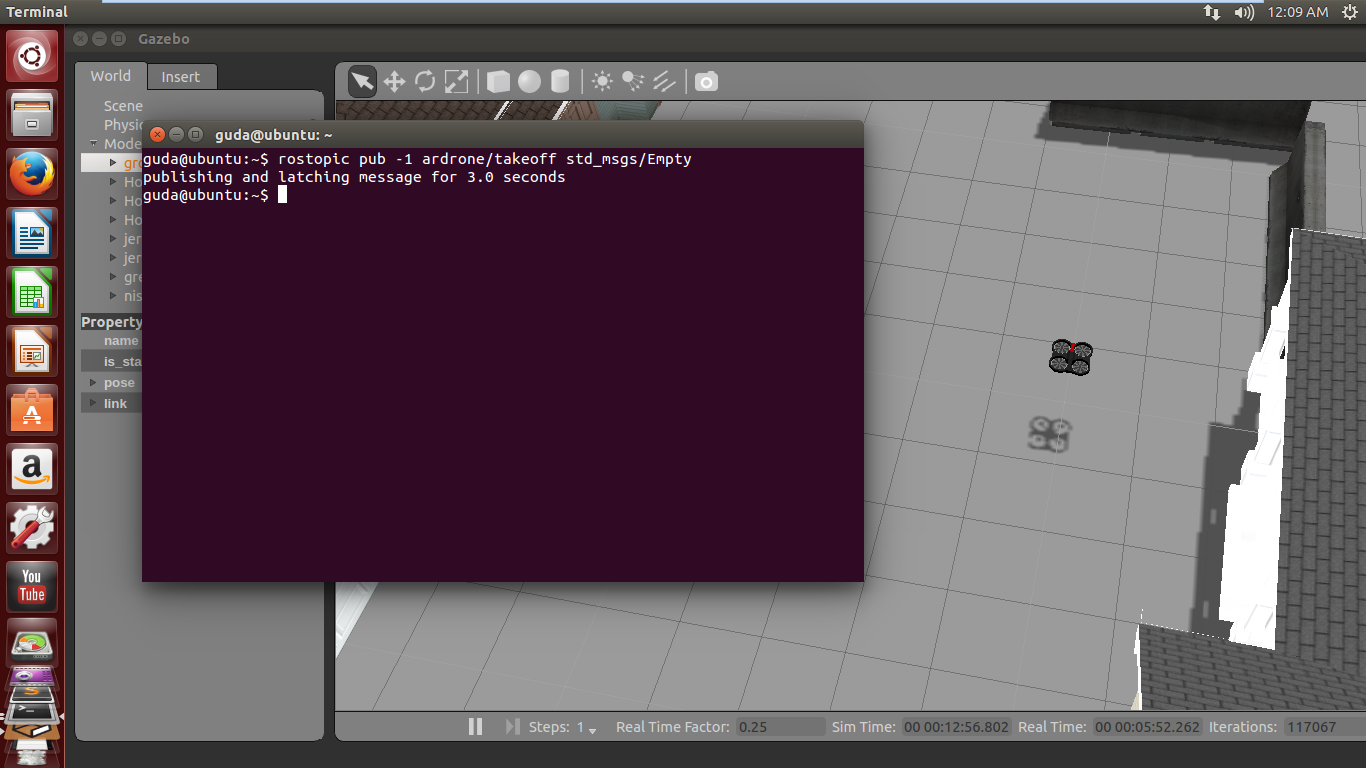


Imagen 3. Despegue del dron.

Para posteriormente realizar el aterrizaje:

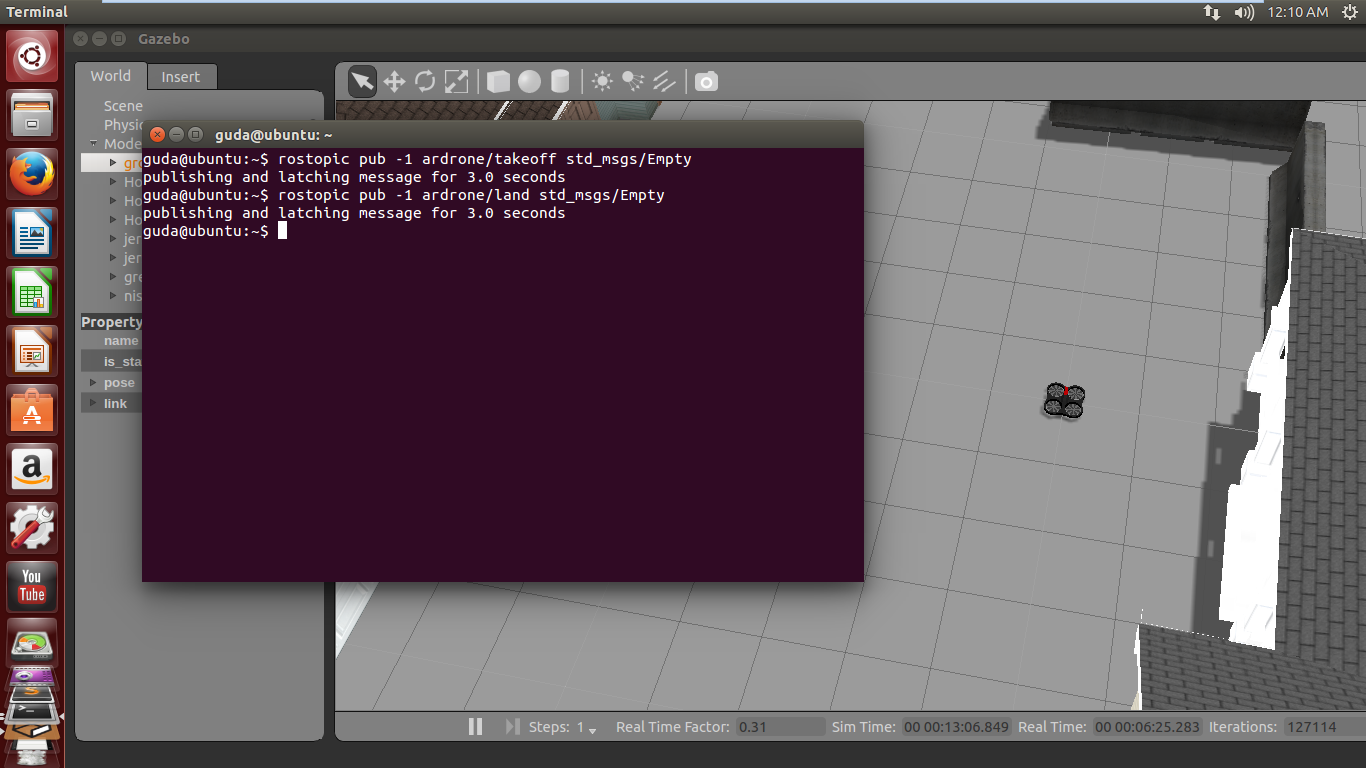


Imagen 4. Aterrizaje del dron.

Después, se muestra cómo hacer que el dron avance cambiando el pitch, a través de la publicación de otro mensaje del tipo twist en CMD, y se muestra como el dron, al llegar a la pared, choca con el entorno, permitiendo así lo más acercado a la realidad.

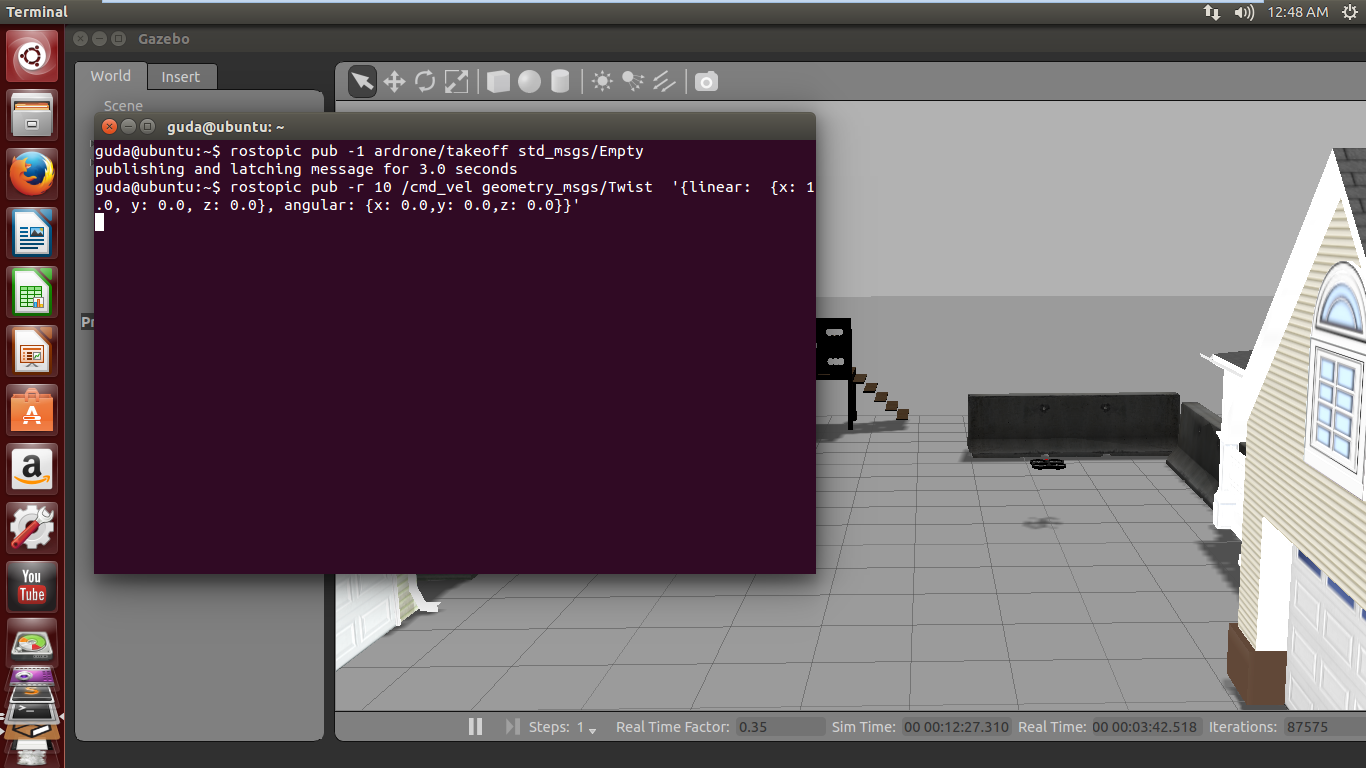


Imagen 5. Movimiento del dron.

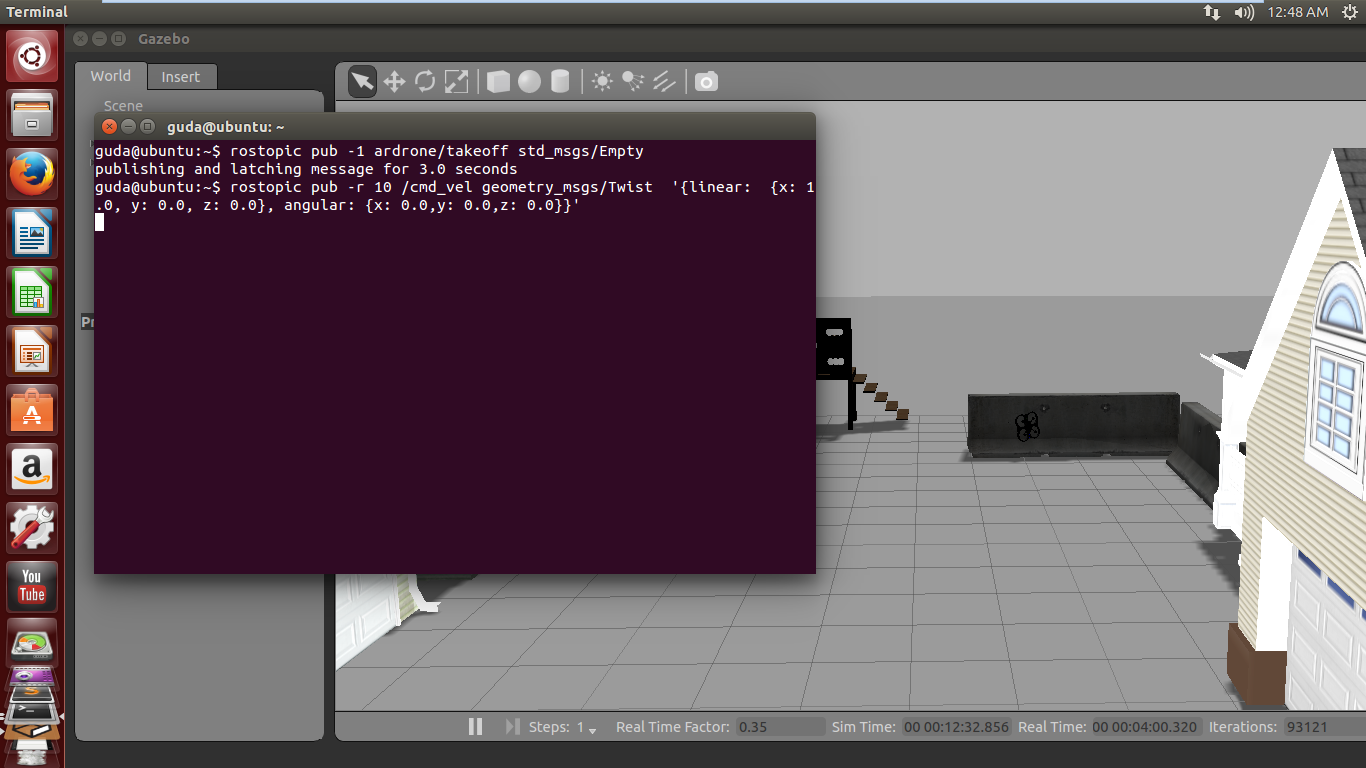


Imagen 6. Dron en choque.

Posteriormente, se realizó una simulación de una trayectoria lineal, se muestra el código en Python en el **Apéndice A**.

Para finalizar, se realizó un código en Python para lograr una trayectoria previamente realizada, con un mapa correspondiente, se muestra el código en el **Apéndice B**.

**Resultados Logrados**

En este proyecto, se obtuvieron los siguientes resultados:

1. Se instaló correctamente Ubuntu, ROS Indigo y los controladores para el Parrot's AR Drone 2.0.
2. Se consiguió que, en la simulación, se simulara un cuadrimotor muy acercado a la realidad, además de la simulación del movimiento y creación de IP ficticia para el uso del Parrot 2.0.
3. Se consiguió la implementación del simulador TUM para la comunicación entre el dron virtual y ROS, para poder envíar mensajes por CMD al dron.
4. Se realizaron también trayectorias generadas por código Python, que eran modificadas posteriormente para el uso del método de “redes resistivas”.

**Conclusiones**

Realizar las prácticas en el INAOE fue una gran experiencia profesional en el ámbito de mecatrónica, porque no sólo te relacionas con gente que te puede aportar mucho a tu formación académica y de profesionista, si no que te permite crecer como persona y ejercer investigación relacionada con tu carrera, utilizando lo previamente aprendido en la universidad.

La sociedad ha implementado drones para uso recreativo o comercial; Pero esto tiene sus limitaciones, los drones operativos para este propósito tienen que seguir reglas para prevenir accidentes debidos al tráfico aéreo. Hoy en día los drones se utilizan en muchas tareas, como en el aspecto ecológico, con vigilancia y estudio de conservas naturales con bajas perturbaciones sobre el medio ambiente o plantación de árboles en zonas áridas. Un problema con estos dispositivos sobre el medio ambiente es la perturbación por el ruido en áreas naturales y urbanas.

El uso de drones es sostenible gracias al desarrollo de nuevas tecnologías que aseguran el equilibrio entre los costes, la experiencia del usuario y el impacto medioambiental. El mercado de UAV es cada vez mayor, con un uso creciente en los últimos años, y con eso también vino el desarrollo de varias mejoras que permiten el uso de drones en nuevas tareas, haciendo la investigación y comercialización de estos productos a prueba de futuro y muy factible.

Además, es importante mencionar que la tecnología UAV también está invirtiendo en la investigación y desarrollo de nuevos materiales, con el objetivo de tener componentes más ligeros pero más resistentes que dan más resistencia y vida útil a los drones, permitiendo al mismo tiempo el uso o en más peligrosos O condiciones hostiles.

La manufacturabilidad de los drones ya está dominada, con varias empresas y plantas dedicadas a la fabricación de estos dispositivos para fines recreativos a militares.

Como conclusión, se aprendió a controlar un drone habitualmente recreativo y convertirlo en una herramienta de desarrollo, capaz de realizar tareas complejas de forma autónoma, solo con la intervención de un usuario para dar la trayectoria a seguir.

**Apéndice A**

#!/usr/bin/env python

import rospy

import time

from std\_msgs.msg import String

from std\_msgs.msg import Empty

from geometry\_msgs.msg import Twist

def move\_forward():

#Se inicia el nodo con cualquier nombre y los publicadores

#Tamano de cola o queue size 10, permite que hayan 10 "sentencias" en cola

rospy.init\_node('Takengo', anonymous=True)

pub = rospy.Publisher("cmd\_vel", Twist, queue\_size=10)

pub2 = rospy.Publisher("ardrone/takeoff", Empty, queue\_size=10 )

pub3 = rospy.Publisher("ardrone/land", Empty, queue\_size=10 )

#se asigna el atributo tipo Twist a command

command = Twist()

#Distancia y velocidad que quieres que se mueva el drone

print("El drone se movera hacia delante y aterrizar ")

speed = input("Introduce la velocidad del drone ")

distance = input("Introduce la distancia que quieres que se mueva ")

#Se asigna la velocidad teorica del dron puede ir de (-1 a 1)

command.linear.x = abs(speed)

command.linear.y = 0

command.linear.z = 0

command.angular.x = 0

command.angular.y = 0

command.angular.z = 0

current\_time = 0

current\_time1 = 0

current\_time2 = 0

rate = rospy.Rate(10) # 10hz, standard

while not rospy.is\_shutdown():

#Utilizando funcion de rospy para el tiempo actual

t0 = rospy.Time.now().to\_sec()

#Poniendo la distancia actual como 0

current\_distance = 0

while (current\_time < 3):

#Se lanza el takeoff

pub2.publish(Empty())

#Da tiempo para hacer el takeoff, 3 segundos

t2=rospy.Time.now().to\_sec()

current\_time = t2-t0

t0 = rospy.Time.now().to\_sec() #Reinicializa t0 para el siguiente while

#Hacer un loop para que el drone se mueva la distancia teorica

while(current\_distance < distance):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance= speed\*(t1-t0)

#Imprime la distancia actual

#Despues del loop para que el dron se pare

command.linear.x = 0 #Velocidad igual a 0

#Forzarlo a parar y aterrizar

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while (current\_time1 < 2):

#Se lanza el hover (Todas las velocidades son iguales a 0)

pub.publish(command)

#Da tiempo para hacer el hover, 2 segundos

t2=rospy.Time.now().to\_sec()

current\_time1 = t2-t0

t0 = rospy.Time.now().to\_sec()

while (current\_time2 < 2):

#Se lanza el land

pub3.publish(Empty())

#Da tiempo para hacer el land

t2=rospy.Time.now().to\_sec()

current\_time2 = t2-t0

rate.sleep() #Hace que el loop se mantenga a 10 Hz

if \_\_name\_\_ == "\_\_main\_\_": #Main donde ejecuta la funcion, el except es el ctrl+c

try:

move\_forward()

except rospy.ROSInterruptException:

pass

**Apéndice B**

import rospy

import time

from std\_msgs.msg import String

from std\_msgs.msg import Empty

from geometry\_msgs.msg import Twist

def move\_forward():

#Se inicia el nodo con cualquier nombre y los publicadores

#Tamano de cola o queue size 10, permite que hayan 10 sentencias en cola

rospy.init\_node('Trayectoria', anonymous=True)

pub = rospy.Publisher("cmd\_vel", Twist, queue\_size=10)

pub2 = rospy.Publisher("ardrone/takeoff", Empty, queue\_size=10 )

pub3 = rospy.Publisher("ardrone/land", Empty, queue\_size=10 )

#se asigna el atributo tipo Twist a command

command = Twist()

#Distancia y velocidad que quieres que se mueva el drone

print("El drone hara la trayectoria proporcionada")

speed = input("Introduce la velocidad del drone ")

print("Coordenada 1 1 6")

#Se asigna la primera cordenada

command.linear.x = abs(speed)/6

command.linear.y = abs(speed)/6

command.linear.z = abs(speed)

distance = 6

current\_time = 0

current\_time1 = 0

current\_time2 = 0

current\_time3 = 0

current\_time4 = 0

current\_time5 = 0

current\_time6 = 0

current\_time7 = 0

current\_time8 = 0

current\_time9 = 0

current\_time10 = 0

current\_time11 = 0

current\_time12 = 0

rate = rospy.Rate(10) # 10hz, standard

while not rospy.is\_shutdown():

#Utilizando funcion de rospy para el tiempo actual

t0 = rospy.Time.now().to\_sec()

#Poniendo la distancia actual como 0

current\_distance = 0

while (current\_time < 2):

#Se lanza el takeoff

pub2.publish(Empty())

#Da tiempo para hacer el takeoff, 3 segundos

t2=rospy.Time.now().to\_sec()

current\_time = t2-t0

t0 = rospy.Time.now().to\_sec() #Reinicializa t0 para el siguiente while

#Hacer un loop para que el drone se mueva la distancia teorica

while(current\_distance < distance):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance= speed\*(t1-t0)

#Hover

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

t0 = rospy.Time.now().to\_sec()

while (current\_time1 < 1.5):

#Se lanza el land

pub.publish(command)

#Da tiempo para hacer el hover

t2=rospy.Time.now().to\_sec()

current\_time1 = t2-t0

#Termina Hover

#coordenada (1 2 6)

print("Coordenada 1 2 6")

command.linear.x = 0

command.linear.y = 1

command.linear.z = 0

distance1 = 1

current\_distance1 = 0

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while(current\_distance1 < distance1):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance1 = (t1-t0)

#Hover

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

t0 = rospy.Time.now().to\_sec()

while (current\_time2 < 1.5):

#Se lanza el land

pub.publish(command)

#Da tiempo para hacer el hover

t2=rospy.Time.now().to\_sec()

current\_time2 = t2-t0

#Termina Hover

#coordenada (2 3 7)

print("Coordenada 2 3 7")

command.linear.x = 1

command.linear.y = 1

command.linear.z = 1

distance2 = 1

current\_distance2 = 0

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while(current\_distance2 < distance2):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance2 = (t1-t0)

#Hover

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

t0 = rospy.Time.now().to\_sec()

while (current\_time3 < 1.5):

#Se lanza el land

pub.publish(command)

#Da tiempo para hacer el hover

t2=rospy.Time.now().to\_sec()

current\_time3 = t2-t0

#Termina Hover

#coordenada (3 4 6)

print("Coordenada 3 4 6")

command.linear.x = 1

command.linear.y = 1

command.linear.z = -1

distance3 = 1

current\_distance3 = 0

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while(current\_distance3 < distance3):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance3 = (t1-t0)

#Hover

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

t0 = rospy.Time.now().to\_sec()

while (current\_time4 < 1.5):

#Se lanza el land

pub.publish(command)

#Da tiempo para hacer el hover

t2=rospy.Time.now().to\_sec()

current\_time4 = t2-t0

#Termina Hover

#coordenada (4 4 7)

print("Coordenada 4 4 7")

command.linear.x = 1

command.linear.y = 0

command.linear.z = 1

distance4 = 1

current\_distance4 = 0

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while(current\_distance4 < distance4):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance4 = (t1-t0)

#Hover

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

t0 = rospy.Time.now().to\_sec()

while (current\_time5 < 1.5):

#Se lanza el land

pub.publish(command)

#Da tiempo para hacer el hover

t2=rospy.Time.now().to\_sec()

current\_time5 = t2-t0

#Termina Hover

#coordenada (5 4 7)

print("Coordenada 5 4 7")

command.linear.x = 1

command.linear.y = 0

command.linear.z = 0

distance5 = 1

current\_distance5 = 0

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while(current\_distance5 < distance5):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance5 = (t1-t0)

#Hover

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

t0 = rospy.Time.now().to\_sec()

while (current\_time6 < 1.5):

#Se lanza el land

pub.publish(command)

#Da tiempo para hacer el hover

t2=rospy.Time.now().to\_sec()

current\_time6 = t2-t0

#Termina Hover

#coordenada (6 5 6)

print("Coordenada 6 5 6")

command.linear.x = 1

command.linear.y = 1

command.linear.z = -1

distance6 = 1

current\_distance6 = 0

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while(current\_distance6 < distance6):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance6 = (t1-t0)

#Hover

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

t0 = rospy.Time.now().to\_sec()

while (current\_time7 < 1.5):

#Se lanza el land

pub.publish(command)

#Da tiempo para hacer el hover

t2=rospy.Time.now().to\_sec()

current\_time7 = t2-t0

#Termina Hover

#coordenada (6 6 5)

print("Coordenada 6 6 5")

command.linear.x = 0

command.linear.y = 1

command.linear.z = -1

distance7 = 1

current\_distance7 = 0

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while(current\_distance7 < distance7):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance7 = (t1-t0)

#Hover

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

t0 = rospy.Time.now().to\_sec()

while (current\_time8 < 1.5):

#Se lanza el land

pub.publish(command)

#Da tiempo para hacer el hover

t2=rospy.Time.now().to\_sec()

current\_time8 = t2-t0

#Termina Hover

#coordenada (7 7 4)

print("Coordenada 7 7 4")

command.linear.x = 1

command.linear.y = 1

command.linear.z = -1

distance8 = 1

current\_distance8 = 0

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while(current\_distance8 < distance8):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance8 = (t1-t0)

#Hover

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

t0 = rospy.Time.now().to\_sec()

while (current\_time9 < 1.5):

#Se lanza el land

pub.publish(command)

#Da tiempo para hacer el hover

t2=rospy.Time.now().to\_sec()

current\_time9 = t2-t0

#Termina Hover

#coordenada (8 8 3)

print("Coordenada 8 8 3")

command.linear.x = 1

command.linear.y = 1

command.linear.z = -1

distance9 = 1

current\_distance9 = 0

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while(current\_distance9 < distance9):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance9 = (t1-t0)

#Hover

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

t0 = rospy.Time.now().to\_sec()

while (current\_time10 < 1.5):

#Se lanza el land

pub.publish(command)

#Da tiempo para hacer el hover

t2=rospy.Time.now().to\_sec()

current\_time10 = t2-t0

#Termina Hover

#coordenada (8 9 2)

print("Coordenada 8 9 2")

command.linear.x = 0

command.linear.y = 1

command.linear.z = -1

distance10 = 1

current\_distance10 = 0

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while(current\_distance10 < distance10):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance10 = (t1-t0)

#Hover

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

t0 = rospy.Time.now().to\_sec()

while (current\_time11 < 1.5):

#Se lanza el land

pub.publish(command)

#Da tiempo para hacer el hover

t2=rospy.Time.now().to\_sec()

current\_time11 = t2-t0

#Termina Hover

#coordenada (9 9 1)

print("Coordenada 9 9 1")

command.linear.x = 1

command.linear.y = 0

command.linear.z = -1

distance11 = 1

current\_distance11 = 0

t0 = rospy.Time.now().to\_sec() #Reinicializacion de t0

while(current\_distance11 < distance11):

#Publica la velocidad

pub.publish(command)

#Toma el tiempo actual

t1=rospy.Time.now().to\_sec()

#Calcula la distancia actual, teorica

current\_distance10 = (t1-t0)

command.linear.x = 0

command.linear.y = 0

command.linear.z = 0

pub.publish(command)

rate.sleep() #Hace que el loop se mantenga a 10 Hz

if \_\_name\_\_ == "\_\_main\_\_": #Main donde ejecuta la funcion, el except es el ctrl+c o z

try:

move\_forward()

except rospy.ROSInterruptException:

pass

**Referencias**

* Geisendörfer, F. (2017, 5 de Julio). Felixge/node-ar-drone. [en línea]. Disponible en: https://github.com/felixge/node-ar-drone
* Hamer, M. (2017, 4 de Junio). Up and flying with the AR.Drone and ROS: Getting started | Robohub. [en línea]. Disponible en: http://robohub.org/up-and-flying-with-the-ar-drone-and-ros-getting-started.
* Mexico, SCT, DGAC. (2016, 12 de Junio). Circular obligatoria que establece los requerimientos para operar un sistema de aeronave pilotada a distancia (pp. 1-47). [en línea] Disponible en: http://www.sct.gob.mx/fileadmin/DireccionesGrales/DGAC-archivo/modulo3/co-av-23-10-r3.pdf.
* Nodecopter.com (2017, 8 de Junio). Hacker Guide. [en línea] Disponible en: http://www.nodecopter.com/hack
* Node.js Fundación (2017, 7 de Mayo). Node.js. [en línea] Disponible en: https://nodejs.org/es/
* Wiki.ros.org. (2017, 20 de Junio). Documentation - ROS Wiki. [en línea]. Disponible en at: http://wiki.ros.org/
* Wiki.ros.org. (2017, 6 de Abril). wiimote - ROS Wiki. [en línea] Disponible en: http://wiki.ros.org/wiimote?distro=kinetic