UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Montaje y puesta a punto de un dron para agricultura de precisión

TRABAJO FIN DE GRADO

GRADO EN INGENIERIA EN TECNOLOGIAS INDUSTRIALES

Autor: **JUAN JOSÉ GARCÍA ANDREU**

Director: Juan Antonio López Riquelme

Codirectora: Nieves Pavón Pulido

Cartagena, a 9/12/2020

A mi familia,
A mis amigos,
Y a mi perro.

"Un viaje de mil millas comienza con un primer paso."
Lao-Tsé

Agradecimientos

Tal vez se pueda pensar que el trabajo es únicamente resultado de mi trabajo, tal vez, pero no es así hay mucha gente detrás de el por eso quiero agradecerles a ellos a todos ellos el apoyo y la ayuda que de tantas formas distintas he recibido.
Por ello quiero agradecer en especial a mi familia y a amigos, no solo los que tenía antes de esta aventura, los de toda la vida, también a los nuevos, GRACIAS.
También quiero recordar a mi compañero de vida, mi perro Lucas.
No quiero olvidarme de la inestimable ayuda de mis tutores en este proyecto Juan Antonio y Nieves, muchas gracias por todo.

Y para finalizar, solo quiero añadir que, tal vez haya tardado demasiado en llegar hasta este punto, tal vez se puede creer eso, tal vez, pero solo unos cuantos saben lo que he pasado, solo unos cuantos, por eso a ellos, a esas pocas personas en especial quiero darle gracias, muchas gracias.

Pero por encima de todo, a ti, si a ti . A mi madre que ha sufrido conmigo cada paso que he dado, gracias, gracias y mil gracias. porque sin tu ayuda esto hubiera sido posible.
Juanjo.
Cartagena,2020.

- *"Lo que importa verdaderamente en la vida no son los objetivos que nos marcamos, sino los caminos que seguimos para lograrlo."*
  - *Peter Bamm*

- *"Cuanto más alto coloque el hombre su meta, tanto más crecerá."*
  - *Friedrich Von Schiller*

- *"Quien no madruga pierde el día, y quien pierde la juventud pierde la vida"*
  - *J.A. Andreu*

Índice de contenido

Índice de Figuras

ANEXO II 93

Índice de tablas

Resumen

El objetivo principal de este TFG es la puesta a punto y el ensamblaje de un dron que se pueda utilizar como plataforma para llevar a cabo diversas misiones en el entorno de la agricultura de precisión. Para ello, una vez montado y calibrado el dron, será necesario estudiar el middleware para robótica ROS (*Robotic Operating System*), así como los paquetes ROS disponibles para el dron utilizado en este trabajo.
En el actual documento indica el procedimiento llevado a cabo para la formación de un espacio virtual donde se realizará la simulación de un dron. En concreto, realizaremos una simulación en un medio virtual, en el que probaremos con varios parámetros y trayectorias, con el fin de realizar las misiones de precisión. Para conseguir realizar esta simulación se han utilizado unas herramientas básicas como pueden ser ROS y Gazebo.

Abstract

The main objective of this TFG is the assembly and tuning of a drone that can be used as a platform to carry out various missions in the field of precision agriculture. To do this, once the drone is assembled and calibrated, it will be necessary to study the ROS (Robotic Operating System) robotics middleware, as well as the ROS packages available for the drone used in this work.
This document reflects the method carried out for the formation of a simulated world where a drone model will be carried out. Specifically, we will carry out a simulation in a virtual environment, in which we will test with various parameters and trajectories, in order to carry out precision missions. To achieve this simulation, basic tools such as ROS and Gazebo have been used.

Capítulo I

Introducción

1.1 Contexto

La agricultura de precisión es un concepto agronómico que surge con el objetivo de entender y gestionar apropiadamente los cultivos, como puede ser el hecho de realizar un seguimiento de la producción. En concreto, se basa en seguir una serie de fases (monitorización, análisis de datos, toma de decisiones y actuación), en las cuales es habitual emplear Tecnologías de la Información y las Comunicaciones, tales como GPS, GIS, redes de sensores, etc.

Con respecto a la fase de monitorización, es habitual desplegar sensores de suelo, agua, planta y ambiente para estudiar los cultivos. Además de estos sensores, los sensores multiespectrales y térmicos a bordo de UAVs (*Unmanned Aerial Vehicles*) están siendo muy utilizados en los últimos años para estudiar los cultivos agronómicos de una forma rápida. Algunos de los modelos comerciales son el erle-copter, de la empresa erlerobotics fundada por los dos hermanos Mayoral Vilches, oriundos de Álava, y el AR dron 2.0 de Parrot.

Figura 1.1: Erle-copter & AR Dron 2.0

1.2 Motivación

En la situación actual los drones han sufrido una gran evolución debido a la multitud de situaciones en las que estos pueden ser utilizados. Ya que estos cuadricópteros son capaces de realizar tareas que sin ellos pueden resultar difíciles de realizar, haciendo de esta multitud de tareas de una forma más precisa y eficiente. Como consecuencia dentro del ámbito de la agricultura se considera útil la utilización de drones para el uso de distintos trabajos como pueden ser vigilancia y control, entre otros.

Por eso el principal motivo de este TFG es realizar el montaje de un dron para que se pueda llevar a cabo diversas tareas en el entorno de la agricultura de precisión, además de hacer su puesta a punto. Una vez montado y calibrado el dron, será necesario estudiar el middleware para robótica denominado (*Robotic Operating System*) para llevar a cabo la puesta a punto, es decir, los paquetes disponibles para el dron utilizado en este proyecto.
Además, en el actual documento indica el procedimiento llevado a cabo para la formación de un espacio virtual donde se realizará la simulación de un dron.
Finalmente, se desarrollarán todos los componentes software necesarios, de manera que se desarrolle un caso de estudio utilizando un dron en agricultura. Dicho caso de estudio se realizará de la forma más estructural posible, de manera que la compresión del trabajo realizado sea fácil para que al retomarlo para llevar a cabo otras misiones con el dron en el mismo campo de estudio.

Originalmente este Trabajo Fin de Grado se enfocaba en el Erle-Copter, tanto en su modelo como en los controladores que usa mientras realiza el vuelo, pero al originarse ciertos problemas con los archivos para su simulación(falta de soporte), se ha optado a realizar el trabajo con el dron de Parrot AR dron 2.0. Haremos una simulación en un medio virtual, gazebo, en el cual haremos ensayos con distintos movimientos y variables, para posteriormente realizar las misiones de precisión.
Por último, cabe destacar que para conseguir realizar esta simulación se han utilizado algunas herramientas básicas como pueden ser ROS y Gazebo.

1.3 Objetivos del Trabajo

El objetivo principal de este Trabajo Fin de Grado es el montaje y configuración de un dron comercial. En concreto, se plantean los siguientes objetivos específicos, de manera que el robot aéreo se pueda utilizar en un caso de estudio agronómico:

- Selección y estudio de un dron para diseñar y desarrollar un caso de estudio en el ámbito de la agricultura de precisión

- Estudiar el middleware para robótica ROS, así como los drivers existentes para el mismo.

- Diseñar y desarrollar los componentes software necesarios para realizar una demostración de funcionamiento del equipo.

1.4 Estructura

- Cap.1: Introducción y estructura del TFG:

  En el capítulo en el que estamos introduciremos los diferentes aspectos afines a la motivación y las metas que queremos lograr en la evolución del TFG. Asimismo, se establece el contexto de trabajo en el que se enmarca esta la estructura e investigación de este TFG.

- Cap.2: Estado del arte:

  En el segundo capítulo se pueden observar las descripciones relacionadas con las herramientas utilizadas y medios que son usados y por tanto necesarios para la realización del trabajo.

- Cap.3: Descripción del sistema:

  En esta sección, correspondiente al tercer capítulo, explicamos la características oportunas de cada dron, así como los motivos por los que se selecciona el dron con el que se realizara el TFG . también se realiza una introducción a ROS, explicando que es, como instalarlo, añadiendo una introducción de cómo podemos usarlo y los motivos de la selección de una distribución en concreto.
  Además de exponer el simulador usado, realizando una explicación de su uso, características y aplicaciones.

- Cap.4: Descripción del trabajo realizado:

  En dicho capítulo, el cuarto, explicaremos los archivos desarrollados en este TFG, los cuales permiten llevar a cabo simulaciones de un dron en un ámbito agronómico.

- Cap.5: Conclusiones y trabajos futuros:

  En este es el capítulo en el cual detallo las conclusiones obtenidas al finalizar el proyecto además de cuáles son las posibles líneas de desarrollo que se pueden beneficiar de este trabajo.

Capítulo II

Estado del arte

2.1 introducción

En este apartado hablaremos sobre las herramientas y medios que son usados, por tanto, necesarios para la realización del trabajo del dron para el desarrollo de este TFG.

2.2 Plataformas robóticas aéreas.

2.2.1 Erle-copter.

David Mayoral Vilches y su hermano Victor son los creadores de la gran startup llamada Erle Robotics, la que fue creada en Álava. Comenzó a finales del 2012, el motivo de llamarla Erle es porque en euskera significa "abeja", este proviene de los primeros drones ya que estos producían un ruido parecido al que hacen las abejas.
Se trata de un dron inteligente basado en el sistema operativo Linux, para ser exacto Ubuntu 14, con soporte oficial para un ecosistema robóticos como ROS. Su elemento más importante es el Erle-Brain 2, módulo electrónico que actúa como un pequeño ordenador Linux, el cual tiene ROS preinstalado. Posee todo el software y sensores necesarios para convertir al dron en un dispositivo autónomo.
Proporciona una Unidad de Control de Vuelo (FCU o FlightControlUnit), encargada de aportar controles básicos de vuelo, también denominado autopiloto. Además de esto, tiene integrados una serie de sensores, tales como una IMU (acelerómetros y giroscopios), GPS, brújula y barómetro, los cuales le permiten tener una estimación de la posición y la orientación (1, s.f.).

Figura 2.0: Erlecopter blanco y rojo

2.2.2 Parrot AR Dron 2.0

Este es un vehículo aéreo no tripulado de radiocontrol diseñado para el ocio. Funciona mediante cuatro motores eléctricos que están colocados en una forma de cuadricóptero, los cuales ejercen una propulsión para moverlo, esta estructura es igual a otros modelos, también poseen la misma aerodinámica , sin embargo varían de estos otros modelos en que tienen un microprocesador y unos sensores, que contienen cámaras las cuales sirven para obtener una imagen de su entorno, además de una conexión Wi-Fi integrada que le ofrece la posibilidad de vinculación a diferentes terminales móviles que incluyan los distintos sistemas operativos que corresponden con Linux, iOS y Android. lo cual da lugar a que se pueda manejar de forma directa al cuadricóptero a través de uno de estos terminales, durante este proceso se obtiene la información, imágenes y datos de lo que los sensores están percibiendo mediante la telemetría(2, s.f.).

Figura 2. 1: AR Drone 2.0

2.3 Sistemas operativos

Ubuntu está creado para el correcto funcionamiento con los ordenadores de última generación y dispositivos táctiles, por ello permite una visión de calidad en pantallas de alta resolución y se considera un sistema operativo de código abierto.
Originalmente, se basa en Debian y usa Linux como núcleo, que es otra alternativa que se podría usar para realizar el trabajo.
Ubuntu es un sistema operativo accesible, incluye ayuda técnica básica y está traducido a más de 50 idiomas.
La seguridad es una de las características que definen a este sistema operativo, el cual tendría una existencia nula sino contara con una comunidad de desarrolladores a nivel mundial, siendo el trabajo de estos totalmente voluntario. (3, s.f.).

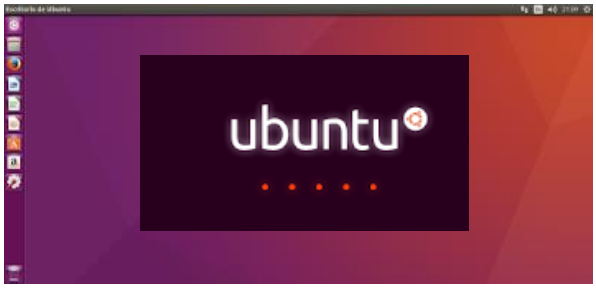


Figura 2. 2: Ubuntu

2.4 Middlewares para robótica

Robot Operating System (ROS) es un conjunto de frameworks que permite que el software de los robots se desarrolle correctamente. De esta manera, se considera un middleware robótico. En el año 2007 fue la presentación de ROS, llamándose switchyard, fue desarrollado en Stanford, gracias al Laboratorio de Inteligencia Artificial de este sitio, como soporte que permitió la realización de su proyecto STAIR2, es decir, su Robot con Inteligencia Artificial. En el siguiente año, el instituto Willow Garage, el cual se especializa en investigar los campos de la robótica, fue el lugar donde siguió evolucionando principalmente (4, s.f.).

Figura 2. 3: Logo de Ros

Aunque no es sistema operativo, ROS puede realizar la misma función que cualquiera de ellos, como son el mantenimiento de paquetes, la implementación de funcionalidad de uso común, el control de dispositivos de bajo nivel, el paso de mensajes entre procesos y la abstracción del hardware.

Existen otros middlewares que se podían haber usado como por ejemplo yarp .

El proceso se lleva a cabo en los nodos que son capaces de recibir, mandar y multiplexar mensajes de control, sensores, actuadores y de diferentes tipos. Las librerías están desarrolladas para un sistema Ubuntu -Linux (UNIX), falta añadir que está basado en una arquitectura de grafos (5, s.f.).

Figura 2.3 - 1:Logo de la fundación Open Source

ROS se basa en dos pates: la parte de ros, ros-pkg, y sistema operativo. Esta última consiste en un grupo de paquetes proporcionados por los usuarios.
ROS tiene licencia BSD y es un software libre. Esto permite un uso investigador y comercial. Hay una gran diversidad en las licencias de los paquetes en ros-pkg . En este TFG trabajaremos con ROS kinetic.

Figura 2.3 - 2: ROS Kinetic

2.4 Gazebo

Gazebo y webots son simuladores dinámicos 3D con la capacidad de simular de forma precisa y eficiente poblaciones de robots en entornos complejos de interior y exterior. Aunque es similar a los motores de juegos, en nuestro caso usaremos gazebo que ofrece simulación física con un grado de fidelidad mucho más alto, un conjunto de sensores e interfaces tanto para usuarios como para programas (6, s.f.).

En definitiva, es una herramienta muy completa que nos permite simular el movimiento de un robot en un ambiente simulado. Sin embargo, el mundo por defecto no tiene ninguna veracidad ni obstáculos, así que es bastante necesario la creación de distintos obstáculos o, en nuestro caso, de un mundo en el que poder simular situaciones reales
Los usos típicos de Gazebo incluyen:

- probar algoritmos de robótica.
- diseñar robots.
- realizar pruebas de regresión con escenarios realistas.

Algunas características clave de Gazebo incluyen:

- múltiples motores de física.
- una rica biblioteca de modelos y entornos de robots.
- una amplia variedad de sensores.
- convenientes interfaces programáticas y gráficas.

Figura 2. 4: Logo del simulador de Gazebo

2.5 protocolos de comunicación

Cosiste en un cúmulo de normas que deben realizar todos los programas y máquinas que actúan en una comunicación de datos entre ordenadores sin las cuales esta seria confusa e imposible.

Ejemplo de estos protocolos pueden ser WiFi y comunicaciones 2G/3G/4G con módulos LTE. Si las comentamos brevemente podemos decir que la utilización de estas redes se está siendo más habitual.
La conexión wifi es un grupo de características de las redes inalámbricas a nivel local (WLAN), están fundadas en el estándar IEEE 802.11. la definición de "Wi-Fi" es debida a "Wireless Fidelity", un término de origen anglosajón. Este tipo de redes son inalámbricas, debido a que la transferencia de datos es mediante el uso de radiofrecuencia.

Si hacemos referencia a las comunicaciones 2G/3G/4G con módulos LTE podemos afirmar que estas siglas representan la tecnología que está utilizando la conexión de la red de comunicación móvil. la "G" hace referencia a generación y el numero a la evolución de dicha red. Uno de los estándares de las comunicaciones de este tipo para una alta velocidad de transmitir datos, usualmente utilizada para telefonía y terminales, son los módulos LTE (Long Term Evolution).

2.6 lenguajes de programación

En este punto hablaremos de los lenguajes de programación. ROS soporta actualmente diversos lenguajes, pero nos centraremos en C++ y Python, los cuales son de los más relevantes sin tener en cuenta java, que en la actualidad es el más usado.

Python es un lenguaje ideal para principiantes por su simpleza, legibilidad y similitud con el inglés, su funcionamiento es en multiplataforma y multiparadigma.
C++ es un lenguaje híbrido y multiparadigma. Además de ser muy didáctico, con este lenguaje puedes aprender muchos otros tipos de lenguajes con gran facilidad. Es una continuación y ampliación del C.

2.7 conclusiones

En este capítulo se han estudiado algunas alternativas para diseñar y desarrollar los elementos fundamentales necesarios en este trabajo. En concreto, se han presentado dos plataformas robóticas aéreas, junto con algunos protocolos de comunicación que se pueden utilizar, así como lenguajes de programación con los que se pueden desarrollar los componentes software necesarios.
En siguiente capítulo tiene como objetivo proporcionar una descripción detallada de los diferentes elementos implicado en este trabajo, las plataformas robóticas las cuales hemos usado dos debido a las situaciones que nos han ido obligando a tomar una serie de decisiones ajenas a nuestra voluntad, que el procedimiento expuesto en este TFG es extrapolable a una gran cantidad de drones diferentes y por tanto es útil para proyectos futuros, para ampliar la programación de estos drones o incluso para aplicar esta programación a otros, también se puede ver que el sistema operativo no tiene por qué ser uno, sino que podemos elegir, pero esta elección siempre estará sujeta al middleware que vayamos a usar para trabajar con el dron, lo mismo sucede con el simulador. Además, se menciona los protocolos que existen para este tipo de procedimientos y los lenguajes que hay y que usaremos el C++.

Dicho esto, este capítulo se centra en introducir las herramientas que se usan y que serán útiles para el desarrollo del TFG, independientemente de las opciones que hay escogimos estas por ser bajo nuestro criterio las más eficientes, como se puede ver estas herramientas son muy útiles y se puede obtener rápida información de ellas.

Son muy utilizadas, por ello es de mucha utilidad aprender a trabajar con ellas ya que en un futuro pueden servir de gran ayuda para el aprendizaje, como de ayuda a quien esté interesado en este TFG

Capítulo III

Descripción del sistema

3.0.1 Introducción

En este capítulo se expondrá sobre el dron que en inicio se iba a usar para desarrollar este TFG, hablaremos sobre su origen, de su montaje y los motivos que nos llevaron a optar por otro dron, y de cómo llegamos a ese punto.
En concreto, se describe en detalle el cuadricóptero que al final se ha utilizado para desarrollar la puesta a punto en este TFG, es decir, el AR Drone 2.0 de Parrot, sobre su origen, de sus características y los componentes.

También se describirá ROS, el middleware llamado Sistema Operativo Robótico (*Robot Operating System*, ROS), que se ha usado para desarrollar este TFG. En concreto, se explicarán diversos aspectos como, por ejemplo, sus funcionalidades principales y cómo se instala.

Además, se mencionará la versión de ROS que se ha usado, de las distintas situaciones que se han producido en el proceso de instalación y de su uso.
También hablaremos sobre Gazebo. Este trabajo se centrará en Gazebo que es el simulador que se va a usar en el entorno de ROS, este se usa para generar u hacer una serie de simulaciones con el modelo de quadrotor que tenemos. Se analizarán tanto sus posibilidades como sus especificaciones y plugins entre otros.

3.2 Plataformas robóticas aéreas.

3.2.1 Erle copter

3.2.1.1 Origen

El origen de este dron parte de una empresa formada por dos hermanos de Álava David y Victor Mayoral Vilches, la startup Erle Robotics en el 2012.

Consiste en un cuadricóptero inteligente basado en el sistema operativo Ubuntu 14 o Ubuntu 16, con soporte oficial para frameworks como ROS. Su elemento más significativo es el Erle-Brain 2, que es módulo electrónico que opera como un pequeño ordenador Linux, el cual tiene ROS preinstalado. Posee todo el software y sensores necesarios para convertir al dron en un dispositivo autónomo.
La empresa fue comprada por otra, acutronic, compañía suiza que dos años más tarde cerró (1, s.f.).

3.2.1.2 Componentes Físicos

En este apartado se expondrá los componentes o elementos básicos que forman el hardware del ErleCopter y sus características: (2, s.f.)

Tabla 3.0.3 - 1:Caracteristicas del dron comercial Erle-copter

Característica	Descripción
Dimensiones	370 x 370 x 95 mm
Peso	878 g (batería incluida)
Carga útil	1 kg
Distancia entre ejes	730 mm
Hélices	10x4.5 o 9.4x4.3
Tiempo de vuelo	~ 20 min con batería de 5000 mAh
Color del frame	Blanco y rojo

Chasis

El chasis o frame, es la estructura principal del dron. Es el soporte principal en el que están unidos los elementos, aporta sujeción y solidez. Este está compuesto por cuatro brazos los cuales tendrán los motores situados en sus extremos, las placas centrales que unen y son la base de la batería, del controlador y de las patas, además de otros.

Figura 3.0.3 - 1: Frame F450

Sus características son:

Característica	Descripción
Color	Rojo / Blanco
Distancia entre ejes	450 mm / 17.7 "
Peso	248 g

Tabla 3.0.3 - 2: Características del frame F450

Motores

Los actuadores existentes en este dron son los motores que modificando solamente la velocidad de giro obtenemos cambiar el empuje. estos motores que hemos usado son de limitada edición de T-Motor, se llaman MN2213. Son motores que no tienen escobillas lo que da lugar a que casi no tengan rozamiento, además de no generar mucha pérdida del calor para que crezca su vida útil.

Figura 3.0.3 - 2: Motores MN2213

Sus características son:

Característica	Descripción
Potencia	950 KV
Eje	3 mm
Peso	58 g
Tamaño estator	22 x 13 mm (diámetro x altura)

Tabla 3.0.3 - 3: Características de los motores MN2213

Como la alimentación de las baterías es de corriente continua, precisaremos de los variadores, que convierten la corriente continua en corriente trifásica que es la que los MN2213 admiten y podemos variarla para regular la velocidad del giro.

Variadores

También llamados ESCs, son circuitos pequeños usados que generan la corriente para controlar la dirección y la velocidad de los motores. También es el componente que envía información a los motores desde el Erle-Brain 2.

Figura 3.0.3 - 3: Variadores simonK30A

Sus características son:

Característica	Descripción
Corriente máxima	30 A
Rango de voltaje	2-4s *LiPo* (desde 7.4 hasta 14.8 V)
Peso	26.5 g (cables y enchufe incluidos)
Tamaño	50 x 25 x 11 mm

Tabla 3.0.3 - 4: Características de los variadores SimonK30A

Hélices

Son las encargadas de transformar la rotación generada por los motores en un movimiento de propulsión para generar un movimiento. Las usadas por este dron son de 9,4x4,3 o de 10x4,5, esto es el tamaño que tienen en pulgadas X ángulo de inclinación, este tamaño es de extremo a extremo.

Figura 3.0.3 - 4: Hélices CW y CWW

Sus características son:

Característica	Descripción
Peso	0.100 Kg
Tipo/Material	Estandar/Plástico

Tabla 3.0.3 - 5: Características de las hélices

Batería

Es la fuente de alimentación principal. Un módulo de potencia conecta la fuente de alimentación al Erle-Brain 2. El dron tiene dos fuentes 3S LiPo (litio y polímero). Estas son ligeras con mucha capacidad y con una elevada tasa de descarga.

Figura 3.0.3 - 5: Batería Floureon 3S LiPo 11,1V v 5500mAh

Entre sus características hay que decir que tienen una tensión de 11,1 voltios y una intensidad de 5500 mAh. La S indica que son fuentes que tienen 3 celdas y una tensión de 3,7 V en cada una, 11,1 V en total.

Soporte de la batería

Elemento que permite fijar la fuente al dron, este soporte nos permite realizar un cambio de bateria fácil, está hecha de plástico.

Figura 3.0.3 - 6: Soporte para la batería

Módulo de potencia

Este módulo también llamado Power Module informa de la tensión e intensidad de la fuente de alimentación y también hace que se pueda alimentar el controlador. Su trabajo es controlar la tensión e intensidad que hay en la salida a 5,3 V y 2,25 A como máximo. También , permite como máximo 90 A y 18 V en la entrada.

Figura 3.0.3 - 7: Módulo de potencia

Sus características son:

Característica	Descripción
Tensión máx entrada	18 V
Corriente máx entrada	90 A
Salidas de conmutación	5.3 V y 2.25 A como máximo

Tabla 3.0.3 - 6: Características del módulo de potencia

Mando radio control

El mando sirve para interactuar con el dron he indicar mediante sus joysticks los movimientos, el empuje, pitch, roll y yaw, que queremos que haga el Erle-Copter. También se puede usar su variedad de interruptores para cambiar los modos de vuelo.

Figura 3.0.3 - 8: Emisora Turnigy TGY-i6

Receptor de señal de radio

Son los encargados de hacer de enlace entre el mando que emite los movimientos y el Erle-Brain 2 que los recibe, estos receptores lo hacen a través de una Modulación por posición de pulsos, es decir, una codificación PPM. Tiene 6 canales y funcionando a una frecuencia de 2,4 GHz.

Figura 3.0.3 - 9: Receptor Turnigy TGY-iA6, de señal de radio

Adaptador Wi-Fi

Módulo sin cables para la medición a distancia que conecta a un acceso WiFi ya existente y hace que se pueda conectar de manera inalámbrica al Erle-Brain 2 desde cualquier punto de acceso como puede ser un ordenador.

Figura 3.0.3 - 10: Edimax EW-7811UAC

Sus características son:

Característica	Descripción
Norma *IEEE*	802.11 ac
Ancho de banda	2.4 GHz
Peso	2 gramos / 0.001 libras

Tabla 3.0.3 - 7: Rasgos del módulo Wi-Fi

Apoyos

Apoyos hechos de fibra de vidrio, son bastante livianos y no generan variación en el peso del dron, amortiguan el golpe al realizar el aterrizaje.

Figura 3.0.3 - 11: Patas de fibra de vidrio

Sus características son:

Característica	Descripción
Material	Fibra de vidrio
Altura	200 mm
Anchura entre piernas	325 mm

Tabla 3.0.3 - 8: Caract. de las patas fiberglass de Erle-copter

3.2.1.3 Sensores de posición

En este apartado se hablará sobre los sensores que usa el Erle-Copter. Veremos en profundidad el GPS y el sensor de la Cámara. Otros tipos de los que hace uso el dron pueden ser el sensor de aceleración o el giroscopio que están contenidos en Erle-Brain 2.

Conjunto GPS y brújula

El Sistema de Posicionamiento Glogal o GPS es un modo de orientación vía satélite el cual ayuda a conseguir la posición del dron. El modelo que incorpora el dron de Erle Robotics es un GPS uBlox Neo-8M. Tanto el GPS como la brújula digital están dentro de un armazón para estar situados en una buena posición en el dron y así estar aislado de las interferencias con el Erle-Brain 2.

Figura 3.0.4 - 1: Encapsulado del Erle Brain 2- GPS

El controlador está acoplado a la brújula digital mediante un puerto I^2C a través de un cable de tipo DF13 con 4 pines, en cambio el controlador y el GPS están conectados con un puerto de tipo UART.

Sensor de la cámara o de flujo óptico

Este es un tipo de cámara que está enfocando hacia el suelo, además, está unida al giroscopio de 3 ejes, usa la variedad de texturas que podemos encontrar en el suelo, también utilizara las propiedades que estén a la vista para obtener la velocidad que tiene una nave aérea. Este sensor que se ha usado en este dron se llama PX4Flow, ha sido desarrollado por la compañía Pixhawk y usaremos la versión 1.3.

Figura 3.0.4 - 2: Sensor de Cámara PX4Flow v1.3

El sensor tiene la capacidad de añadir un SONAR para completar los datos que se han obtenido por este. Algunas propiedades de este son,

Tabla 3.0.4 - 1: Caract. del Sensor de flujo óptico PX4Flow

Característica	Descripción
Resolución	752 x 480 píxeles
Frecuencia de procesamiento	400 Hz
CPU	*Cortex M4F* de 168 MHz
Longitud de la lente	16 mm
Consumo de energía	115 mA / 5 V
Tamaño	45.5 mm x 35 mm

El GPS usara los datos del PX4Flow con el fin de completar la información necesaria para hacer vuelos en zonas de exterior como de interior. para su incorporación en el Erle-Copter, tenemos que asegurar que los ejes X del Erle-Brain 2 y el eje Y del sensor tengan idéntica posición para eliminar posibles datos erróneos. Si esto no sucede hay que cambiar los parámetros internos para considerar dicho cambio de orientación.

Figura 3.0.4 - 3: La adecuada orientación del sensor de flujo óptico

Para finalizar, conectamos el sensor, mediante el otro puerto I^2C, al controlador Erle-Brain 2, ya que anteriormente hemos usado un puerto I^2C para conectar la brújula digital.

Figura 3.0.4 - 4:Conexión del Erle-Brain 2 al sensor de óptico PX4Flow

3.2.1.4 Erle- Brain 2

El controlador es la segunda generación de un sistema que incluye todo el hardware y software necesario para controlar un dron de forma eficiente. Es compatible con ROS, un sistema operativo que está basado en una arquitectura Linux, se usa principalmente para la fabricación de aviones no tripulados (UAV) y robots. Tiene soporte oficial para ROS, este tiene una gran compatibilidad con multitud de vehículos. También es un software que se utiliza como piloto automático, ArduPilot, para controlar el dron. Su estructura tiene una composición formada por una PCB denominada PXFmini conectada a una Raspberry Pi 2.

La Raspberry Pi tiene una tarjeta PXmini que es un shield con código abierto, que sirve para generar un piloto automático que esté listo para volar. Está esbozado esencialmente para el modelo Zero, sin embargo, es apto para los tipos 1, 2 y 3.

También optimiza la conexión a través de la colocación intuitiva y metódica de las conexiones y pines, el controlador también tiene los diversos componentes internos que usa para controlar el cuadricóptero. Los cuales corresponden con:

- Sensor de gravedad de 3 ejes.
- Giroscopio de 3 ejes.
- Brújula digital de 3 ejes.
- Sensor de presión barométrico.
- Sensor de temperatura.
- ADC para la detección de la batería.

El tipo de la IMU es MPU9250 IMU y el del barómetro es MS5611. Tiene también tres luces de tipo LEDs (azul, verde y naranja) indicadores, estas forman un código con colores que informa del estado, además de otras cosas como pueden ser, el indicar si el piloto automático esta apagado. A continuación, podemos ver dos ilustraciones en las que se pueden ver los diferentes tipos de conexiones que podemos encontrar en la placa, además de su conexión a la Raspberry Pi 2.

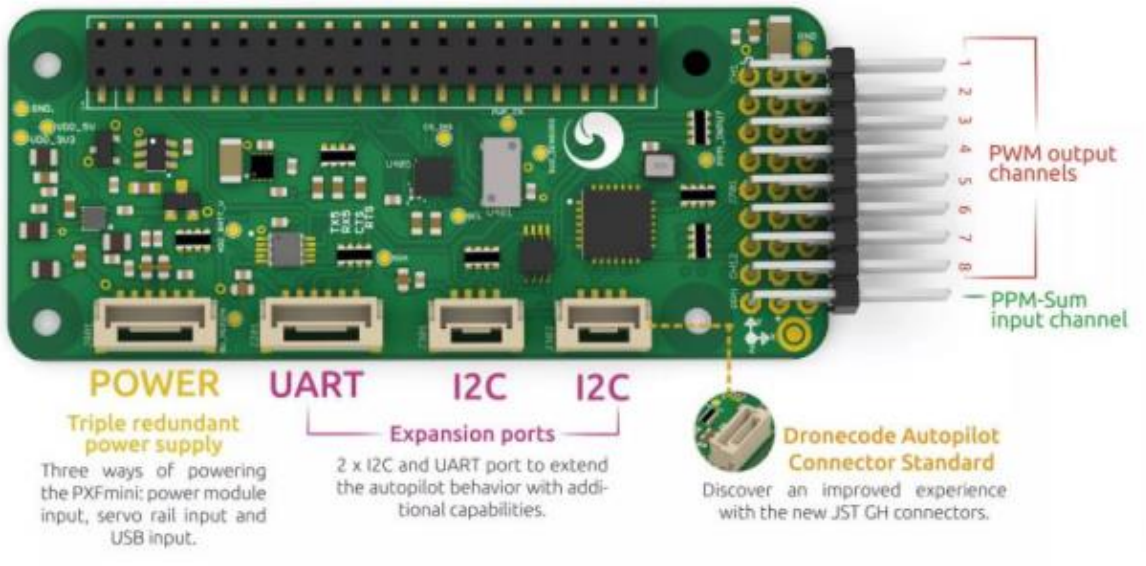


Figura 3.0.5 - 1: Conexiones presentes en la placa PXFmini

Figura 3.0.5 - 2: Unión de la Raspberry Pi 2 a la placa PXFmini

Tabla 3.0.5 - 1: Dimensiones de la placa PXFmini

Característica	Descripción
Peso	15 gramos / 0.033 libras
Tamaño	31 x 73 cm

Conectividad

Esta formado fundamentalmente por la Raspberry Pi 2, podemos encontrar diferentes formas de conexión como los puertos USB, I^2C, UART, o Ethernet. También, tenemos una PiCamera con 5 Mega Pixels para conseguir una imagen en vivo. En la ilustración que hay a continuación podemos encontrar los diversos pines y puertos con los que vamos a tener en el controlador Erle-Brain 2.

Figura 3.0.5 - 3: Conexiones pertenecientes al controlador Erle-Brain 2

Configuración

Como tenemos la Raspberry Pi 2 es menester agregar en la está un sistema que esté fundamentado en un sistema Linux mediante una tarjeta de memoria SD. Cuando ya lo tengamos preparado, conectaremos el Erle-Brain 2 a Internet ya que es necesario de proveerle de la fisionomía de un dron.
Las especificaciones técnicas del controlador:

Tabla 3.0.5 - 2 Especificaciones técnicas de Erle-Brain 2

Característica	Descripción
Dimensiones	63 x 96 x 25 mm
Peso	100 g
Procesador	*ARM Cortex-A7 CPU*, 4 nucleos a 900 MHz
Memoria RAM	1 GB
Cámara	5 MP
E/S	2x*I2C*, *UART*, conexión *POWER*, 4 puertos *USB*
	Puerto *HDMI*, *Ethernet* y conector de audio de 3.5 mm

3.2.1.5 Soporte del Producto

Llegado a este punto se optó por cambiar de dron debido a que, a la hora de obtener las actualizaciones, los modelos para gazebo y el resto de la información ha sido imposible, ya que estas no están disponibles en ningún sitio, incluso se contactó de forma directa con los desarrolladores del producto, uno de los hermanos mayoral Vilches, los cuales no ofrecieron ninguna solución alegando que no tenía nada que pudiese ayudar a realizar el TFG.
Por este motivo se optó por reemplazar este drone por el AR. Drone 2.0 de Parrot.

3.2.2 AR Dron 2.0 de Parrot

3.2.1.1 Origen

Con sede en parís podemos considerar que Parrot, que fue fundada en 1994, actualmente, es entre todas las industrias que se dedican a la fabricación de dispositivos Wireless el líder a nivel mundial, es una empresa exportadora ya que el 85% de sus ventas son en el extranjero y además cuenta con un gran numero de colaboradores, 450 para ser exactos. En 2006 esta gran empresa empezó a cotizar en bolsa en la sede de parís llamada NYSE Euronext.

Esto ha sucedido gracias a la evolución de la red de telefonía a nivel mundial que ha dado lugar a que se puedan desarrollar todo tipos de productos para el uso cotidiano, ya puedan desde unos auriculares inalámbricos hasta unas manos libres para el coche. Aunque no es su especialidad ha diversificado en otros productos más específicos del mundo del sonido y la imagen.

Una de las ramas de producción mas atractivas y de mayor rendimiento hoy en día es su gama de desarrollo de drones, los cuales son fáciles de montar y manejar a cualquier rango de edad, estos son de gran calidad. Hay que destacar que en 2010 saco al mercado uno de sus productos que a la larga le ha dado muchos beneficios a la empresa, dicho producto fue el cuadricóptero al que denominaron AR. Dron, que es un dron teledirigido por Wi-Fi y de gran eficiencia, tiene la capacidad de generar una red propia para conectar el mando inalámbrico.
Con su gran capacidad de generar ingresos esta empresa también opto por desarrollar app para el manejo del AR dron a través de cualquier dispositivo que tenga un sistema operativo de Apple o Android. gracias a todos estos logros en 2010 este cuadricóptero de Parrot fue galardonado mediante el premio CES por su innovación para el hardware de juegos electrónicos (7, s.f.).

3.2.1.2 Características

El Parrot AR Dron 2.0 es la segunda versión de un dron perfecto para aquellos que se quieran iniciar en el pilotaje de aeronaves con altas prestaciones de imagen y vuelo, en este cuadricoptero han añadido un sensor de presión para mejorar el control de altitud.
El sensor de presión actúa como un altímetro. Ahora se obtiene la presión del aire durante el vuelo y se compara con la presión del suelo.
También cuenta con un nuevo algoritmo más inteligente para interpretar la medición ultrasónica cuando el dron se cierne sobre un suelo inestable como arbustos (8, s.f.).

Figura 3.1.2-1: especificaciones técnicas

	AR.DRONE	AR.DRONE 2
		
Tipo de Batería / Duración	Polímero de Litio / 12 minutos	Polímero de Litio / 12 minutos
Rango de Control	50 metros	50 metros
Cámara Frontal	Cámara VGA (640 x 480)	Cámara HD (1280 x 720)
Tecnología WIFI	802.11 b/g	802.11 b/g/n <– Más distancia de control.
Software AR.FreeFlight 2.0 App	Compatible	Diseñado para el AR.Drone 2.0
Compartir vídeo en Youtube, Picasa o guardar en la memoria del dispositivo	Resolución VGA	Resolución VGA
Grabación de Vídeos	Resolución VGA	Resolución VGA
Sensor de Control Absoluto	No	Sí <– Pilotaje más fácil, adoptando el AR.Drone la misma posición que el dispositivo de control.
Sensor de Estabilidad a gran Altitud	No	Sí
Habilidad Looping	No	Sí

¡NUEVA! **AR.Drone Academy.** Página web para compartir los vídeos grabados desde el AR.Drone. Más aplicaciones profesionales: Grabaciones en altura de gran calidad HD (ingenierías, arquitecturas, construcción, media, eventos, seguridad...)

Figura 3.1.2 - 2: comparación del AR dron

3.2.1.3 Componentes

Cuando se adquiere el producto de la colección de Parrot AR. Dron 2.0, el Cuadricóptero controlado por Wi-Fi, se presenta en su embalaje con el contenido siguiente :

- Carcasa de exterior.
- Carcasa de interior.
- juego de aspas.
- otro juego de hélices personalizadas.
- batería de litio de 1000mAh,con 12 minutos de autonomía.
- Cargador con varios adaptadores.

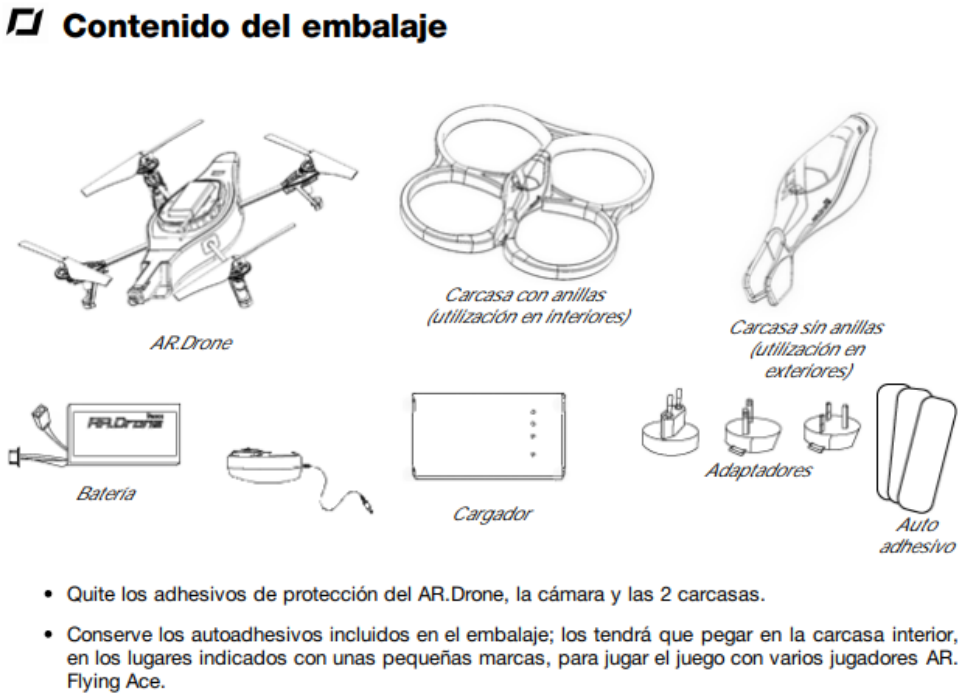


Figura 3.1.3 - 1: Contenido embalaje

3.2.3 Control mediante bibliotecas o librerías software

En cuanto al desarrollo de software propio para controlar el drone, Parrot ofrece un SDK que podría ser utilizado. Sin embargo, al existir un driver (conjunto de paquetes) estable para ROS, se ha optado por trabajar con este middleware, ya que ofrecen grandes ventajas trabajar con este ecosistema para el desarrollo de aplicaciones robóticas, entre las que se pueden destacar la abstracción del hardware, facilidad de intercambio de información, existencia de un simulador, facilidad de ser controlado de diferentes formas, etc.

ROS permite trabajar con herramientas y librerías, los cuales podemos activar mediante unos topic, que puede interpretar las órdenes recibidas y enviarlas al Dron, y a la inversa, obtener información del dron y mostrárnoslas. y con ello controlar nuestro dron. Además, se puede integrar ROS con librerías existentes. Esto da lugar una gran flexibilidad, la cual puede ser necesaria ya que ROS posee un grupo de usuarios dando lugar a que sea fácil obtener e intercambiar numerosas aplicaciones y librerías de código abierto.

3.3 ROS

3.3.1 ¿Qué es ROS?

Como ya hemos comentado al inicio del proyecto, cuando presentamos ROS, solo cabe recordar que Robot Operating System (ROS) corresponde a un middleware robótico desarrollado en 2007 bajo el nombre de switchyard por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto del Robot (STAIR2). Desde 2008, el desarrollo continuó principalmente en Willow Garage, un instituto de investigación robótico con más de veinte instituciones colaborando en un modelo de desarrollo federado (5, s.f.).

El sistema operativo para robots es un framework el cual se usa para la escritura de software utilizados en robots. Está formado por un conjunto de librerías y herramientas admitidas en una gran diversidad de sistemas de robots. ROS no debe de ser considerado un sistema operativo. Sin embargo, ofrece diversas funciones como pueden ser entre otras la comunicación entre procesos, la abstracción del hardware, control de dispositivos de bajo nivel, también tiene funciones asíncronas y síncronas que son de alto nivel, y también tiene otras funciones otras como pueden ser tener una base de datos centralizada (5, s.f.).

ROS se basa en dos aspectos: el primero es la correspondiente al sistema operativo, ros, y el segundo es el ros-pkg que reside en un conjunto de paquetes que los usuarios generan que efectúan funcionalidades como pueden ser la localización o el mapeo simultáneo, etc (7, s.f.).

Existen diferentes versiones de ROS que se tienen que instalar sobre una versión específica de un sistema operativo tipo UNIX. En este trabajo se ha instalado la versión de ROS Kinetic sobre la versión de escritorio de Ubuntu 16.

3.3.2 Tutoriales

A nivel de documentación es muy recomendable estudiar algunos de los tutoriales que ofrece ROS, ya que es la mejor forma de aprender a manejar este entorno para desarrollo de aplicaciones robóticas (9, s.f.).

1. Core ROS Tutorials

1.1 Beginner Level

1. Installing and Configuring Your ROS Environment

   This tutorial walks you through installing ROS and setting up the ROS environment on your computer.
2. Navigating the ROS Filesystem

   This tutorial introduces ROS filesystem concepts, and covers using the roscd, rosls, and rospack commandline tools.
3. Creating a ROS Package

   This tutorial covers using roscreate-pkg or catkin to create a new package, and rospack to list package dependencies.
4. Building a ROS Package

   This tutorial covers the toolchain to build a package.
5. Understanding ROS Nodes

   This tutorial introduces ROS graph concepts and discusses the use of roscore, rosnode, and rosrun commandline tools.
6. Understanding ROS Topics

   This tutorial introduces ROS topics as well as using the rostopic and rqt_plot commandline tools.
7. Understanding ROS Services and Parameters

   This tutorial introduces ROS services, and parameters as well as using the rosservice and rosparam commandline tools.
8. Using rqt_console and roslaunch

   This tutorial introduces ROS using rqt_console and rqt_logger_level for debugging and roslaunch for starting many nodes at once. If you use `ROS fuerte` or ealier distros where rqt isn't fully available, please see this page with this page that uses old `rx` based tools.
9. Using rosed to edit files in ROS

   This tutorial shows how to use rosed to make editing easier.
10. Creating a ROS msg and srv

    This tutorial covers how to create and build msg and srv files as well as the rosmsg, rossrv and roscp commandline

Figura 3.3.2 - 1: Listado de tutoriales de ROS

Crear un espacio de trabajo ROS

Un primer paso para poder desarrollar un proyecto con ROS es configurar adecuadamente el espacio de trabajo, también conocido como Workspace. Para ello, el primer paso es crear dicho directorio y ejecutar el siguiente comando en el mismo: Estas instrucciones son para ROS Groovy y versiones posteriores.

Creemos y construyamos un espacio de trabajo de catkin :

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

El comando catkin_make es una herramienta práctica para trabajar con espacios de trabajo catkin . Al ejecutarlo por primera vez en su espacio de trabajo, creará un enlace CMakeLists.txt en su carpeta 'src'.

Además, si busca en su directorio actual, ahora debería tener una carpeta 'build' y 'devel'. Dentro de la carpeta 'devel' puede ver que ahora hay varios archivos setup.*Sh.

Otra configuración importante de ROS es la asociada a la ventana de comandos o terminal. En concreto, para poder ejecutar comandos de ROS desde cualquier ruta es importante seguir las indicaciones de la documentación y utilizar el siguiente comando:

Antes de continuar con la fuente de su nuevo archivo setup. * Sh:

```
$ source devel/setup.bash
```

Para asegurarse de que su espacio de trabajo esté correctamente configurado por el script asociado, hay que asegurar de que la variable de entorno ROS_PACKAGE_PATH incluye el directorio en el que se encuentra.

```
$ echo $ROS_PACKAGE_PATH
/home/youruser/catkin_ws/src:/opt/ros/kinetic/share
```

Creación de catkin Package

Cualquier desarrollo hecho con ROS está basado en paquetes, de modo que muy importante conocer dicho proceso. El primer paso para crear un paquete es cambiar al directorio /src dentro del workspace en el que se esté trabajando:

Primero cambie al directorio del espacio de origen del espacio de trabajo catkin que creó en el tutorial Creación de un espacio de trabajo para catkin :

```
# Debería haber creado esto en el Tutorial de creación de un espacio de trabajo
$ cd ~/catkin_ws/src
```

El siguiente paso es usar el comando catkin_create_pkg para crear el paquete:

'beginner_tutorials' que depende de std_msgs, roscpp y rospy:

```
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

Esto creará una carpeta que contiene un package.xml y un CMakeLists.txt , que se han completado parcialmente con la información que le dio a catkin_create_pkg .

En concreto, en el caso anterior se ha creado el paquete con el nombre x y que tiene como dependencias x, y y Z. De forma general la creación de un paquete tiene la siguiente sintaxis:

```
# Este es un ejemplo, no intente ejecutar este
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

catkin_create_pkg también tiene funcionalidades más avanzadas que se describen en catkin / commands / catkin_create_pkg.

A continuación, seguimos con la construcción de un espacio de trabajo catkin y obteniendo el archivo de instalación.
Para ello se necesita construir los paquetes en el espacio de trabajo catkin:

```
$ cd ~/catkin_ws
$ catkin_make
```

Una vez que se ha creado el espacio de trabajo, se ha creado una estructura similar en la subcarpeta `devel` a la que suele encontrar en `/opt/ros/$ROSDISTRO_NAME`.

Para agregar el espacio de trabajo a su entorno ROS, debe obtener el archivo de configuración generado:

```
$ . ~/catkin_ws/devel/setup.bash
```

Para seguir con la construcción del paquete debemos seguir los siguientes pasos.

Si está utilizando esta página para crear su propio código, también eche un vistazo a los tutoriales posteriores (C ++) / (Python), ya que es posible que deba modificar CMakeLists.txt .

Ya debería tener un espacio de trabajo de catkin y un nuevo paquete de catkin llamado beginner_tutorials del tutorial anterior, Creación de un paquete . Vaya al espacio de trabajo de catkin si aún no está allí y busque en la carpeta src:

```
$ cd ~/catkin_ws/
$ ls src
```

```
beginner_tutorials/  CMakeLists.txt@
```

Debería ver que hay una carpeta llamada beginner_tutorials que creó con catkin_create_pkg en el tutorial anterior. Ahora podemos construir ese paquete usando catkin_make:

```
$ catkin_make
```

Debería ver una gran cantidad de resultados de cmake y luego make, que debería ser similar a esto:

```
Base path: /home/user/catkin_ws
Source space: /home/user/catkin_ws/src
...
...
-- Generating done
-- Build files have been written to: /home/user/catkin_ws/build
####
#### Running command: "make -j4" in "/home/user/catkin_ws/build"
####
```

Tenga en cuenta que catkin_make primero muestra qué rutas está usando para cada uno de los 'espacios'. Los espacios se describen en el REP128 y en la documentación sobre espacios de trabajo catkin en el wiki: catkin / workspaces .

Lo importante para tener en cuenta es que debido a estos valores predeterminados se han creado varias carpetas en su espacio de trabajo de catkin. Eche un vistazo con ls:

```
$ ls
```

```
build
devel
src
```

La carpeta de compilación es la ubicación predeterminada del espacio de compilación y es donde se llama a cmake y make para configurar y compilar sus paquetes. La carpeta devel es la ubicación predeterminada del espacio devel , que es donde van sus ejecutables y bibliotecas antes de instalar sus paquetes.
Ahora que ha creado su paquete ROS, hablemos más sobre los nodos ROS.

Comprensión de los nodos ROS, Usando rosrun

Un nodo realmente es un ejecutable en un paquete de ROS. Estos nodos usan una biblioteca para relacionarse con otros. Los nodos son capaces de publicar o suscribirse a un topic y proporcionar o utilizar un Servicio.
Una vez creado un paquete, para ejecutar el mismo dentro del ecosistema de ros es necesario usar el comando rosrun, que tiene como parámetros tanto el nombre del paquete como el nombre del nodo.
`rosrun` le concede utilizar el nombre del paquete para ejecutar un nodo dentro de un paquete (sin tener que saber la ruta del paquete).
Utilización:

```
$ rosrun [package_name] [node_name]
```

Así, a modo de ejemplo se puede ejecutar el paquete turtlesim dentro del nodo turtlesim_node de la siguiente manera:

Luego, en una **nueva terminal**:

```
$ rosrun turtlesim turtlesim_node
```

Verá la ventana de turtlesim:

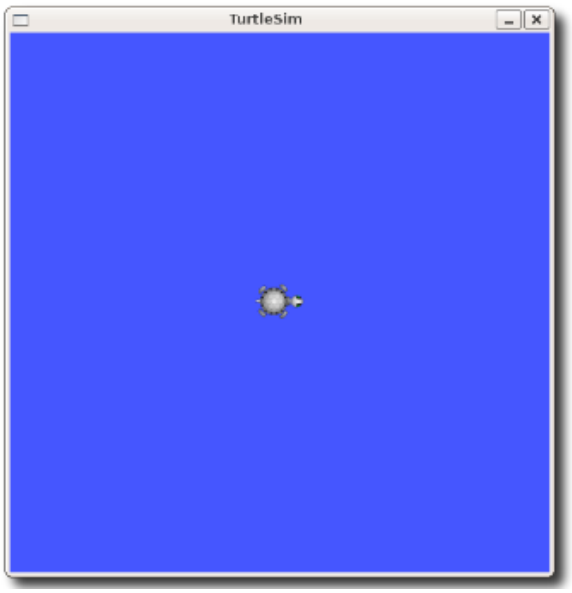


Figura 3.3.2 - 2: Turtle

En una **nueva terminal**:

```
$ rosnode list
```

La ejecución del paquete anterior dará lugar a que aparezcan topics asociados al nodo dentro del ecosistema de ROS (/turtlsim en este caso).

Verá algo similar a:

```
/rosout
/turtlesim
```

Vemos nuestro nuevo nodo / my_turtl . Usemos otro comando rosnode, ping , para probar que está activo:

```
$ rosnode ping my_turtle
```

```
rosnode: node is [/my_turtle]
pinging /my_turtle with a timeout of 3.0s
xmlrpc reply from http://aqy:42235/     time=1.152992ms
xmlrpc reply from http://aqy:42235/     time=1.120090ms
xmlrpc reply from http://aqy:42235/     time=1.700878ms
xmlrpc reply from http://aqy:42235/     time=1.127958ms
```

Lista de algunos tutoriales

A continuación, se pueden ver algunos ejemplos y explicamos en qué consisten algunos de ellos, que han sido utilizados en este trabajo:

1. Instalación y configuración de su entorno ROS

   Este tutorial lo guía a través de la instalación de ROS y la configuración del entorno de ROS en su PC.

2. Navegando por el sistema de archivos ROS

   Este tutorial presenta los conceptos del sistema de archivos ROS y cubre el uso de las herramientas de línea de comandos roscd, rosls y rospack .

3. Crear un paquete ROS

   Este tutorial cubre el uso de roscreate-pkg o catkin para crear un nuevo paquete y rospack para enumerar las dependencias del paquete.

4. Construyendo un paquete ROS

   Este tutorial cubre la cadena de herramientas para crear un paquete.

5. Comprensión de los nodos ROS

   Este tutorial presenta los conceptos de gráficos ROS y analiza el uso de las herramientas de línea de comandos roscore , rosnode y rosrun .

6. Comprensión de los temas de ROS

   Este tutorial presenta topic de ROS, así como el uso de las herramientas de línea de comandos rostopic y rqt_plot .

7. Comprensión de los servicios y parámetros de ROS

   Este tutorial presenta los servicios y parámetros de ROS, así como el uso de las herramientas de línea de comandos rosservice y rosparam .

8. Usando rqt_console y roslaunch

   Este tutorial presenta ROS usando rqt_console y rqt_logger_level para depurar y roslaunch para iniciar muchos nodos a la vez. Si usa ROS fuerte o ealier distribuciones donde rqt no está completamente disponible, consulte esta página con esta página que usa herramientas antiguas basadas en rx .

9. Usando rosed para editar archivos en ROS

   Este tutorial muestra cómo utilizar rosed para facilitar la edición

3.4 GAZEBO

3.4.1 ¿Qué es Gazebo?

Gazebo, que nace con el nombre de Gazebo Project, es un simulador 3D, cinemático, dinámico y multi-robot con la capacidad de simular de forma precisa y eficiente poblaciones de robots en entornos complejos de interior y exterior. Aunque es similar a los motores de juegos, Gazebo ofrece simulación física con un grado de fidelidad mucho más alto, un conjunto de sensores e interfaces tanto para usuarios como para programas que permite realizar simulaciones de robots articulados en entornos complejos, interiores o exteriores, realistas y tridimensionales (6, s.f.).
Gazebo es un software libre financiado, en parte, por Willow Garage, pudiendo ser reconfigurado, ampliado y modificado Compatible con ROS y Player.
Podemos ejecutar Gazebo desde ROS y utilizar las APIs de este último para controlar los robots en las simulaciones, es decir, enviar y recibir datos de éstas (6, s.f.).
Los usos típicos en los cuales suele usarse Gazebo pueden ser probar algoritmos de robótica, diseñar robots y realizar pruebas de regresión con escenarios realistas (6, s.f.).

Algunas características clave de Gazebo incluyen:

- Múltiples motores de física
- Una rica biblioteca de modelos y entornos de robots
- Una amplia variedad de sensores
- Convenientes interfaces programáticas y gráficas
- Simulación realista de la física de los cuerpos rígidos.
- Los robots pueden interactuar con el mundo.
- Capacidad de desarrollar y simular modelos propios.
- Posibilidad de crear escenarios de simulación.
- Contiene plugins para añadir sensores al modelo del robot y simularlos.

A nivel de requisitos del sistema, Gazebo es muy compatible con distribuciones Ubuntu y requiere de un PC con las siguientes prestaciones:

- Una GPU dedicada
    - Las tarjetas Nvidia tienden a funcionar bien en Ubuntu
- Una CPU que sea al menos Intel I5, o equivalente
- Al menos 500 MB de espacio libre en disco
- Ubuntu Trusty o posterior instalado.

Internamente, Gazebo hace uso de las siguientes prestaciones (6, s.f.).

- Open Dynamics Engine (ODE), un motor de física basada en la formulación de problemas sobre la complementariedad de restricción (LCP).
- OGRE, motor de renderizado de escenas gráficas en 3D.

3.4.2 Trabajar con Gazebo

Algunos comandos asociados al trabajo con Gazebo son los siguientes:

Lanzamiento de Gazebo

Configurar las variables de entorno de ros (10, s.f.):

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

Iniciar una simulación de mundo vacío:

```
roslaunch gazebo_worlds empty_world.launch
```

El archivo de inicio del ar dron en Gazebo se inicia con:

```
roslaunch cvg_sim_gazebo ardrone_testworld.launch
```

Esto debería iniciar el simulador y abrir una ventana GUI que se vería así:

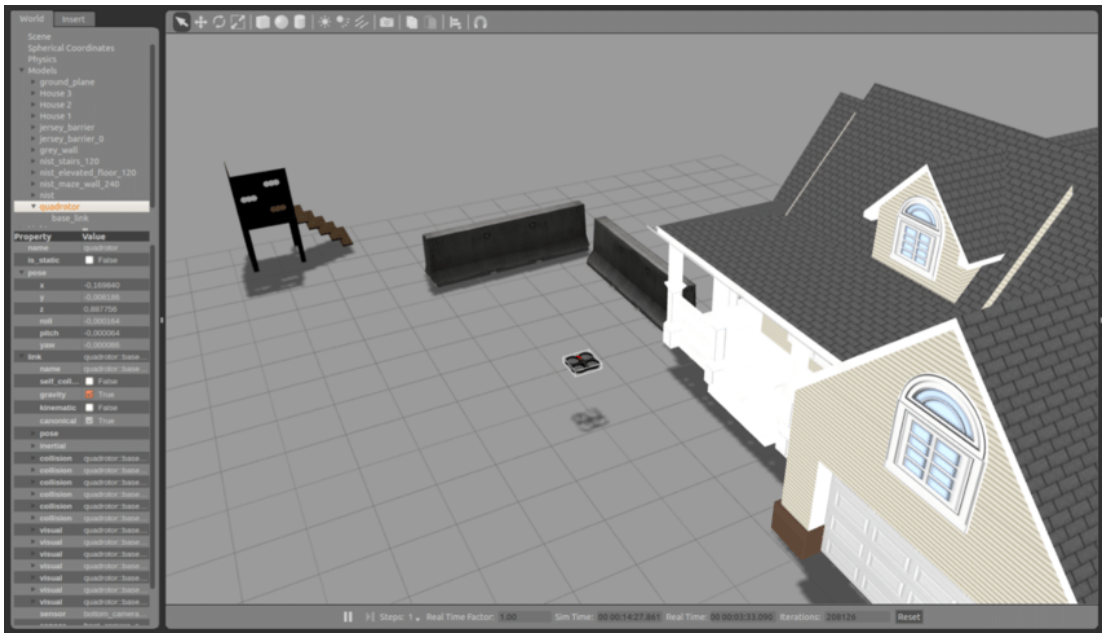


Figura 3.4.2 - 1: AR drone

Diseño del entorno

Una vez que hemos iniciado la herramienta Gazebo mediante roslaunch y ejecutando él .launch que se encontrará en home/catkin_ws/src.
Al ejecutarlo, podremos seleccionar en la esquina superior izquierda el botón “Edit” donde podremos seleccionar la opción de editar un nuevo mundo.
En esta nueva ventana, podremos colocar donde introducir los distintos elementos como muros, puertas o ventanas.
Utilizamos las distintas herramientas para finalmente formar nuestro entorno Y, aunque esto sea opcional, puedes añadir veracidad y profundidad al diseño dándole un material a los muros del mundo.

Cabe destacar que como podemos obtener modelos o diseños en 3D de internet o de la base de datos de gazebo e incluirlos en nuestro proyecto esta es la forma que más he usado para realizar mi entorno de tal manera que hemos podido aumentar la complejidad de nuestro entorno. Esto se hace en la pestaña en la que pone “insert”, en la base de datos antes mencionada podemos seleccionar el objeto 3D que deseamos y colocarlo en nuestro entorno en el lugar y situación en la cual haga la función que buscamos.

Podemos guardar nuestro mundo mediante la opción “Save As” del menú. Para que funcione correctamente, se pide guardarlo en la pestaña que queremos de nuestro ordenador. Para poder abrirlo, en el propio programa podremos ejecutarlo mediante la barra que se encuentra en la izquierda, donde debería aparecer el nombre de nuestro diseño.

A pesar de la opción anterior, existe una manera de cargar directamente el mundo en el programa, de tal manera que nada más ejecutarlo estén en la pantalla todos los obstáculos y todos los objetos en la posición predefinida. Para ello, en el paquete creado deberemos entrar en la carpeta correspondiente y crear el archivo “launch” introduciendo en el nombre de nuestro mundo creado.

Para los paquetes que he hecho yo he creado un mundo personalizado y para iniciarlo debemos de usar:

```
roslaunch cvg_sim_gazebo mission.launch
```

3.4.4 Los paquetes de ROS

Una vez que nos hemos introducido en ROS y en Gazebo puedo comentar que existen varios comandos de ros para manipular el dron en dicho entorno. Estos comandos son los pertenecientes a un paquete de ros llamado ardrone_autonomy, el cual debe de haber sido descargado previamente y tiene que formar parte de nuestro sistema operativo. dichos comandos los podemos ver usando:

```
rostopic list
```

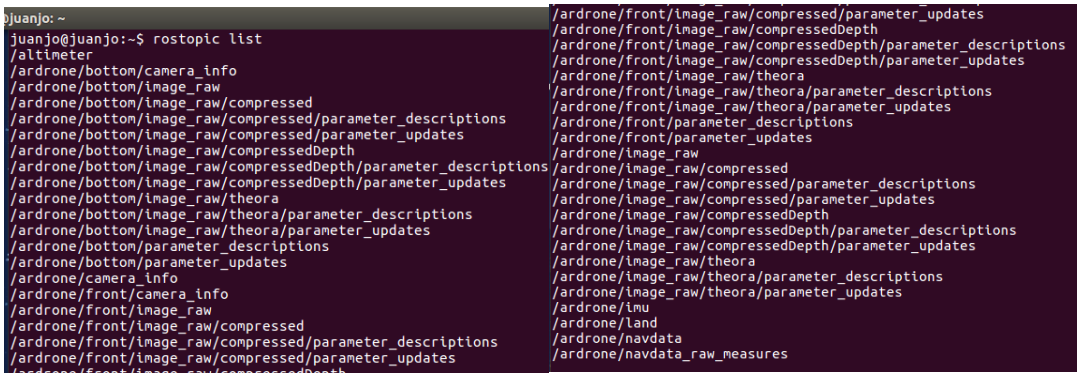


Figura 3.4.4 - 2: Rostopic List

Y con esta lista podemos usarla de la siguiente forma para mover el dron o para iniciar la cámara, por ejemplo.

Para que levante el vuelo usamos el take off:

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

Para que aterrice el dron usamos el land:

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

Sí abrimos otra pestaña del terminal, podemos usar el siguiente comando para la velocidad:

```
rostopic pub /cmd_vel geometry_msgs/Twist "linear:
```

En el caso de abrir la cámara usamos:

```
rosrun image_view image_view image:=/ardrone/image_raw
```

Los comandos que se han descrito anteriormente permiten manipular el drone de forma manual. Sin embargo, se pueden crear procesos para realizar dichas tareas de forma automática en función de ciertas entradas y/o condiciones. Dicha forma de trabajo automática se describirá en el capítulo siguiente.

3.5 Conclusiones

En este capítulo se ha descrito en detalle el cuadricóptero con el que se ha desarrollado el TFG, así como los conceptos necesarios del ecosistema ROS para trabajar con dicho drone en el simular Gazebo.
En el siguiente capítulo se describirán todos los componentes software desarrollados para poder utilizar el drone mencionado sobre un caso de estudio agronómico que se ha diseñado a medida para ejecutarlo sobre Gazebo.

Igualmente hemos podido ver como es el uso de los paquetes de ROS, que tenemos una gran diversidad a nuestro alcance y que dependiendo de los paquetes que tengamos podemos manejar cada dron usando unos topic distintos.

Para concluir podemos decir que el trabajo realizado hasta este punto ha tenido una serie de problemas ya resueltos, la mayoría de estos han estado relacionados a la obsolescencia de las herramientas utilizadas. Por todo esto hemos aprendido que tanto el sistema operativo como ros y demás herramientas utilizadas son de gran maleabilidad, tienen una gran adaptación y variedad de opciones, las cuales la mayoría la hemos llevado a la práctica .

Capítulo IV

Descripción del trabajo realizado

4.1 introducción

En este capítulo hablaremos sobre los paquetes que he hecho para los movimientos del dron. Nos centraremos en la creación, el funcionamiento y programación, de estos, para moverlo y realizar una tarea centrada en la agricultura. Se explicarán el flujograma, programación y otras opciones de las que dispone.

4.2 Creación del workspace y de paquetes

Para crear un paquete lo primero que tenemos que hacer es ir a la carpeta correspondiente que en nuestro caso es src dentro de la carpeta catkin_ws (11, s.f.).

```
Cd catkin_ws/src
```

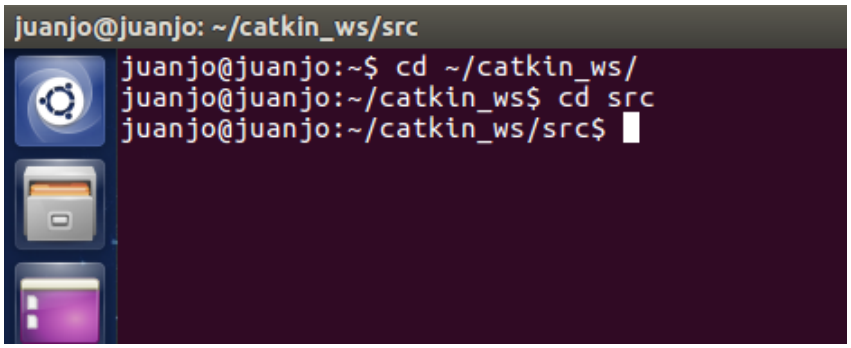


Figura 4.2 - 1: accediendo a la ruta de creación del paquete

Una vez dentro, ya podemos proceder con la creación de nuestro paquete, tecleando el siguiente comando:

```
Catkin_create_pkg nombre roscpp geometry_msgs
```

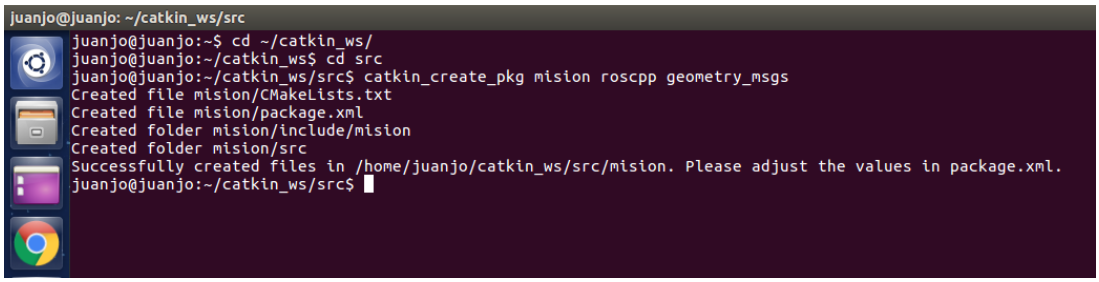


Figura 4.2 - 2: creación del paquete

Hemos de tener en cuenta que en el lugar en el cual pone "nombre" debemos de sustituirlo por la denominación que queramos darle a nuestro paquete, en nuestro caso usaremos misión o codeardrone, como veremos más adelante.
Roscpp y geometry_msgs son las distintas dependencias que vamos a incluir, hemos puesto roscpp porque el lenguaje que vamos a usar es c++ y geometry_msgs porque vamos a enviar mensajes de velocidad.

Comprobamos que lo hemos hecho bien, entrando en el paquete y viendo los archivos que lo componen, mediante los siguientes comandos.

```
Cd nombre

ls
```

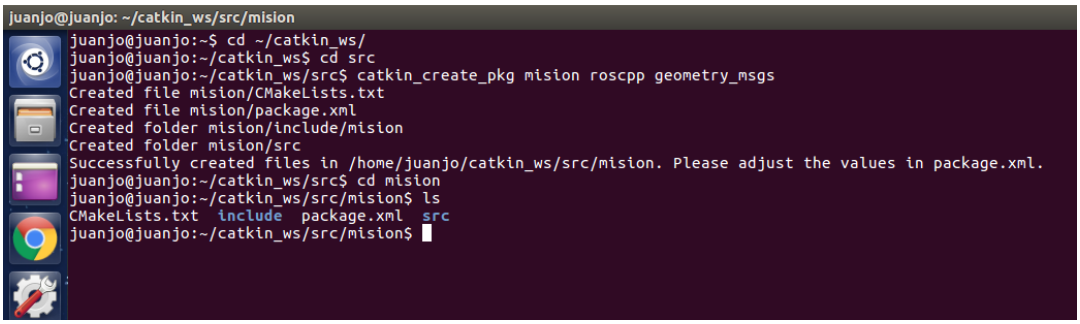


Figura 4.2 - 3: verificación de la correcta creación del paquete

Como podemos observar en la imagen anterior están todos los archivos y carpetas que vamos a necesitar, por tanto, podemos continuar con el proceso de desarrollo de la creación del paquete, procediendo como veremos a continuación.

Antes de continuar vamos a comprobar los archivos, como observamos las carpetas situadas en catkin_ws dentro del directorio s,rc podemos ver las carpetas creadas mediante esta forma y en las cuales están situados los archivos donde vamos a realizar la programación.

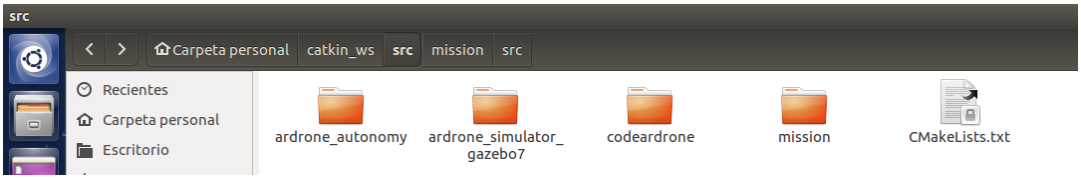


Figura 4.2 - 4: visualización gráfica del contenido del paquete

A continuación, para seguir con el proceso debemos hacer una serie de modificaciones en el archivo denominado como Cmakelist.txt, de la carpeta llamada misión que hemos creado anteriormente, como podemos observar el archivo originalmente es el de la siguiente imagen.

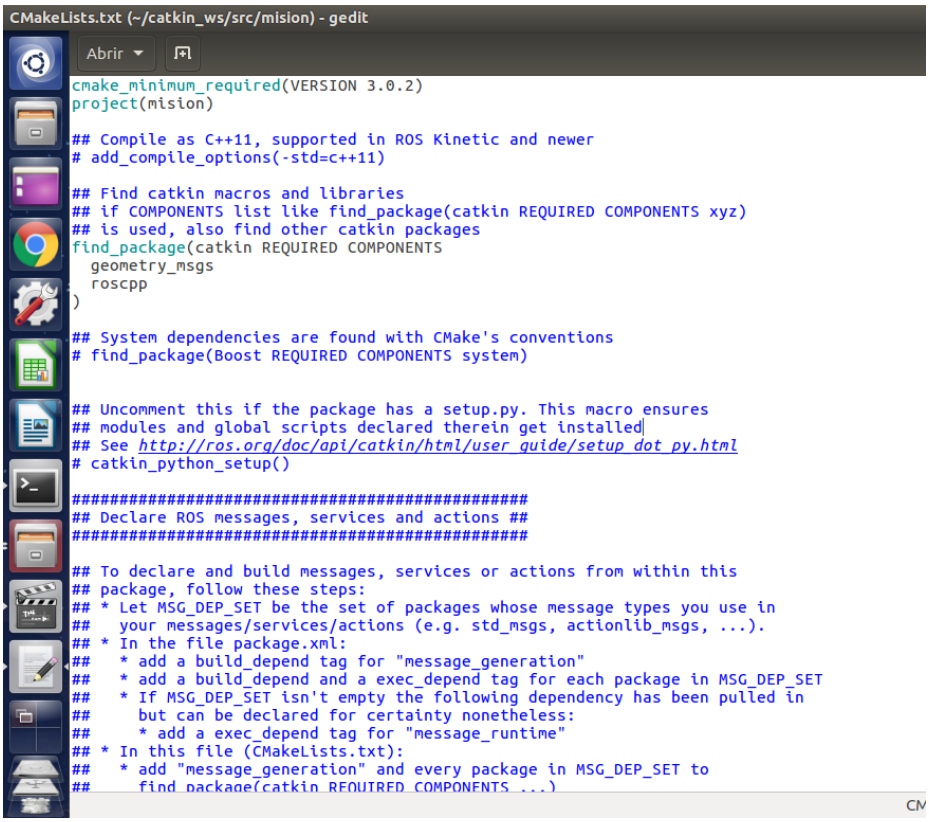


Figura 4.2 - 5: CmakeList.txt

Para hacer estas modificaciones, lo que vamos a realizar consiste en suprimir algunos de los comandos comentados, los cuales no vamos a necesitar, como ejemplo.

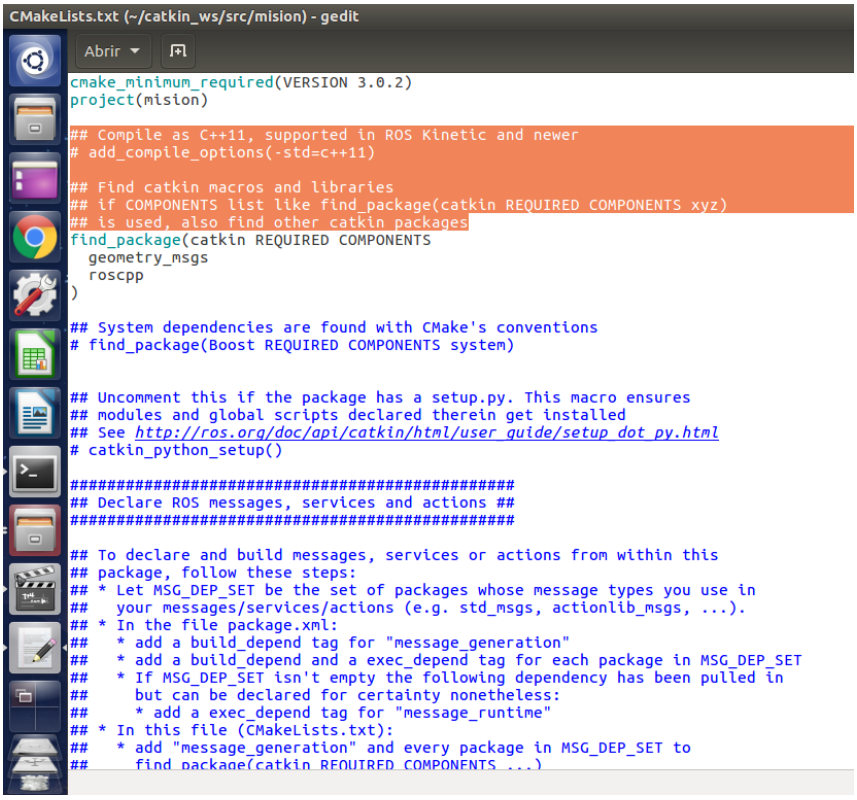


Figura 4.2 - 6: Edición CMakeList

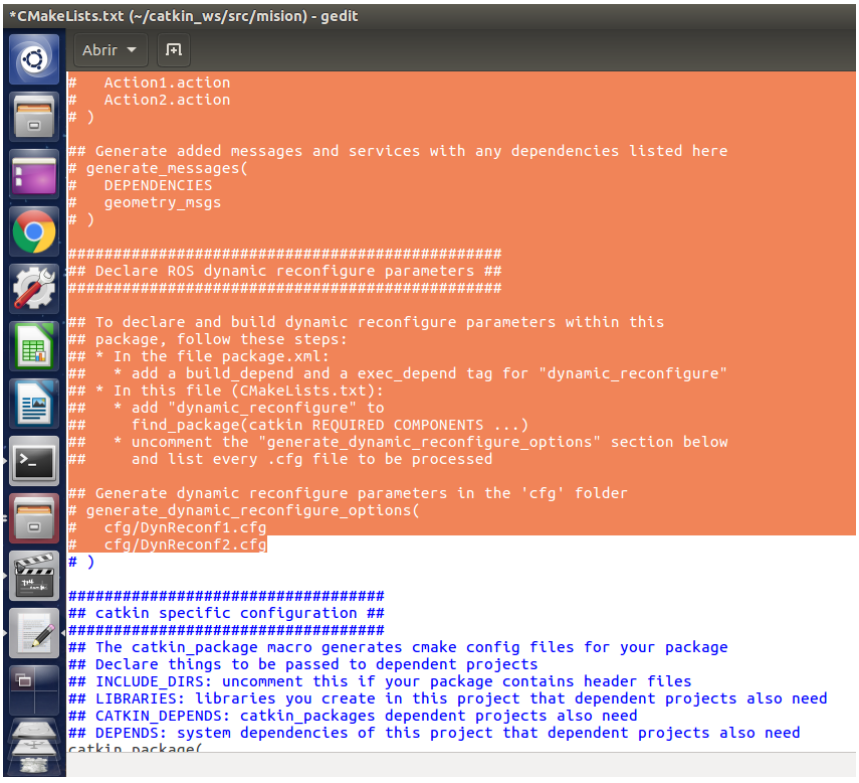


Figura 4.2 - 7: Edición CMakeList

Este proceso de eliminación debemos de hacerlo cuidadosamente debido a que mientras hacemos la eliminación de los comandos debemos de ir dejando algunos comandos importantes, además de ir quitando sus correspondientes comentarios, los comandos en cuestión corresponden a los siguientes:

- El ejecutable de la imagen que vemos a continuación, debido a que tiene una importante función, esta consiste en realizar la llamada al archivo donde haremos la programación:

```
add_executable(${PROJECT_NAME}_node src/mission_node.cpp)
```

Figura 4.2 - 8: add_executable

- Ahora podemos observar el siguiente comando, el cual hace que podamos llamar a las librerías que hagan falta en catkin :

```
target_link_libraries(${PROJECT_NAME}_node
   ${catkin_LIBRARIES}
)
```

Figura 4.2 - 9: Libraries

Quedando el resultado final asi:

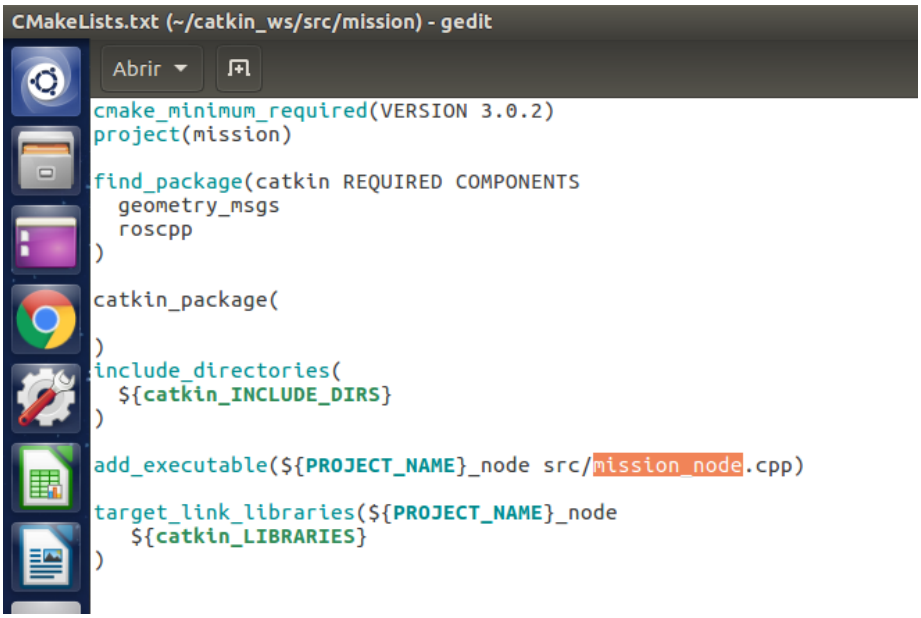


Figura 4.2 - 10: CmakeList final

Ahora vamos a proceder a desarrollar el código necesario para que podamos realizar los distintos objetivos, que como veremos más tarde en este proyecto se centran en que podamos controlar el movimiento del drone.

Lo primero que vamos a usar es el comando que vemos a continuación para ir a la carpeta src de nuestro paquete:

```
cd src
```

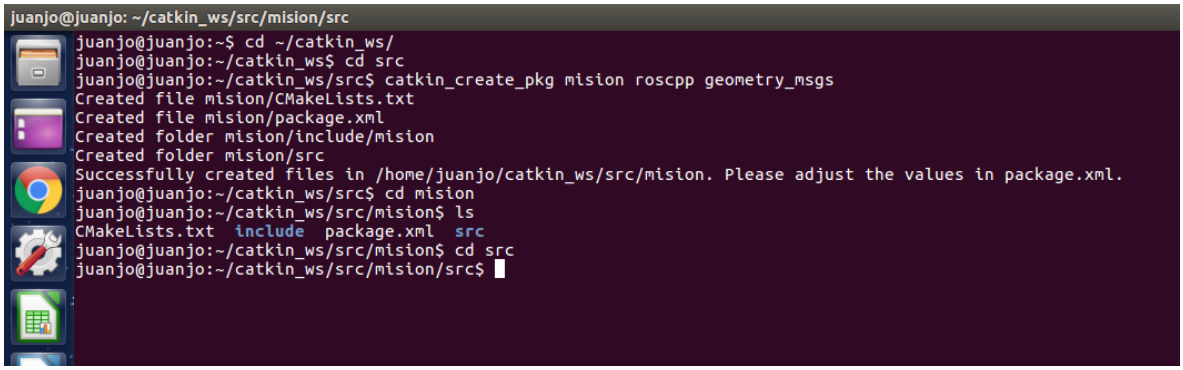


Figura 4.2 - 11: Accediendo a la carpeta src de la carpeta misiion

A continuación, y una vez dentro de esta carpeta en ROS debemos de crear un fichero de código fuente (con extensión .cpp), donde escribiremos la programación para desarrollar los distintos movimientos.

```
gedit nombre.cpp
```

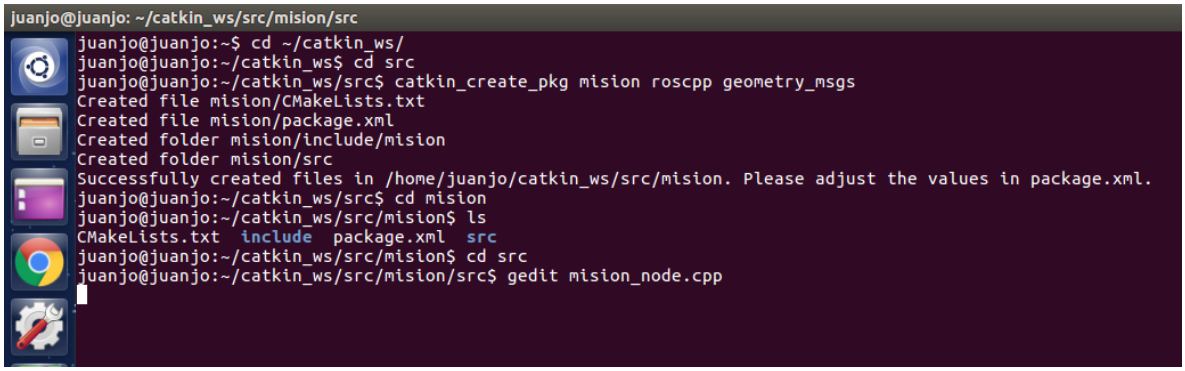


Figura 4.2 - 12: Abriendo el fichero de código fuente con Gedit

Dando lugar a un fichero vacío, llamado misión_node.cpp, que es donde escribiremos todo el conjunto de instrucciones necesarias.

Figura 4.2 - 13: misión node vacío

El siguiente paso consiste en empezar a crear código, esto lo haremos partiendo desde un punto de partida, dicho punto es el que vemos a continuación.

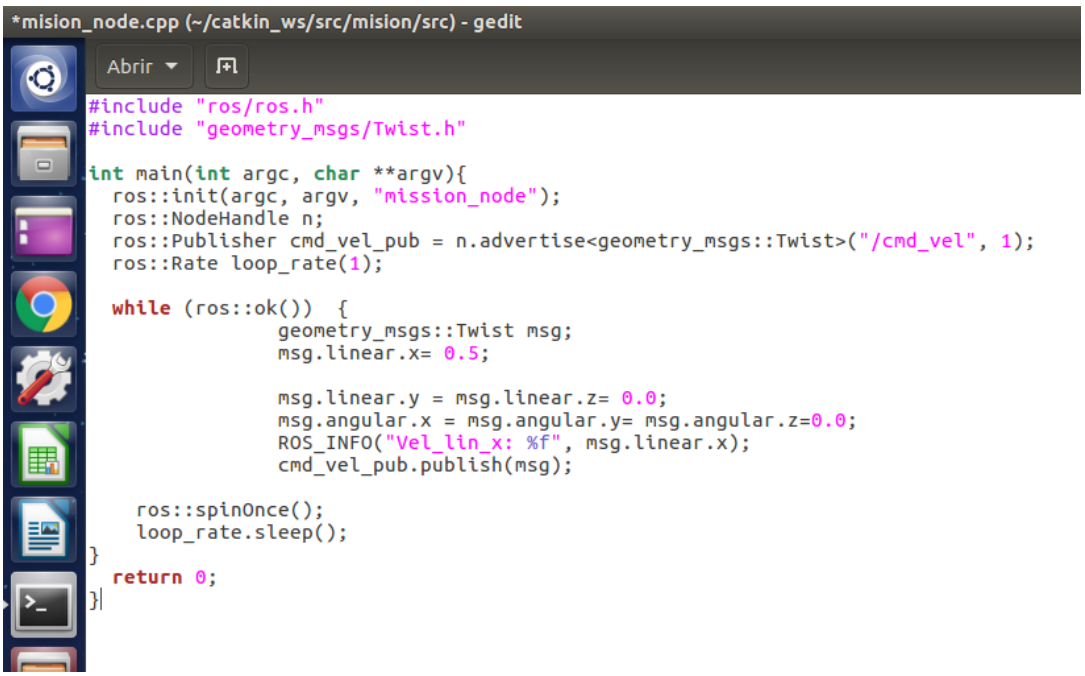


Figura 4.2 - 14: Misión node

Este punto de partida consiste en una serie de comandos, que a continuación vamos a proceder en hacer un breve desarrollo de los comandos más importantes sus funciones principales.

- Al inicio incluimos las librerías de ROS y la que necesitamos para enviar los comandos de velocidad, pero para nuestros programas como veremos usaremos más librerías.
- `ros::init(argc, argv, "mission_node");`

  → Esta función inicia ROS para registrar el nodo en el ecosistema de ROS.

- `ros::Publisher cmd_vel_pub = n.advertise<geometry_msgs::Twist>("/cmd_vel", 1);`

  → Es una función de tipo Publisher, que indica que vamos a enviar mensajes tipos twist, velocidad, usando el topic cmd_vel, que es el puerto y el uno indica que no vamos a almacenar mensajes.

- `while (ros::ok()) {}`

  → indica que mientras el entrono ros este activo hacemos, lo que incluyamos en los corchetes que van a continuación.

- `geometry_msgs::Twist msg;`

  → Declaración de un mensaje de velocidad el cual tiene dos campos, lineal y angular, ambos con las tres coordenadas, x, y, z. cada una de ellas le podemos dar una velocidad distinta .

- `ROS_INFO("lo que muestra%f",variable);`

  →esto va a hacer que se vaya a ir imprimiendo por pantalla la información que le digamos que nos muestre, siendo %f el comando que hace que se muestre el valor de la variable .

- `cmd_vel_pub.publish(msg);`

  → El motivo de escribir este comando es publicar el mensaje de velocidad, para que sea efectivo y el dron se mueva con los valores que le hemos dado

El siguiente paso que debemos de dar es el de compilar, esta compilación se debe de ejecutar una vez que estemos en la carpeta que corresponde a la localización del paquete a evaluar. Anteriormente vimos cómo hacerlo y que en nuestro caso lo hemos llamado catkin_ws:

```
Cd catkin_ws
```

Una vez que ya estemos situados en el interior de la carpeta ya podemos realizar la compilación, y lo haremos con el siguiente comando:

```
Catkin_make
```

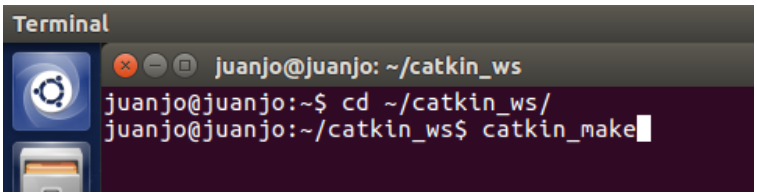


Figura 4.2 - 15: catkin_make

Si hubiere errores en nuestro código nos avisaría, además de no realizarse la compilación hasta rectificarlos. Una vez corregidos podemos compilar, viéndose el resultado como la imagen que vemos a continuación.

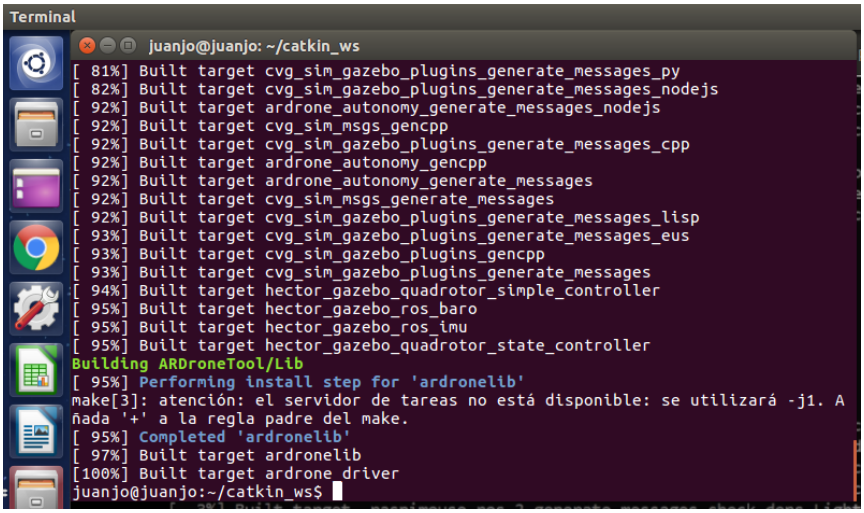


Figura 4.2 - 16: compilación del paquete

Y, por último, una vez todo este correctamente compilado podemos ejecutar la misión que hemos programado, mediante:

```
Rosrun mission mission_node
```

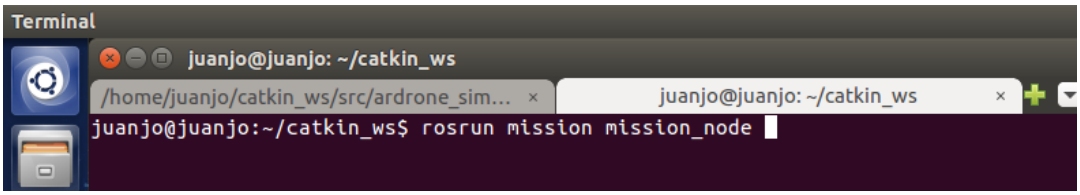


Figura 4.2 - 17: ejecución del paquete usando el comando rosrun

Esto es lo que hemos realizado para crear el espacio de trabajo, los paquetes y para empezar a diseñar nuestros códigos, los que a continuación veremos con más profundidad en los siguientes puntos.

4.3 Mi entorno

El entorno que he diseñado según los pasos del anterior punto es el siguiente:

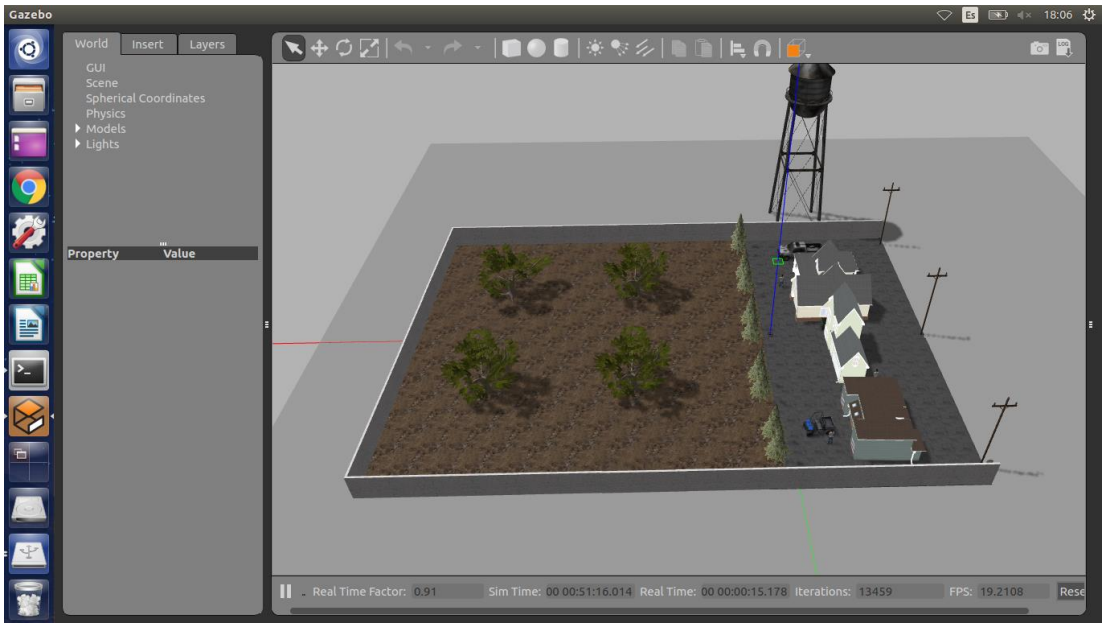


Figura 4.3 - 1: Mi entorno

Donde podemos ver el dron en la siguiente posición:

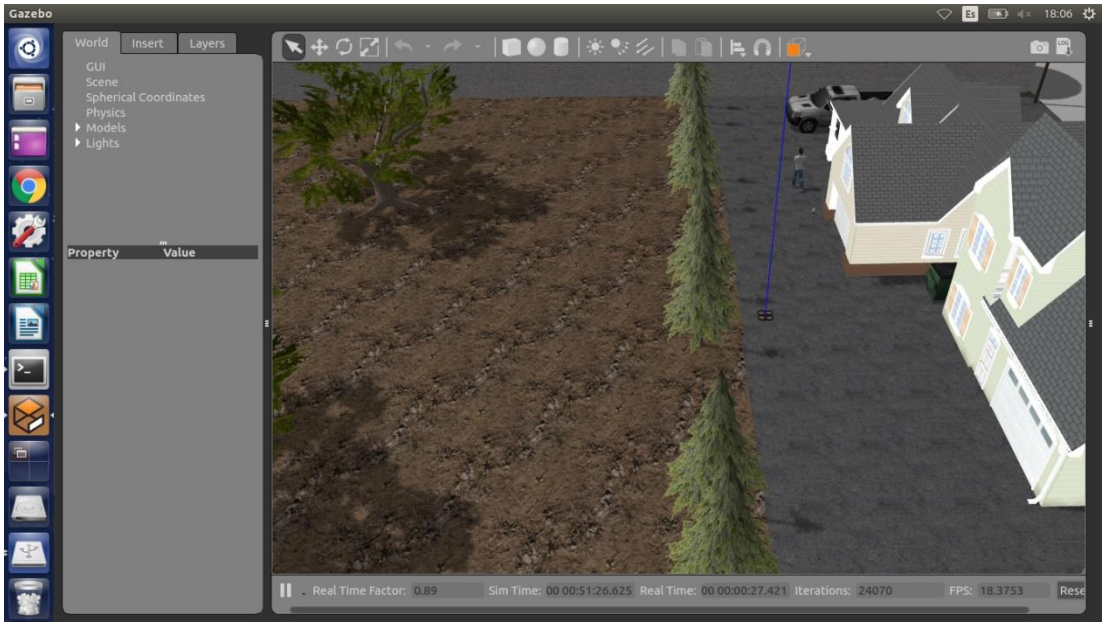


Figura 4.3 -3: Dron en gazebo

Lo he hecho así porque este proyecto consiste en un dron basado en agricultura. Por dicho motivo este mundo consta de una serie de árboles situados como en una especie de huerto tipo rancho.
Este mundo ha sido creado a partir de un mundo ya CREADO en gazebo y los árboles que se ven en las imágenes los he colocado a partir de una base de datos de gazebo y han sido colocados en cada posición mediante las opciones de gazebo las cuales permiten colocarlos en una posición simétrica y girarlos al gusto del usuario.

4.4 La misión de agricultura

4.4.1 Objetivo

La finalidad de este paquete llamado “mission” es que el AR Dron realice una tarea basada en la agricultura. Dicha tarea consiste en que el dron sobrevuele la plantación en cuestión, que en mi caso es de robles, y que en dicho vuelo pueda realizar una observación de la evolución de la plantación, como de fotografiar a la misma.
Este programa también pretende ser la base de otros que puedan realizar distintos trabajos como pueden ser fumigar si hubiese que hacerlo o incluso vigilar, etc.
Para ello, en este punto veremos los flujogramas desarrollados para entender el proceso de programación del dron y la explicación de la programación.

4.4.2 Flujograma

En esta sección se presentan dos flujogramas: teórico y práctico.
El flujograma teórico es el que la teoría indica que debería de funcionar, pero al no considerar inicialmente algunos efectos del simulador Gazebo, ha sido necesario realizar algunas modificaciones, dando lugar al flujograma denominado práctico. A modo de ejemplo, en el flujograma práctico se han tenido en cuenta fuerzas que simula Gazebo, tales como el viendo y la gravedad.

El funcionamiento del flujograma teórico se puede describir de la siguiente manera:
Después de incluir las librerías, declaramos las variables que vamos a usar, como empezamos de cero. vamos a comenzar con un mensaje "empezamos" y leemos un scanf y si en esa lectura decimos que e='s' empezamos con el código, si no acabamos el programa.
Una vez comenzado, y una vez que hemos declarado a=j despegamos y declaramos a =s y así sucesivamente según sea el movimiento que queramos realizar.
Lo último por destacar es que en un momento del código usamos las variables c & d para seleccionar el movimiento de a y así poder realizar un código más concreto y no entrar en una redundancia ya que hay varios movimientos y giros que se repiten.
Una vez que hemos hecho todos los movimientos y para finalizar hacemos un land para aterrizar y declaramos a='o' para volver a comenzar el programa, si queremos finalizar hacemos e ≠ 's'.

El funcionamiento del flujograma modificado es muy parecido al teórico la única diferencia es como ya hemos mencionado es que al realizar la simulación es que los giros no se realizaban de la misma forma pro tanto se ha tenido que añadir distintos giros según la situación que nos íbamos encontrando, para ello se usan c & d, pero dándole más opciones para poder seleccionar los distintos movimientos.

4.3.2.1 Teórico

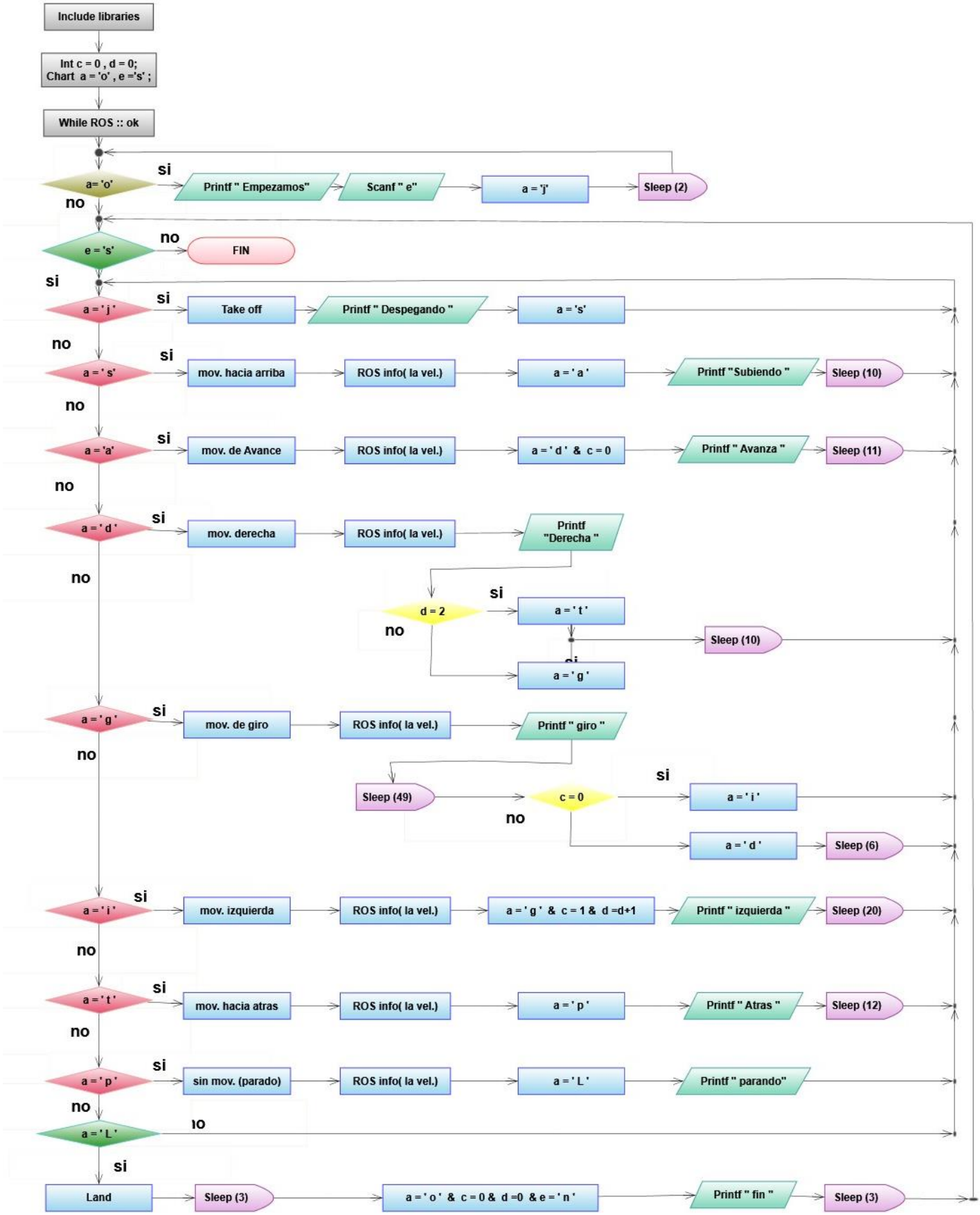
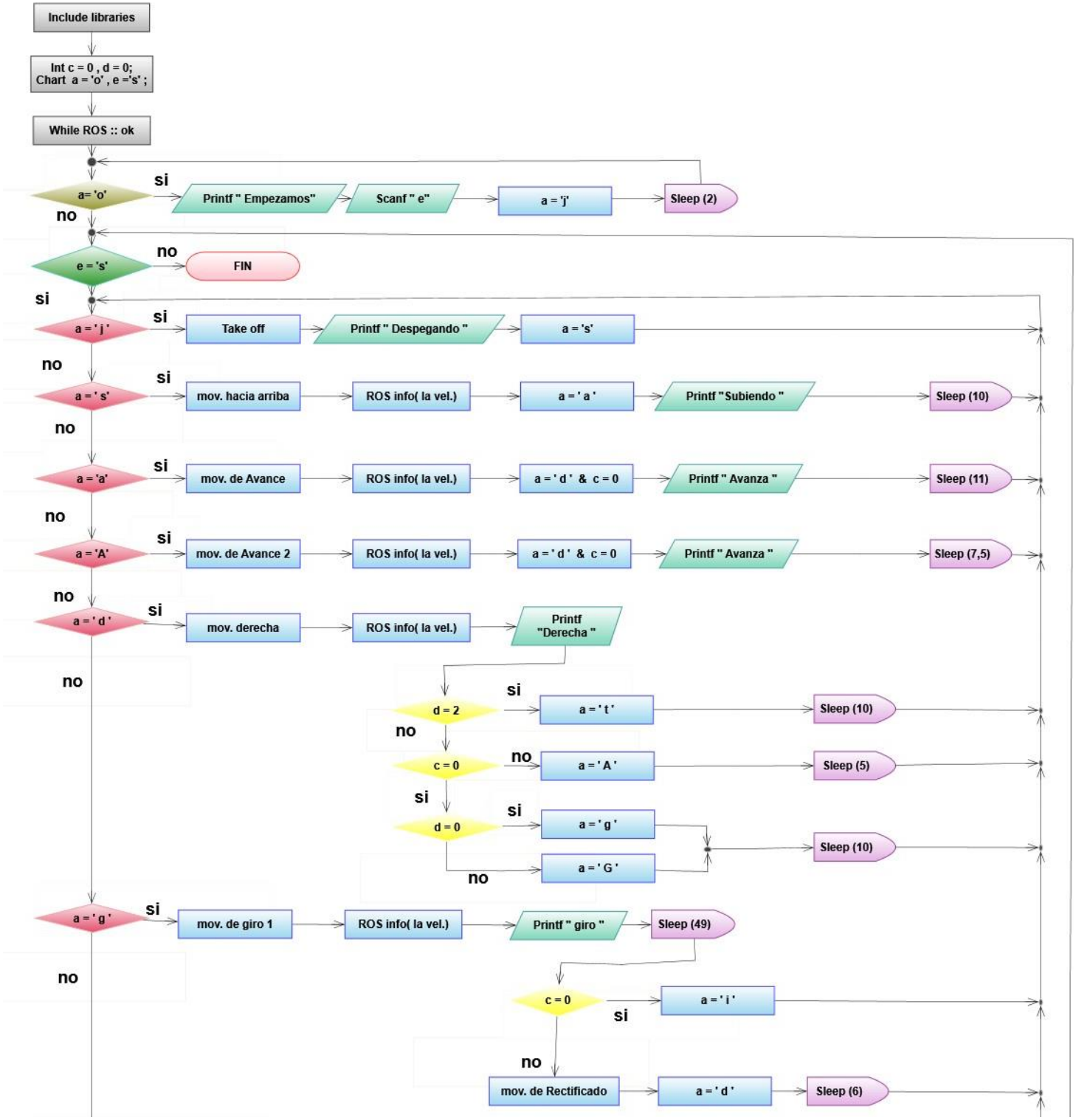


Figura 4.3.2 - 1: Flujograma teórico

7.3.2.2 Modificado



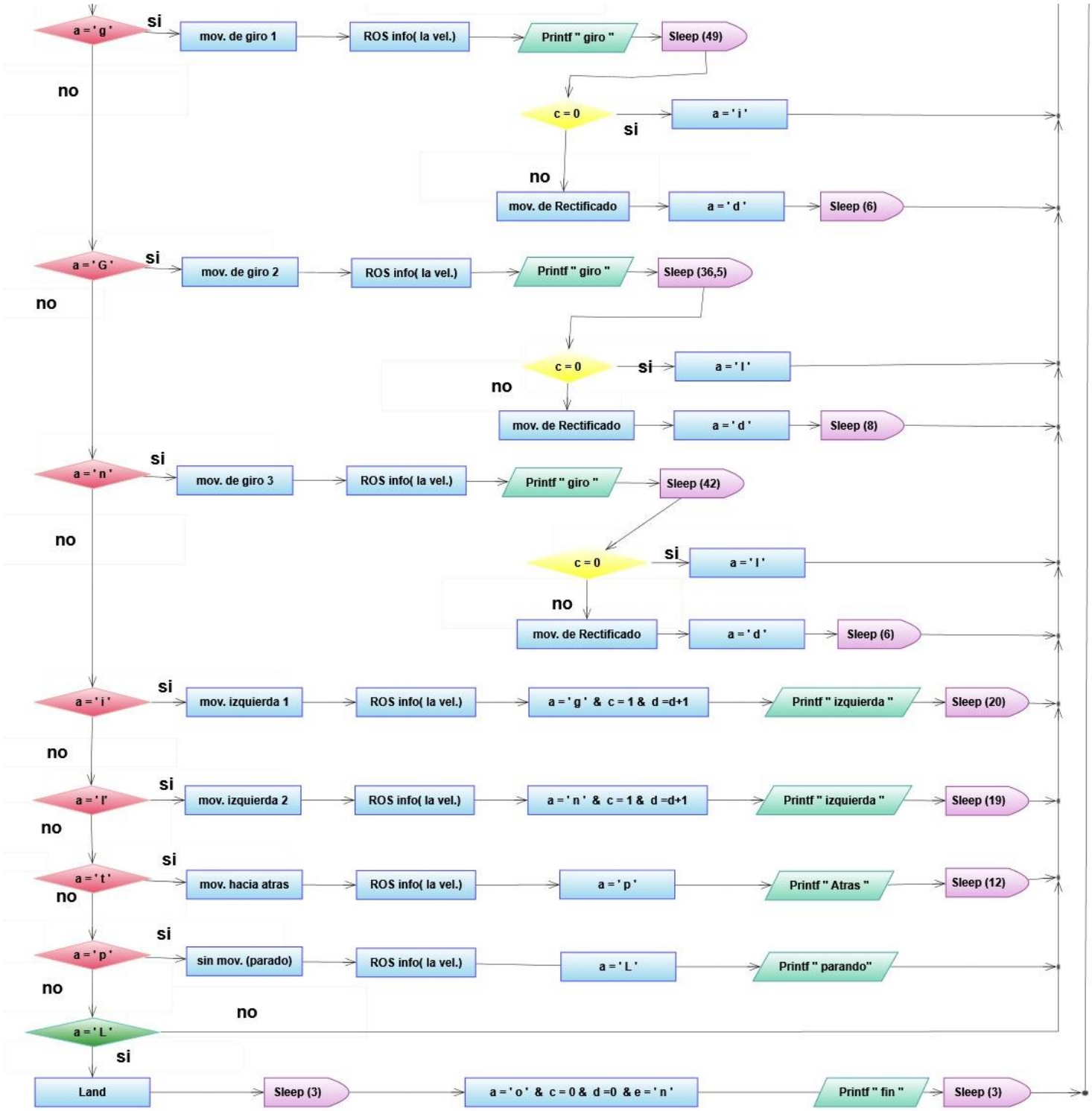


Figura 4.3.2 - 2: Flujograma Modificado

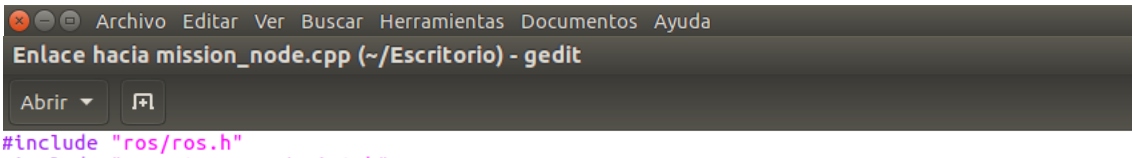
4.4.3 Programación

El procesamiento del código es similar al del flujograma, es decir, funciona de la misma manera. Dicho esto, cabe destacar que su funcionamiento es de la siguiente manera.

Primero incluimos los 'include' con los ficheros necesarios para poder usar los comandos que necesitemos

Luego declaramos las variables con el 'int' y el 'char'. También usamos los comandos de ROS para usar posteriormente los comandos que hacen referencia a los topic necesarios para despegar, aterrizar y moverse.

Para finalizar decir que usamos los 'if' y los 'else' para ir seleccionando los movimientos que queremos que el dron vaya realizando, dentro de los' if' hay que decir que usamos el 'std_msgs' para indicar la velocidad y la dirección del movimiento y al final usamos un 'sleep' para dormir el código y que le dé tiempo al dron a desplazarse.




```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "stdio.h"
#include "std_msgs/Empty.h"


int main(int argc, char **argv){
int c=0,d=0;
char a='o',e='s';

  ros::init(argc, argv, "mission_node");
  ros::NodeHandle n;
  ros::Publisher cmd_vel_pub = n.advertise<geometry_msgs::Twist>("/cmd_vel", 1);
  ros::Publisher takeOff=n.advertise<std_msgs::Empty>("/ardrone/takeoff",1);
  ros::Publisher land=n.advertise<std_msgs::Empty>("/ardrone/land",1);

  ros::Rate loop_rate(1);
  while (ros::ok())  {

if(a=='o'){     printf("empezamos(s/n)\n");                    //scanf & print
                scanf("%c",&e);
                a='j';
                sleep(2);}
if(e=='s'){

        if(a=='j'){
                std_msgs::Empty msg;
                takeOff.publish(msg);                                          //el takeoff
                printf("despegado\n");
                a='s';
```

```
        }
if(a=='s'){
        geometry_msgs::Twist msg;                       //sube
        msg.linear.z=0.4;
        msg.angular.z=0.045;
        msg.linear.x = msg.linear.y= 0.0;
        msg.angular.x = msg.angular.y= 0.0;
        ROS_INFO(" Vel_lin_z: %f ",msg.linear.z);
        cmd_vel_pub.publish(msg);
        a='a';
        printf("sube\n");                                       //voy al siguiente paso
        sleep(10);
        }

if(a=='a'){                                             //avanza
        geometry_msgs::Twist msg;
        msg.linear.x= 0.8;
        msg.linear.z= 0.0;
        msg.linear.y =0.0;
        msg.angular.x = msg.angular.y= msg.angular.z=0.0;
        ROS_INFO("Vel_lin_x: %f", msg.linear.x);
        cmd_vel_pub.publish(msg);
                a='d';
                c=0;
        printf("avanza\n");                         //voy al siguiente paso , reinicio c
        sleep(11);
        }
if(a=='A'){                                              //avanze 2
        geometry_msgs::Twist msg;
        msg.linear.x= 2;
        msg.linear.z= 0.08;
        msg.linear.y =0.0;
        msg.angular.z=-0.07;
        msg.angular.x = msg.angular.y=0.0 ;
        ROS_INFO("Vel_lin_x: %f", msg.linear.x);
        cmd_vel_pub.publish(msg);
                a='d';
                c=0;
        printf("avanza\n");                     //voy al siguiente paso , reinicio c
        sleep(7.5);
        }

if(a=='d'){                                              //derecha
        geometry_msgs::Twist msg;
        msg.linear.y=-0.7;

        msg.linear.x = msg.linear.z= 0.0;
        msg.angular.x = msg.angular.y= msg.angular.z=0.0;
        ROS_INFO("Vel_lin_y: %f ",msg.linear.y);
        cmd_vel_pub.publish(msg);
        printf("derecha\n");
        if(d==2) {a='t';
                sleep(10);}
        else{
                if(c==0){
                        if(d==0){a='g';}
                        else{a='G';}
        sleep(10);}
                else{a='A';
        sleep(5);}
                }}
```

```
if(a=='g'){                                                    //giro
        geometry_msgs::Twist msg;
        msg.linear.y= -1.1;
        msg.angular.z= 0.15;
        msg.linear.z= -0.02;
        msg.linear.x = 0.0;
        msg.angular.x = msg.angular.y= 0.0;
        ROS_INFO("Vel_lin_y: %f, Vel_ang: %f", msg.linear.y, msg.angular.z);
        cmd_vel_pub.publish(msg);
        printf("giro\n");
        sleep(49);
                        if(c==0){a='i';}
                        else{
        geometry_msgs::Twist msg;                              //rectificado
        msg.linear.y= 0.0;
        msg.angular.z= -0.12;
        msg.linear.z= 0.024;
        msg.linear.x = 0.0;
        msg.angular.x = msg.angular.y= 0.0;
        cmd_vel_pub.publish(msg);
                        a='d';
        sleep(6);}
                }
if(a=='G'){                                                    //giro2
        geometry_msgs::Twist msg;
        msg.linear.y= -1.5;
        msg.angular.z= 0.15;
        msg.linear.z= 0.025;
        msg.linear.x = 0.0;
        msg.angular.x = msg.angular.y= 0.0;
        ROS_INFO("Vel_lin_y: %f, Vel_ang: %f", msg.linear.y, msg.angular.z);
        cmd_vel_pub.publish(msg);
        printf("giro\n");
        sleep(36.5);
                         if(c==0){a='I';}
                         else{
         geometry_msgs::Twist msg;
         msg.linear.y= 0.0;
         msg.angular.z= -0.045;
         msg.linear.z= 0.8;
         msg.linear.x = 0.0;
         msg.angular.x = msg.angular.y= 0.0;
         cmd_vel_pub.publish(msg);
                         a='d';
         sleep(8);}
                }
        geometry_msgs::Twist msg;
        msg.linear.y= -1.15;
        msg.angular.z= 0.18;
        msg.linear.z= 0.03;
        msg.linear.x = 0.0;
        msg.angular.x = msg.angular.y= 0.0;
        ROS_INFO("Vel_lin_y: %f, Vel_ang: %f", msg.linear.y, msg.angular.z);
        cmd_vel_pub.publish(msg);
        printf("giro\n");
        sleep(42);
                        if(c==0){a='I';}
                        else{
        geometry_msgs::Twist msg;
        msg.linear.y= 0.0;
        msg.angular.z= -0.045;
        msg.linear.z= 0.8;
        msg.linear.x = 0.0;
        msg.angular.x = msg.angular.y= 0.0;
        cmd_vel_pub.publish(msg);
                        a='d';
        sleep(6);}
                }
```

```
            if(a=='i'){                                                 //izquierda
                    geometry_msgs::Twist msg;
                    msg.linear.y= 0.9;
                    msg.angular.z=0.032;
                    msg.linear.z=-0.028;
                    msg.linear.x =0.0;
                    msg.angular.x = msg.angular.y=0.0;
                    ROS_INFO("Vel_lin_y: %f", msg.linear.y);
                    cmd_vel_pub.publish(msg);
                    printf("izquierda\n");
                    a='g';
                    c=1;
                    d=d+1;                                              //d+1
                    sleep(20);}
            if(a=='I'){                                                 //izquierda2
                    geometry_msgs::Twist msg;
                    msg.linear.y= 0.9;
                    msg.angular.z=-0.02;
                    msg.linear.z=-0.040;
                    msg.linear.x =0.0;
                    msg.angular.x = msg.angular.y=0.0;
                    ROS_INFO("Vel_lin_y: %f", msg.linear.y);
                    cmd_vel_pub.publish(msg);
                    printf("izquierda\n");
                    a='n';
                    c=1;
                    d=d+1;                                              //d+1
                    sleep(19);}

if(a=='t'){                                                             //atras
        geometry_msgs::Twist msg;
        msg.linear.x= -3.4;
        msg.angular.z=-0.12;
        msg.linear.y = msg.linear.z= 0.0;
        msg.angular.x = msg.angular.y= 0.0;
        ROS_INFO("Vel_lin_x: %f, Vel_ang: %f, Vel_lin_z: %f ", msg.linear.x, msg.angular.z, msg.linear.z);
        cmd_vel_pub.publish(msg);
        printf("atras\n");
        sleep(12);
        a='p';}

if(a=='p') {                                                            //parar
        geometry_msgs::Twist msg;
        msg.linear.x=msg.linear.y = msg.linear.z= 0.0;
        msg.angular.x = msg.angular.y= msg.angular.z=0.0;
        ROS_INFO("Vel_lin_x: %f, Vel_ang: %f, Vel_lin_z: %f ", msg.linear.x, msg.angular.z, msg.linear.z);
        cmd_vel_pub.publish(msg);
        a='l';
        printf("para\n");}

if(a=='l'){
        std_msgs::Empty msg;
        land.publish(msg);
        sleep(3);
        a='o';
        c=0;
        d=0;
        e='n';
        printf("fin\n");
        sleep(3);}

      }                                                                 //es elde if e s

         ros::spinOnce();
         loop_rate.sleep();
      }                                                                 //es el del while
        return 0;
      }
```

C++ Anchura de la pestaña: 8 Ln 110, Col 36 INS

Figura 4.3.3 - 1: programación de la misión

4.5 Movimiento del dron

4.5.1 Objetivo

La finalidad de este paquete llamado “codeardrone” es que el AR Dron realice una serie de movimientos controlados con el teclado. Dicho movimiento consiste en que el dron pueda ser manejado al antojo del usuario, además en dicho vuelo podremos realizar una observación de la plantación o de lo que deseemos, como de realizar fotografías o incluso vigilar nuestro entorno (si hubiese que hacerlo tendríamos que activar la cámara del AR dron) .
Este programa también pretende ser la base de otros que puedan realizar distintas modificaciones de los tipos de movimiento, como pueden ser los distintos giros.
Para ello en este punto veremos el flujograma desarrollado para entender el proceso de desarrollo para los movimientos del dron y la explicación de la programación.

Figura 4.4.1 - 1: AR dron

4.5.2 Flujograma

EL flujograma en el objetivo en cuestión, el cual he tenido que hacer algún proceso redundante en teoría, debido a que en la experimentación surgían errores y por tanto los he tenido que desarrollar para el correcto funcionamiento del programa.
El funcionamiento de este consiste en que una vez ya hemos.

El flujograma correspondiente es el que veremos a continuación:
Después de incluir las librerías, declarar las variables que vamos a usar, como empezamos con a='s' y b ='p', hacemos un 'take off' , despegamos y escribimos en pantalla 'nos movemos?'.

Luego leemos b y según sea, seleccionamos un movimiento , esto será un bucle hasta que b = 'l' que entonces se produce el 'land', es decir, aterrizamos y escribimos en pantalla 'al suelo, despegamos?'.

Para finalizar volvemos a leer a si es a='s' empezamos otra vez el bucle de b y si a= 'n' acabamos el programa.

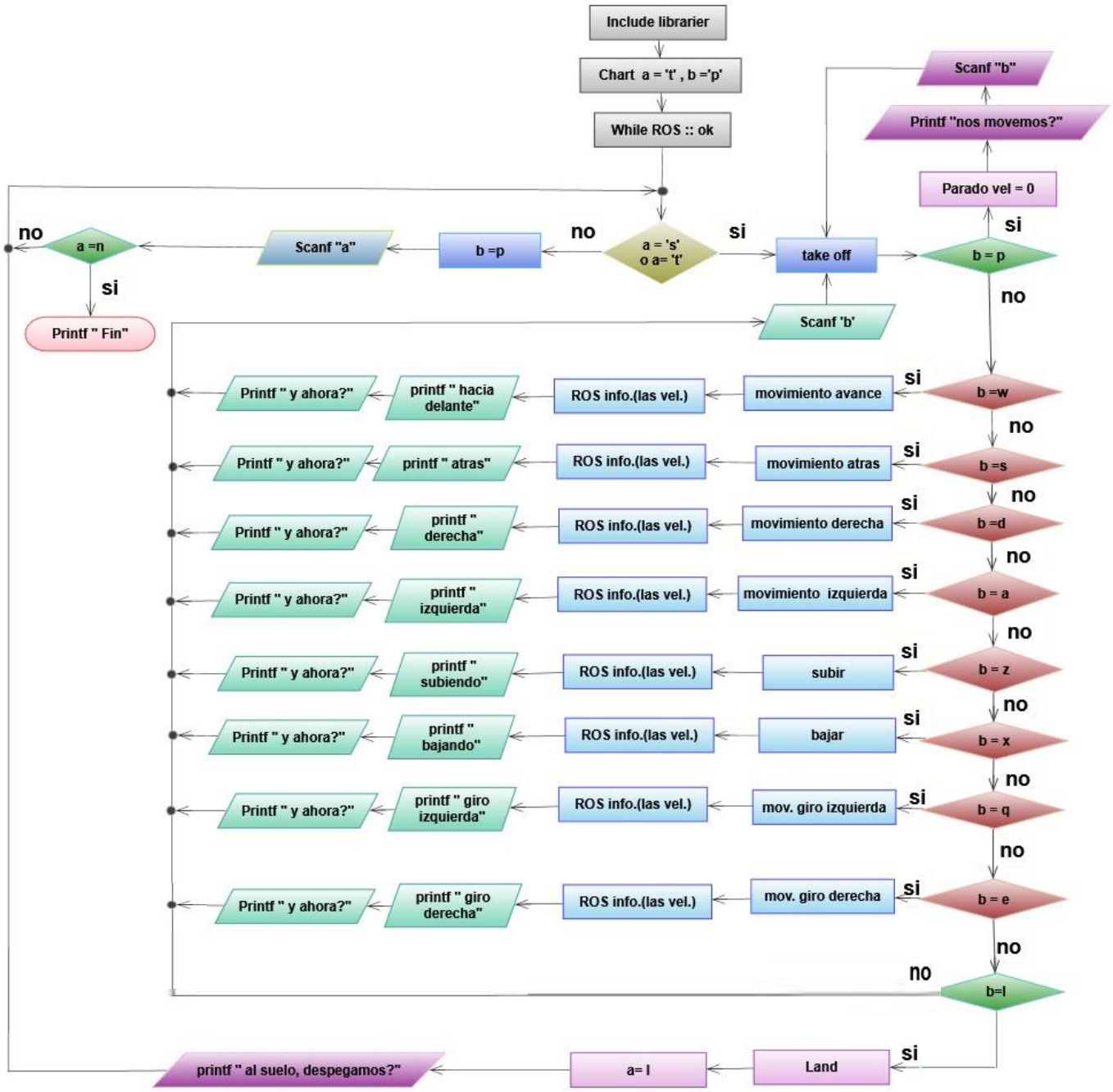


Figura 4.4.2 - 1: flujograma de Movimiento

4.5.3 Programación

El procesamiento del código es igual al del flujograma. Por tanto, su funcionamiento es de forma parecida al código anterior.

Primero incluimos los 'include' ,luego declaramos las variables con el 'int' y el 'char'. También usamos los comandos de ROS para los topic necesarios para despegar, aterrizar y moverse. Y usamos los 'if' y los 'else' para ir seleccionando los movimientos que queremos que el dron vaya realizando, dentro de los' if' hay que decir que usamos el 'std_msgs' para indicar la velocidad y la dirección del movimiento. para finalizar decir que los mensajes que salgan en pantalla utilizamos 'printf' y para leer las variables usamos 'scanf'

Archivo Editar Ver Buscar Herramientas Documentos Ayuda

*Enlace hacia codeardrone_node.cpp (~/Escritorio) - gedit

Abrir

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "stdio.h"
#include "std_msgs/Empty.h"


int main(int argc, char **argv){
char a='t', b='p';

  ros::init(argc, argv, "codeardrone_node");
  ros::NodeHandle n;
  ros::Publisher cmd_vel_pub = n.advertise<geometry_msgs::Twist>("/cmd_vel", 1);
  ros::Publisher takeOff=n.advertise<std_msgs::Empty>("/ardrone/takeoff",1);
  ros::Publisher land=n.advertise<std_msgs::Empty>("/ardrone/land",1);

  ros::Rate loop_rate(1);
  while (ros::ok())  {

        if(a=='t' or a=='s'){                                          //a=t o s para el takeoff, es que hemos hecho land
                std_msgs::Empty msg;
                takeOff.publish(msg);                          //aqui va el takeoff            // como a=t despegamos

                if(b=='p') {                                                    // empezamos apartado y b=p paramos
                geometry_msgs::Twist msg;
                msg.linear.y = msg.linear.x= msg.linear.z= 0.0;
                msg.angular.x = msg.angular.y= msg.angular.z=0.0;
                ROS_INFO("Vel_lin_x: %f, Vel_ang: %f, Vel_lin_z: %f ", msg.linear.x, msg.angular.z, msg.linear.z);
                cmd_vel_pub.publish(msg);
                printf("stop,nos movemos o que hacemos?\n");
                scanf("%c",&b);                                 }
```

```
else{                                                   // segun lo que ponga elijo una de las siguientes opciones

        if(b=='w'){                                                     // palante w
        geometry_msgs::Twist msg;
        msg.linear.x= 0.5;

        msg.linear.y = msg.linear.z=0.0;
        msg.angular.x = msg.angular.y=msg.angular.z= 0.0;
        ROS_INFO("Vel_lin_x: %f, Vel_ang: %f, Vel_lin_z: %f ", msg.linear.x, msg.angular.z, msg.linear.z);
        cmd_vel_pub.publish(msg);
        printf("para adelante\ny ahora:");              }


        if(b=='s'){                                                     // atras s
        geometry_msgs::Twist msg;
        msg.linear.x=-0.5;

        msg.linear.y = msg.linear.z=0.0;
        msg.angular.x = msg.angular.y= msg.angular.z=0.0;
        ROS_INFO("Vel_lin_x: %f, Vel_ang: %f, Vel_lin_z: %f ", msg.linear.x, msg.angular.z, msg.linear.z);
        cmd_vel_pub.publish(msg);
        printf("para detras\ny ahora:");                }


        if(b=='d'){                                                     // derecha d
        geometry_msgs::Twist msg;
        msg.angular.z=-0.5;

        msg.linear.y = msg.linear.x=msg.linear.z=0.0;
        msg.angular.x = msg.angular.y= 0.0;
        ROS_INFO("Vel_lin_x: %f, Vel_ang: %f, Vel_lin_z: %f ", msg.linear.x, msg.angular.z, msg.linear.z);
        cmd_vel_pub.publish(msg);
        printf("rotando a la derecha\ny ahora:");       }


      if(b=='a'){                                                       // izq a
      geometry_msgs::Twist msg;
      msg.angular.z= 0.5;

      msg.linear.y = msg.linear.x= msg.linear.z=0.0;
      msg.angular.x = msg.angular.y= 0.0;
      ROS_INFO("Vel_lin_x: %f, Vel_ang: %f, Vel_lin_z: %f ", msg.linear.x, msg.angular.z, msg.linear.z);
      cmd_vel_pub.publish(msg);
      printf("rotando a la izquierda\ny ahora:");     }


      if(b=='z'){                                                       // subir z
      geometry_msgs::Twist msg;
      msg.linear.z= 0.5;

      msg.linear.y = msg.linear.x=0.0;
      msg.angular.x = msg.angular.y= msg.angular.z=0.0;
      ROS_INFO("Vel_lin_x: %f, Vel_ang: %f, Vel_lin_z: %f ", msg.linear.x, msg.angular.z, msg.linear.z);
      cmd_vel_pub.publish(msg);
      printf("subiendo\ny ahora:");                   }


      if(b=='x'){                                                       // bajar x
      geometry_msgs::Twist msg;
      msg.linear.z= -0.4;

      msg.linear.y = msg.linear.x=0.0;
      msg.angular.x = msg.angular.y= msg.angular.z=0.0;
      ROS_INFO("Vel_lin_x: %f, Vel_ang: %f, Vel_lin_z: %f ", msg.linear.x, msg.angular.z, msg.linear.z);
      cmd_vel_pub.publish(msg);
      printf("bajando\ny ahora:");                    }
```

```
                if(b=='q'){                                                     // girar izquierda q
                geometry_msgs::Twist msg;
                msg.linear.x= 0.8;
                msg.angular.z=0.4;

                msg.linear.y = msg.linear.z=0.0;
                msg.angular.x = msg.angular.y= 0.0;
                ROS_INFO("Vel_lin_x: %f, Vel_ang: %f, Vel_lin_z: %f ", msg.linear.x, msg.angular.z, msg.linear.z);
                cmd_vel_pub.publish(msg);
                printf("giro izquierda\ny ahora:");             }


                if(b=='e'){                                                     // girar derecha e
                geometry_msgs::Twist msg;
                msg.linear.x= 0.8;
                msg.angular.z=-0.4;

                msg.linear.y = msg.linear.z=0.0;
                msg.angular.x = msg.angular.y= 0.0;
                ROS_INFO("Vel_lin_x: %f, Vel_ang: %f, Vel_lin_z: %f ", msg.linear.x, msg.angular.z, msg.linear.z);
                cmd_vel_pub.publish(msg);
                printf("giro derecha\ny ahora:");               }


                //printf("Y ahora:");                                           //printf("Y ahora:");
                scanf("%c",&b);
                }                                                               //else

        if(b=='l'){                                                     //Land
                std_msgs::Empty msg;
                land.publish(msg);      //aqui va el land
                a='l';
                printf("al suelo.\nDespegamos?(s/n)\n");}

    }                                                                   // el final de if inicial del a
    else{
    b='p';
    scanf("%c",&a);}

    if(a=='n'){
    printf("fin\n");
    scanf("%c",&a);}

  ros::spinOnce();
  loop_rate.sleep();
}                                                                       //es el del while
  return 0;
}
```

C++ Anchura de la pestaña: 8 Ln 11, Col 1 INS

Figura 4.4.3 - 1: Programación de Movimiento

Capítulo V

Conclusiones y trabajo futuro

5.1 introducción

Por ir ultimando el trabajo, se aglutinará en el capítulo una sinopsis de las diversas conclusiones obtenidas tras la realización de este, verificando que se ha logrado las metas que inicialmente se planteaban en el comienzo del trabajo. Para finalizar, se comentarán proyectos que se podrán desarrollar en un futuro próximo a partir de lo que se ha expuesto en este trabajo.

5.2 Conclusiones

En este trabajo se ha realizado un estudio de un paquete el cual tiene que poder manejar el dron Ar-Drone 2.0 de Parrot, compatible con el controlador ROS ardrone_autonomy. Tambien hemos visto que el nodo de ROS llamado roscore es el que se dedica a realizar una centralización de las comunicaciones en ROS. Y que existen una serie de funciones que se han puesto a prueba utilizando tanto Gazebo como el controlador de ROS que modela el dron en cuestión en un entorno virtual.

Para llevar a cabo la realización de los objetivos se ha hecho una división en el trabajo dando lugar a una serie de apartados como pueden ser el aprendizaje y uso del dron, conocer el entorno de desarrollo Robótico ROS, desarrollar un escenario en Gazebo, así como desarrollar y probar los paquetes necesarios para el movimiento del mismo.

Lo primero ha sido aprender el funcionamiento del entorno del desarrollador, después continuamos conociendo la forma de trabajar con ROS para conseguir una simulación partiendo de los paquetes que teníamos.

Dicho esto, podemos concluir diciendo que los objetivos del trabajo han sido satisfactorios. Ya que los resultados conseguidos son lógicos en el entorno de gazebo, por tanto, la simulación se ha conseguido ya que el AR Drone vuela correctamente y ejecuta la misión propuesta de forma correcta.

En definitiva, y como resumen, podemos asegurar que con este TFG se ha visto la funcionalidad del sistema operativo robótico ROS para el manejo de cualquier tipo de dron, debido a que este tiene un tipo de lenguaje de alto nivel.

5.3 trabajos futuros

En un futuro, sería interesante hacer estas mismas pruebas con más equipos y con otros protocolos, para tener un estudio más completo del trabajo de los drones y las comunicaciones entre ellos.

Como se comentó, algunos de los trabajos propuestos, asumiendo los conocimientos que después de desarrollar el trabajo hemos adquirido, pueden ser:

- Trabajo en conjunto de drones: consistiría en el uso de otro drone trabajando simultáneamente con el AR Drone 2.0 para hacer más eficiente el proceso desarrollado con la agricultura de precisión.
- Cambio del dron utilizado por otro dron más actual y eficiente: un drone que tenga más ventajas en este proceso, que pueda volar en otras zonas más inaccesibles, que sea capaz de aguantar un mayor lastre. Por tanto, en un futuro se pueda realizar un paquete que admita manejar un nuevo drone de la misma forma que el AR Drone 2.0.
- Uso del AR Drone 2.0 y su Cámara para conseguir un mapeo del terreno que tengamos: La cámara del AR Drone 2.0 puede ser muy eficiente. Se pueden usar los distintos paquetes existentes para poder usar la camara, que ya lleva incorporada el dron, de esta forma se podría realizar el mapeo pertinente, tambien se podría hacer el tres dimensiones.

Para concluir, cabe decir que en el proceso de realización de este trabajo se han conseguido la mayoría de metas que originalmente nos hemos propuesto, además de ser resolutivo a la hora de solucionar algunos de los problemas que se han ido presentando buscando otras formas de realizarlo, haciendo que al final podamos concluir este trabajo diciendo que hemos logrado realizar la simulación del vuelo de una forma exitosa.

Bibliografía

(1, s.f.) Información general sobre el UAV Erle-Copter. Disponible en: https://www.todrone.com/erle-copter-dron-ubuntu/

(2, s.f.) Información general sobre el UAV AR Dron. Disponible on-line en: http://bibing.us.es/proyectos/abreproy/91351/fichero/1351_GERM_TFG_Jimenez_Leon_Mario.pdf

(3, s.f.) Información general sobre el sistema operativo Ubuntu. Disponible en: https://www.xn--linuxenespaol-skb.com/distribuciones/ubuntu/

(4, s.f.) Información general sobre ROS. Disponible on-line en: https://openwebinars.net/blog/que-es-ros/

(5, s.f.) Descripción concreta de ROS. accesible en la web: http://mariachirobot.com

(6, s.f.) Información general sobre gazebo. Disponible on-line en: http://gazebosim.org/tutorials?tut=guided_b1&cat=

(7, s.f.) Información sobre las especificaciones del UAV AR.dron. Esta en la web: https://es.wikipedia.org/

(8, s.f.) El detalle de las características del AR dron lo podemos ver en: https://ardrone-2.es/especificaciones-ar-drone-2/

(9, s.f.) Los tutoriales de ROS los podemos encontrar en la web: http://wiki.ros.org/ROS/Tutorials

(10, s.f.) Referencia sobre los tutoriales de Gazebo. Disponible on-line en: http://wiki.ros.org/simulator_gazebo/Tutorials

(11, s.f.) Información sobre la creación de un paquete. Disponible en: https://geekgasteiz.wordpress.com/2018/02/14/ros-creando-nuestro-workspace-y-primer-paquete/

(12, s.f.) La referencia sobre el montaje del Erle-Copter. hallada en: https://erlerobotics.gitbooks.io/erle-robotics-erle-copter/content/es/variado/assembly.html

(13, s.f.) Información general sobre Ubuntu. Disponible on-line en: https://es.wikipedia.org/wiki/Parrot_(empresa)

(14, s.f.) Instalación general de Ubuntu 14. Disponible en: https://www.adslzone.net/2016/04/21/ya-puedes-descargar-gratis-la-iso-ubuntu-16-04-lts-xenial-xerus/

(15, s.f.) Actualización de Ubuntu 14 a 16. Encontrada en: https://www.xataka.com/basics/como-instalar-linux-a-windows-10-ordenador

(16, s.f.) Instalación de Ubuntu 16, paso a paso. Encontrada en: https://www.gestionatuweb.net/instalar-ubuntu-16-04-lts-paso-a-paso/

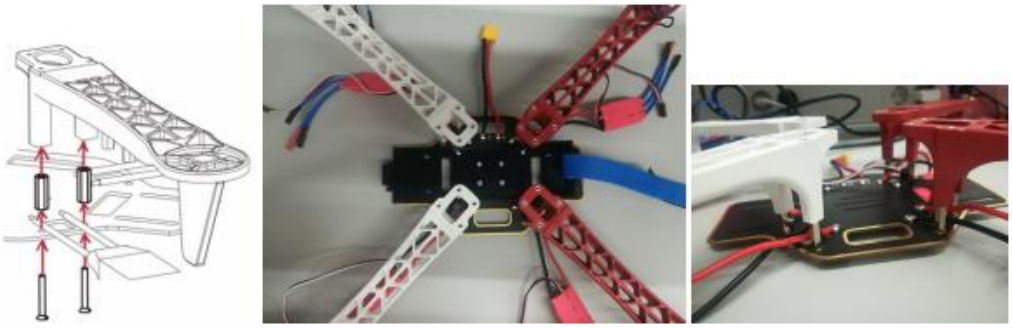
(17, s.f.) Instalación de ROS. Encontrada en: https://geekgasteiz.wordpress.com/2018/01/18/instalando-ros-en-ubuntu-16-04/

Anexo I

Montaje erlecopter

Para realizar el montaje seguiremos las instrucciones presentes en el apartado "assembly" de su página web correspondientes a Erle robotics gitbooks, " https://erlerobotics.gitbooks.io/erle-robotics-erle-copter/content/es/index.html". Debido a que la página de erleborotics ya no está disponible, ni el manual original tampoco, además de información obtenida de distintas fuentes. Montaremos el dron siguiendo estos pasos (12, s.f.):

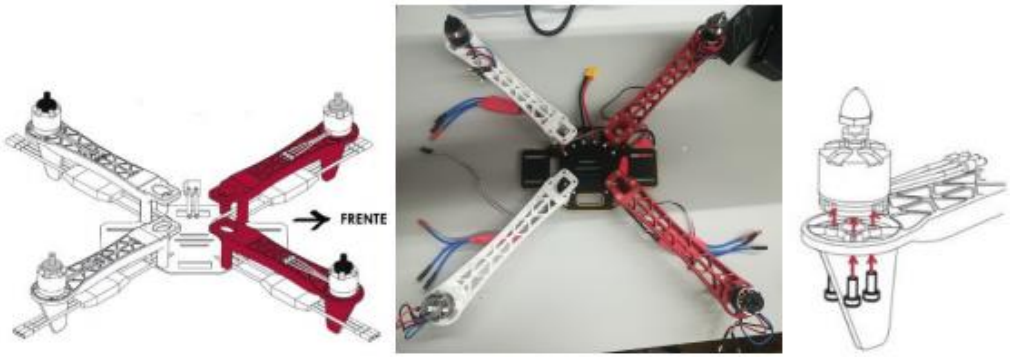
- En primer lugar, se fijan los cuatro brazos a la parte metálica inferior del chasis mediante ocho tornillos con embellecedores. Los dos brazos rojos indicarán la parte delantera del dron. Mientras que los dos brazos blancos la trasera.



(a) Montaje de los brazos *(b) Montaje de los brazos* *(c) Embellecedores*

Figura I - 1: Montaje del chasis

- Se colocan los cuatro motores. Uno por cada brazo y se fija cada uno de ellos con cuatro tornillos. Se deben colocar tal y como se muestra a continuación. Enfrentados dos a dos, los motores negros y los plateados.



(a) Esquema montaje motores de Erle-Robotics (b) Montaje de los motores (c) Montaje de los motores

Figura I - 2: Montaje de los motores

Es importante destacar el montaje en una posición correcta de los motores, que según notifican las instrucciones, en nuestro Erle-Copter DIY kit se incluyen motores distintos a los que habitualmente se utilizan y, según el manual proporcionado por la compañía, deben ir colocados en una posición diferente. La colocación correcta de éstos se aprecia en la siguiente imagen.

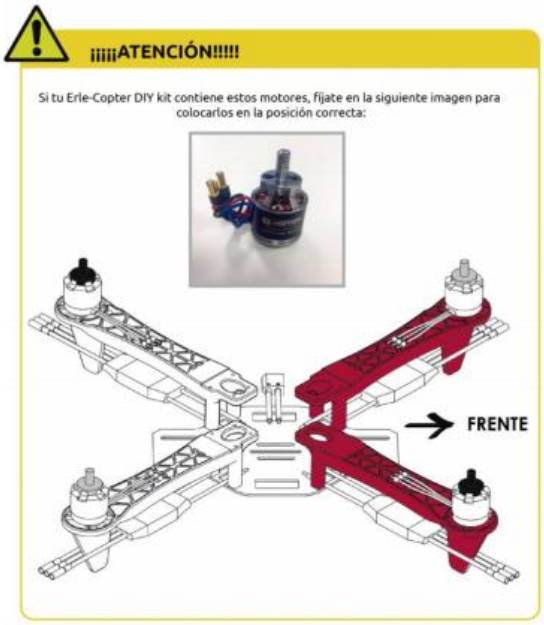


Figura I - 3: Posición correcta de los motores en Erle-Copter

Teniendo en cuenta la imagen anterior se asocia un número distinto a cada motor según su posición. De este modo, los motores 1 y 2 se corresponden con los motores de punta de color negro y los números 3 y 4 con los de punta de color blanco. Esta distribución se aprecia de mejor forma en esta otra ilustración.

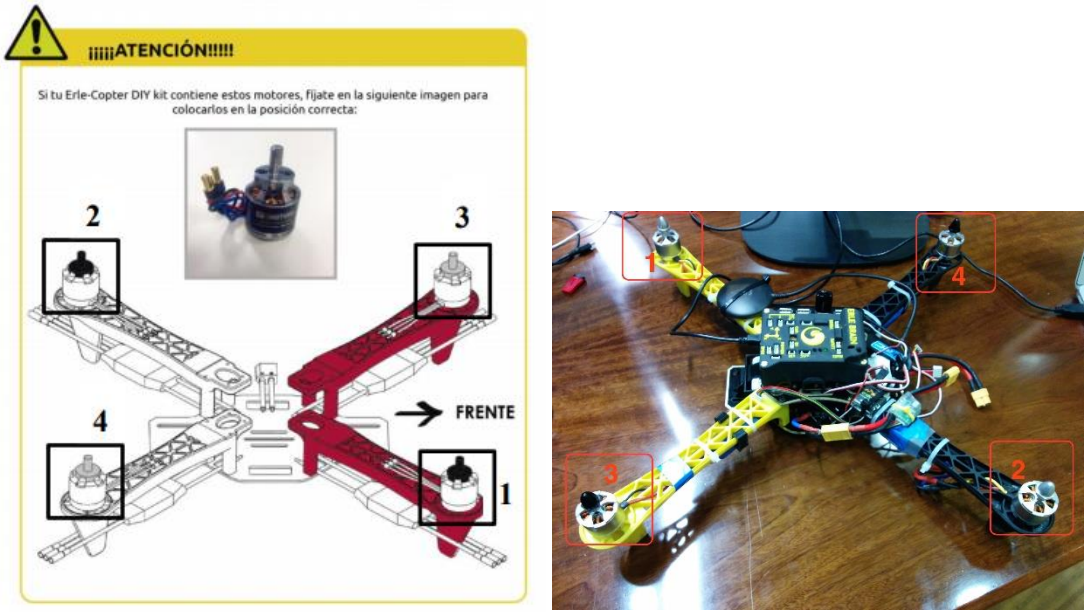
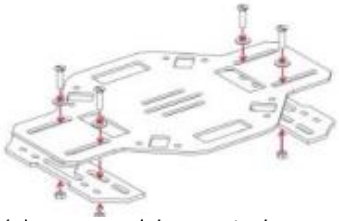
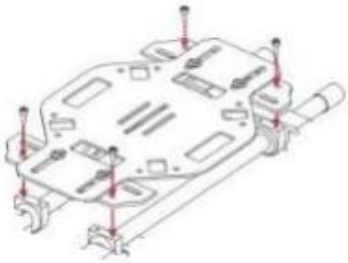


Figura I - 4: Numeración de los motores en Erle-Copter

- Se fijan dos estructuras metálicas en la parte inferior del chasis, las cuales, servirán de base para el montaje de las patas del Erle-Copter.



(a) Esquema del montaje de las estructuras metálicas

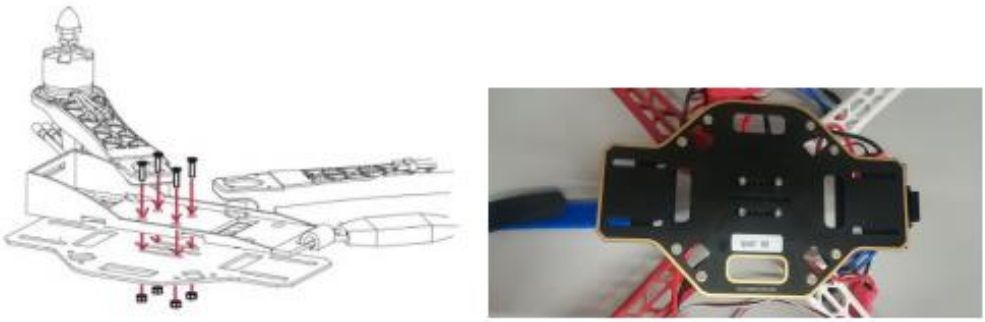


(b) Esquema del montaje de las patas

(c) Montaje

Figura I - 5: Montaje de las patas

- Se monta el soporte para la batería en la parte inferior metálica del chasis con cuatro tornillos y cuatro tuercas.

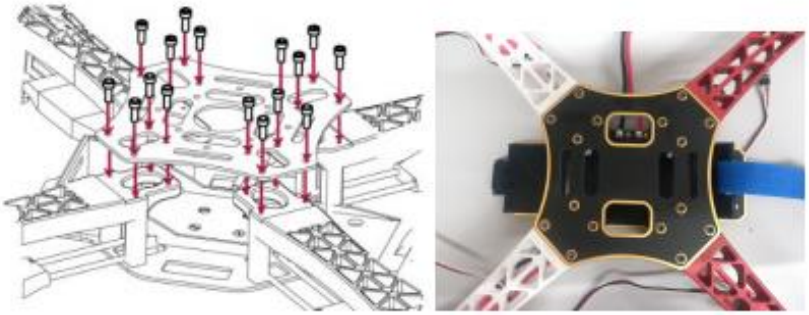


(a) Esquema montaje del soporte para la batería de Erle-Robotics

(b) Montaje del soporte del batería visto desde abajo

Figura I - 6: Montaje del soporte de la batería

- Se atornilla la parte superior metálica del chasis. Esta estructura servirá como base al Erle-Brain.

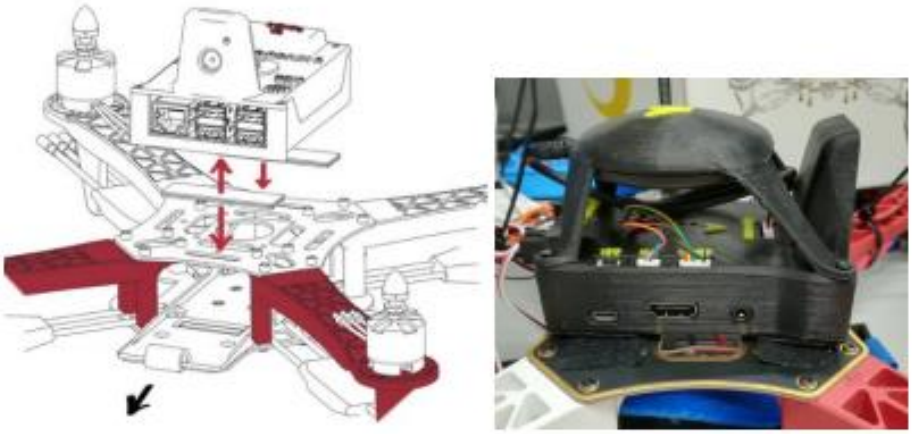


(a) Esquema montaje parte superior del chasis de Erle-Robotics

(b) Montaje de la parte superior del chasis

Figura I - 7: Montaje de la parte superior del chasis

- A continuación, mediante dos tiras de velcro adhesivo de doble cara se adhiere el Erle-Brain a la parte superior del chasis que ha sido fijada en el paso anterior.



(a) Esquema montaje del Erle-Brain de Erle-Robotics

(b) Montaje del Erle-Brain

Figura I - 8: Montaje del Erle-Brain

- El siguiente paso que se debe dar es conectar los ESCs a los motores. Se debe tener especial cuidado ya que esta conexión varía según el tipo de motor que se tenga, y de ella depende que el sentido de los motores sea el correcto. En este caso, al disponer de motores T-motor MN2213 la conexión mostrada en la figura es la forma correcta de conectar ambos componentes.
  Una vez que se han conectado los variadores a los motores, se deben fijar a los brazos del chasis mediante bridas para evitar daños en ellos durante el vuelo.

(a) *Esquema conexión de los ESCs a los motores de Erle-Robotics*

(b) *Conexión de un motor T-motor MN2213 a su ESC*

Figura I - 9: : Conexión entre los motores T-motor MN2213 y los ESCs

Sin embargo, hay que destacar que esto puede inducir a error ya que con dicha conexión los motores giran en dirección opuesta a la adecuada. La dirección de giro necesaria para que el dron se controle de forma satisfactoria aparece en la siguiente figura.

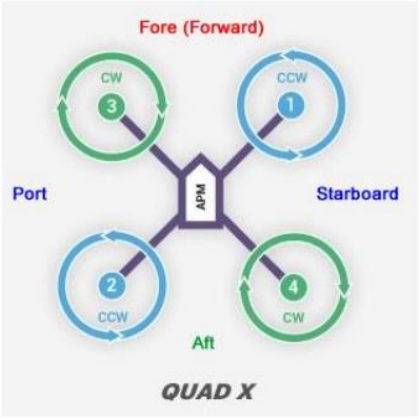


Figura I - 10: Giro adecuado de los motores en un quadrotor

Teniendo en cuenta la imagen anterior, una configuración general para la conexión entre variadores y motores, independientemente del modelo de éstos y teniendo en cuenta el sentido de giro, puede obtenerse de la siguiente tabla,

Tabla I - 1: Conexiones ESC-motores según el sentido de giro de los mismos

Sentido de giro	Motores	Conexión
En contra de las agujas del reloj	Uno y dos	+ *ESC* y - Motor - *ESC* y + Motor
A favor de las agujas del reloj	Tres y cuatro	+ *ESC* y + Motor - *ESC* y - Motor

Por lo tanto, siendo de color rojo los cables positivos y de color negro los negativos en ambos componentes, en los motores número 1 y 2 se cruzan los cables para invertir la polaridad, mientras que en los números 3 y 4 se conectan de manera directa a los variadores. Las conexiones expuestas en la Tabla 3-1 son válidas para cualquier tipo de vehículo quadrotor.

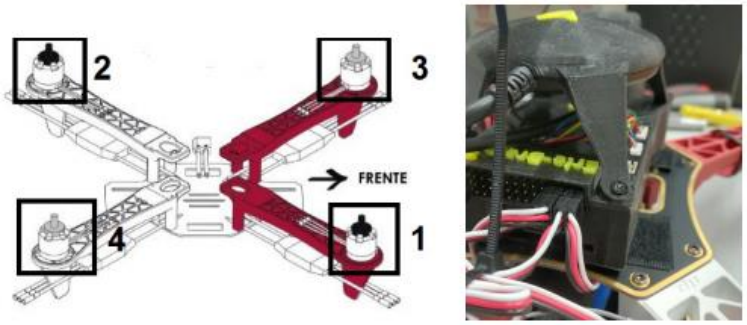
- Posteriormente, para conectar los distintos variadores a Erle-Brain 2 debemos tener en cuenta el orden de los distintos pines PWM disponibles en el controlador. Puede verse en la siguiente imagen.

Figura I - 11: Orden de los pines PWM presentes en Erle-Brain 2

Se conectan los ESCs a las salidas PWM del Erle-Brain según el esquema de derecha a izquierda, siendo el ESC identificado con 1 el conectado a la PWM situada más a la derecha, y el ESC nombrado con un cuatro el conectado más a la izquierda de los cuatro ESCs. En el pin de entrada del extremo izquierdo (pin número 14) debe ir conectado el receptor de señal, tal y como puede verse en la siguiente ilustración

Figura I - 12: Modo de conexión del receptor de señal de radio

Además, los conectores deben quedar con el cable blanco en la parte superior tal y como se observa en la imagen.

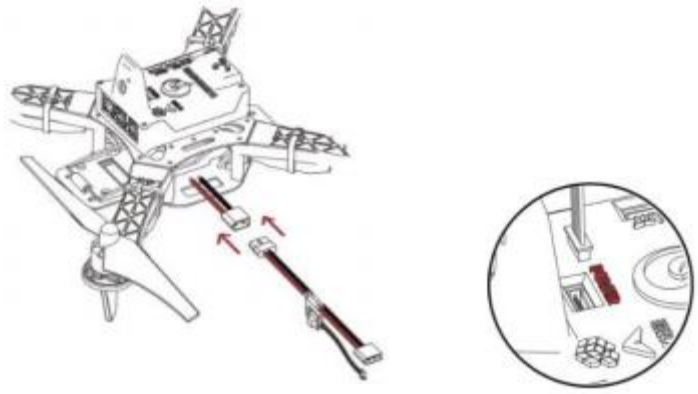


(a) Esquema conexión de los ESCs al Erle-Brain

(b) Conexión de los ESCs al Erle-Brain

Figura I - 13: Conexión de los ESCs al Erle-Brain.

- Se conecta el Power Module al chasis del UAV mediante el conector XT-60,y el Erle-Brain mediante el cable DF13 de 6 posiciones tal y como se puede observar en la imagen.

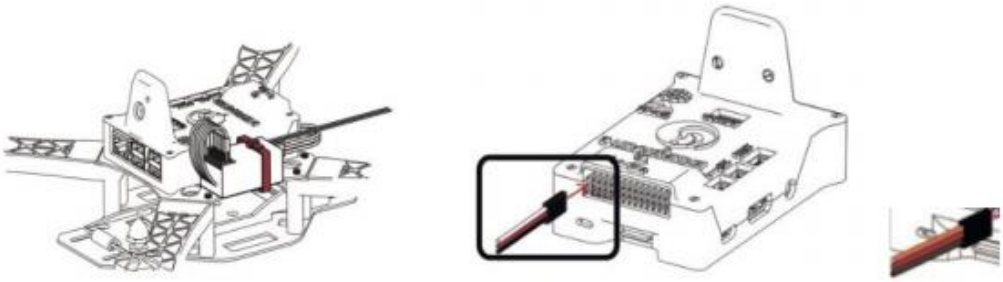


(a) Esquema conexión del Power Module al Chasis mediante el XT-60

(b) Conexión del Power Module al Erle-Brain mediante el DF13

Figura I - 14: Conexión del Power Module al Erle-Brain.

- A continuación, se fija el receptor del mando de radio control al lado del Erle-Brain mediante una brida. Y se conecta a este, en el canal 14 de sus pines de manera que el cable naranja del conector quede en la parte superior.



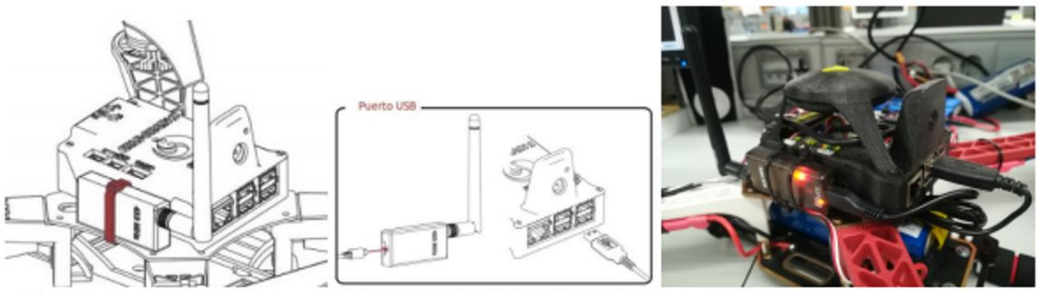
(a) Posicionamiento del receptor del mando de radio control

(b) Conexión del receptor al Erle-Brain

(c) Conector

Figura I - 15: Montaje del receptor del mando radio control.

- Al igual que se ha hecho en el paso anterior, se fija el módulo de telemetría mediante una brida en el otro lateral del Erle-Brain y se conecta a este, en nuestro caso, mediante un puerto USB como se muestra en la imagen. También existe la posibilidad de realizar esta conexión mediante la UART. Por otro lado, el segundo módulo de telemetría será conectado al ordenador mediante un puerto USB.



(a) Posicionamiento del módulo de telemetría (b) Conexión de la telemetría al Erle-Brain (c) Posicionamiento del módulo de telemetría

Figura I - 16: Montaje de la telemetría.

- Tras haber montado la telemetría, se colocan las cuatro hélices. Para ello, se deben hacer girar las hélices 1 y 2 en sentido antihorario, y las hélices 3 y 4 en sentido horario tal y como se muestra en la imagen 3.12 . Cuando se desee desmontar las hélices se deberá girar en los sentidos opuestos a los nombrados anteriormente.

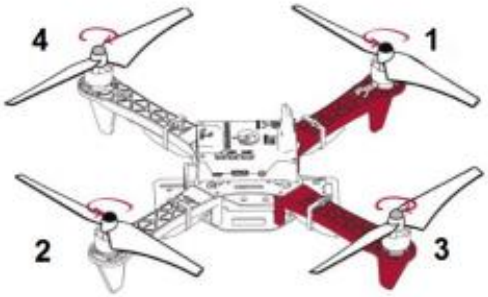
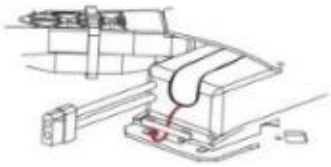


Figura I - 17: Montaje de las hélices

- Para finalizar con el montaje del UAV, se conecta el comprobador de voltaje a la batería para poder comprobar dicho valor durante el vuelo. Esta conexión se debe realizar de manera que el cable negro quede en uno de los extremos del comprobador tal y como se observa en la imagen. Por último, se posiciona la batería en su soporte y se conecta ésta al Power-Module a través del conector XT-60 como se muestra en las figuras.

a)Conexión del Comprobador de voltaje a la batería



(b) Posicionamiento de la batería

(c) Conexión de la batería al Power-Module

Figura I - 18: Montaje de la batería.

El resultado final del montaje se puede ver en las fotografías siguientes:

Figura I - 19: montaje final

Anexo II

Ubuntu

Introducción

En este capítulo hablaremos sobre Ubuntu. El sistema operativo que he usado para desarrollar este TFG, hablaremos sobre él entrando en diversos aspectos como, por ejemplo, que es y cómo se instala.
Además del tipo de Ubuntu que hemos usado, las distintas situaciones que se han producido en el proceso de instalación o de su actualización.

¿Qué es Ubuntu?

Ubuntu es un sistema operativo de software libre y código abierto. Es una distribución de Linux basada en Debian. Actualmente corre en computadores de escritorio y servidores , en arquitecturas Intel, AMD y ARM (13, s.f.).
Está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario. Está compuesto de múltiple software normalmente distribuido bajo una licencia libre o de código abierto. Estadísticas web sugieren que la cuota de mercado de Ubuntu dentro de las distribuciones Linux es, aproximadamente, del 52 %, con una tendencia a aumentar como servidor web.

Su patrocinador, Canonical, es una compañía británica propiedad del empresario sudafricano Mark Shuttleworth. Ofrece el sistema de manera gratuita, y se financia por medio de servicios vinculados al sistema operativo y vendiendo soporte técnico.

Además, al mantenerlo libre y gratuito, la empresa es capaz de aprovechar los desarrolladores de la comunidad para mejorar los componentes de su sistema operativo. Extraoficialmente, la comunidad de desarrolladores proporciona soporte para otras derivaciones de Ubuntu, con otros entornos gráficos, como Kubuntu,

Xubuntu, Ubuntu MATE, Edubuntu, Ubuntu Studio, Mythbuntu, Ubuntu GNOME y Lubuntu.
Cada seis meses se publica una nueva versión de Ubuntu. Esta recibe soporte por parte de Canonical durante nueve meses por medio de actualizaciones de seguridad, parches para bugs críticos y actualizaciones menores de programas. Las versiones LTS (Long Term Support), que se liberan cada dos años, reciben soporte durante cinco años en los sistemas de escritorio y de servidor.

Procedimientos de la Instalación

Para la instalación lo primero que hacemos es ver que Ubuntu necesitamos, por tanto, lo que hay que hacer a continuación es averiguar qué versión de ROS necesitamos para Desarrollar este TFG, cuando el dron original, Erle-copter, era en un primer momento el que se iba a utilizar, la versión que necesitábamos era ROS Indigo, entonces, el Ubuntu que le correspondería y por tanto deberíamos instalar seria Ubuntu 14 Trusty Tahr (14, s.f.).

Sin embargo, para el dron definitivo, AR dron, con el que hemos hecho la puesta a punto, aunque en teoría usa ros indigo también, en la practica la versión que mejor funciona y la que vamos a usar es ROS Kinetic, por tanto, el Ubuntu 14 no es válido ya que la versión del sistema operativo que le corresponde es Ubuntu 16 Xenial Xerus.

Figura II - 1: Ubuntu

4.3.1 Actualización

El proceso que he seguido en este proyecto fue instalar Ubuntu 14, por que aún no era consciente de que no se iba a usar, y luego lo actualicé a Ubuntu 16, para ello tuve que eliminar toda instalación de ros indigo que había realizado, la actualización es sencilla de realizar conociendo este detalle, únicamente es necesario ir a "Opciones" y allí buscar donde están el "Software y Actualizaciones", lo que nos interesa a nosotros es configurar las actualizaciones (15, s.f.).
Para ello, haremos clic sobre la pestaña "Actualizaciones" y aquí podremos ver todas las opciones que podemos configurar relacionadas con las actualizaciones, vamos a encontrar una opción en esta misma ventana que nos permitirá elegir qué queremos hacer con las nuevas versiones de Ubuntu. Podemos recibir avisos cada vez que haya una nueva versión, ver avisos solo con las versiones LTS o no mostrar avisos nunca, le decimos que nos avise y nos saldrá una ventana para actualizar.

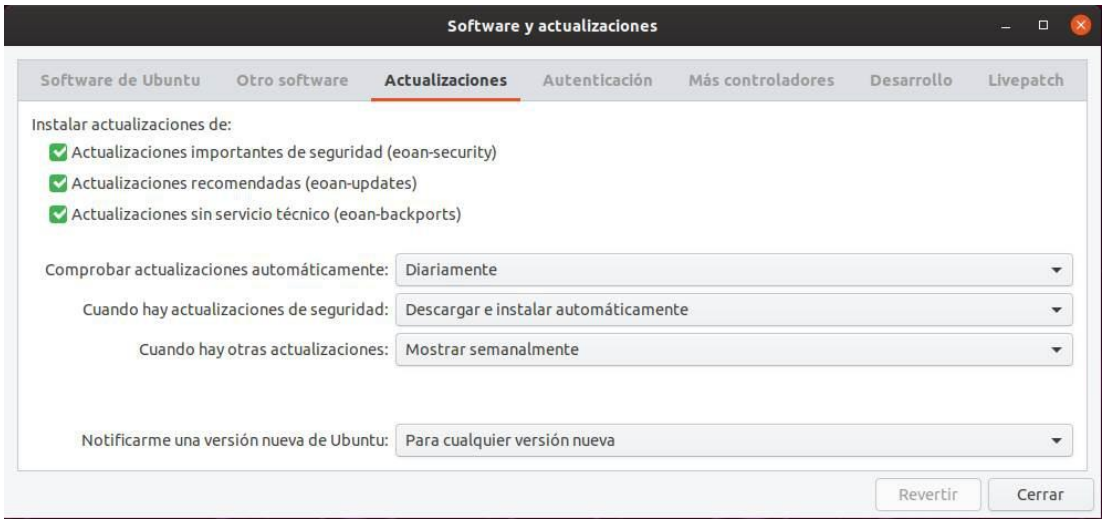


Figura II - 1: Software y actualizaciones

4.3.2 Instalación

Si hubiese instalado Ubuntu 16 desde el comienzo, el procedimiento que hubiese realizado es el mismo que he hecho al instalar Ubuntu 14 (16, s.f.).
A continuación, veremos cómo se hace la instalación en ambos casos de Windows y Linux en el mismo ordenador, dualboot, mediante los siguientes pasos:

Partición del disco duro

Lo primero es hacer una partición en el disco duro para ello Lo primero que tienes que hacer es abrir el menú de inicio y escribir partición, de manera que este te sugiera directamente la herramienta Crear y formatear particiones del disco duro. Cuando la veas haz click sobre ella, y se abrirá el programa de administración de discos.

- Para hacer una nueva partición vas a tener que hacerle hueco. Por eso, haz click derecho sobre el disco duro en el que quieras hacer la partición (1), y cuando te aparezca el menú desplegable, elige la opción Reducir Volumen (2) para hacer más pequeña la primera partición del disco. También puedes Eliminar volumen para borrar una partición y hacer hueco así a la nueva, o incluso Formatear el disco para limpiarlo antes de empezar a particionar.

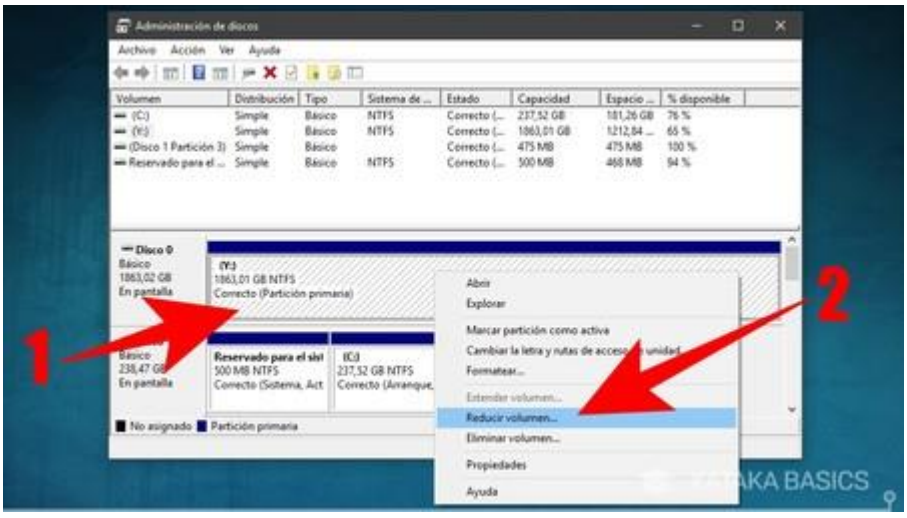


Figura II - 1: Reducir Volumen

- Si eliges reducir el volumen o partición, te aparecerá una ventana en la que tienes que poner cuánto espacio quieres dejar libre en el campo Tamaño del espacio que desea reducir. La cantidad la tienes que poner en MB, y es importante que te fijes en el espacio disponible a la hora de tomar la decisión.

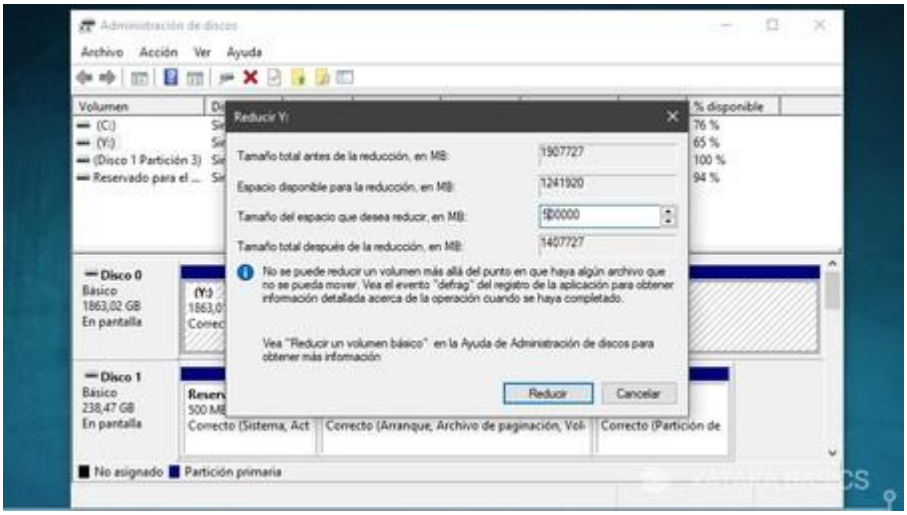


Figura II - 2: Reducir

- Cuando termine el procedimiento te aparecerá en negro una zona llamada No Asignado, que es la capacidad del disco duro disponible para crear nuevas particiones. Haz click derecho sobre este espacio.

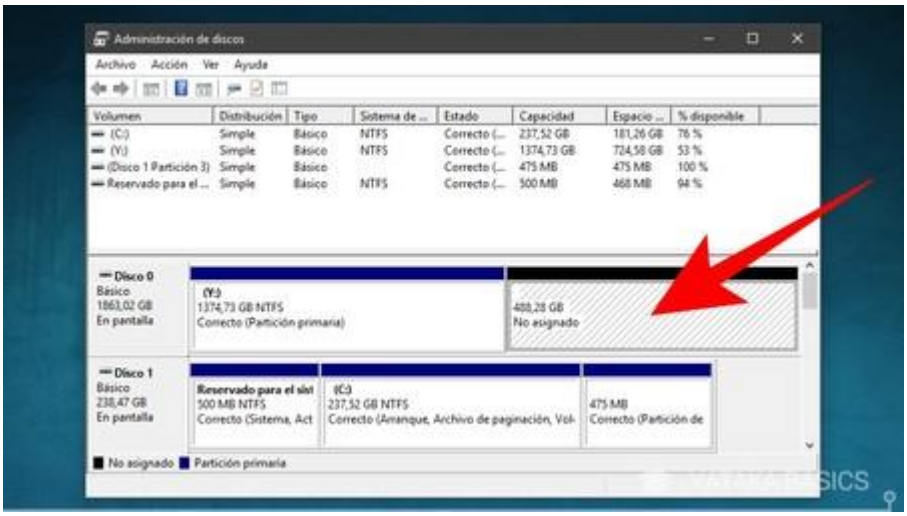


Figura II - 3:Espacio No Asignado

- Al hacer click derecho sobre el espacio libre del disco duro te aparecerá una ventana emergente. En ella pulsa sobre la opción Nuevo volumen simple para abrir un asistente que te guiará en el proceso de crear la nueva partición.

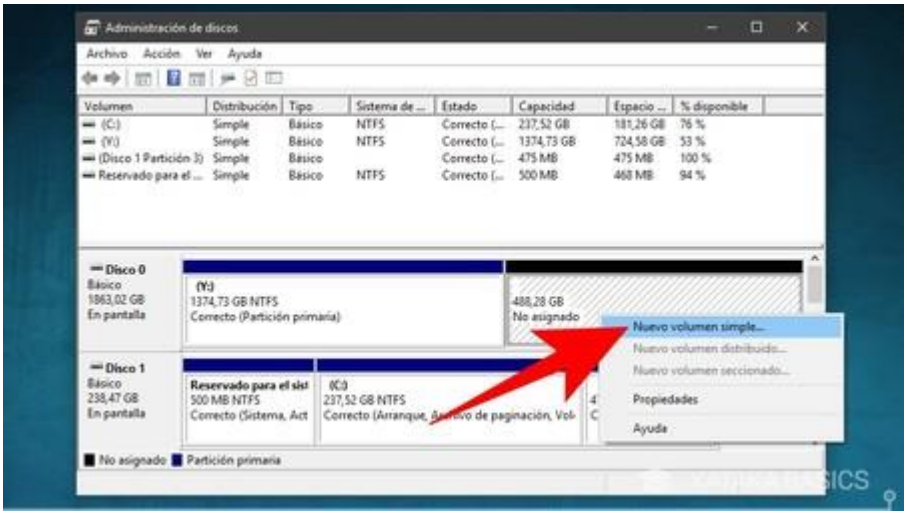


Figura II - 4:Nuevo Volumen Simple

- Tras pulsar Seguir en la ventana de introducción del asistente, irás a una nueva ventana en la que tienes que elegir el tamaño de tu nueva partición. Automáticamente el sistema te pondrá todo el espacio disponible, pero lo podrás editar para elegir una cantidad de MB que tú quieras. Cuando lo tengas, pulsa Siguiente.

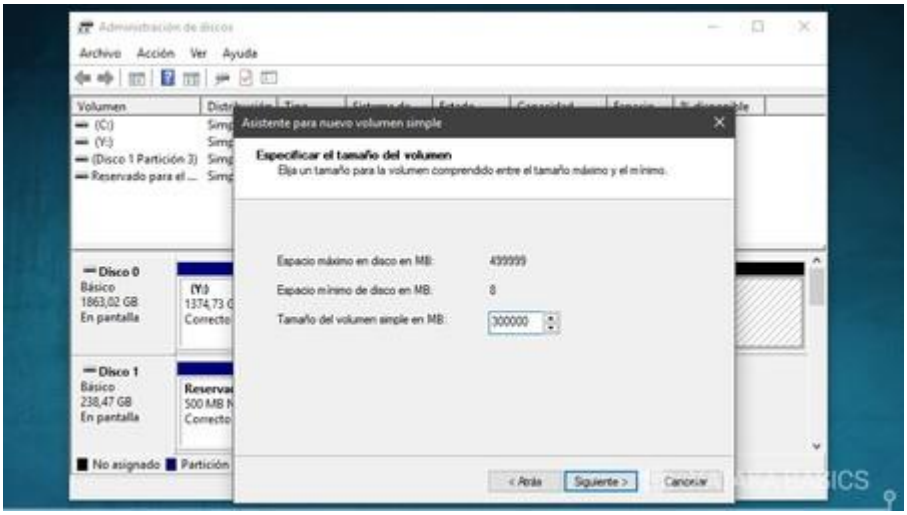


Figura II - 5: Tamaño De Partición

- Como Windows no concibe que quieras utilizar una partición para instalar otro sistema operativo, en el siguiente paso te pedirá que le añadas una letra a la unidad para que la partición haga de unidad de almacenamiento extra. Sin embargo, también tienes la opción No asignar una letra o ruta de acceso de unidad para que se cree una partición, pero Windows no empieza a utilizarla como unidad de almacenamiento.

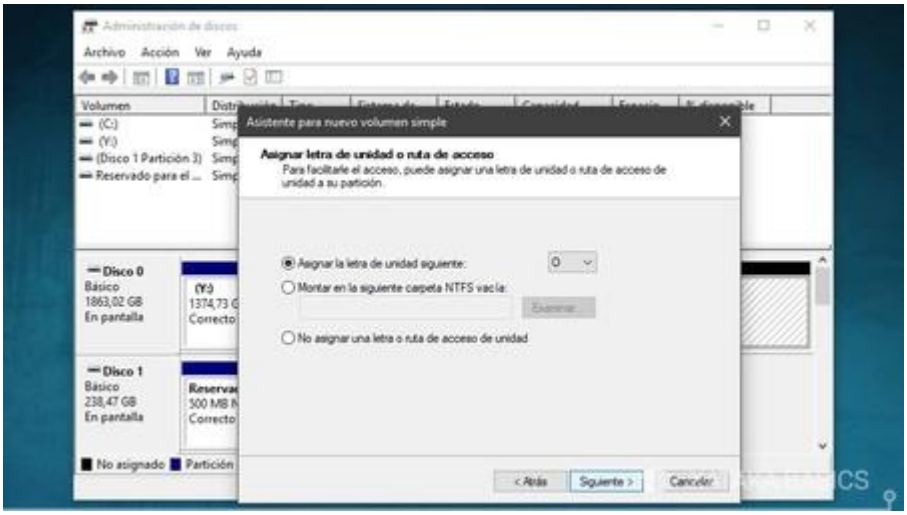


Figura II - 6: Elegir Letra

- En el siguiente paso tienes que elegir si quieres formatear la partición, lo cual es recomendable. También podrás elegir el sistema de archivos que quieres utilizar en la partición o el tamaño de la unidad. También tendrás un hueco en el que escribir un nombre para hacer reconocible la partición.

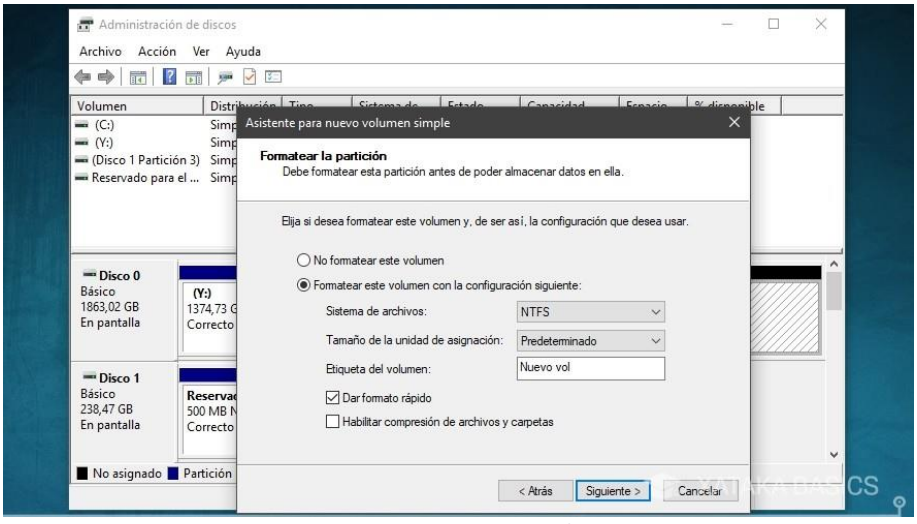


Figura II- 7: Sistema De Archivos

- Y, por último, llegarás a una pantalla final en la que se te resumirá la configuración de tu nueva partición. Pulsa Finalizar para aceptar y que Windows cree tu nueva partición, y si quieres cambiar algo pulsa Atrás.

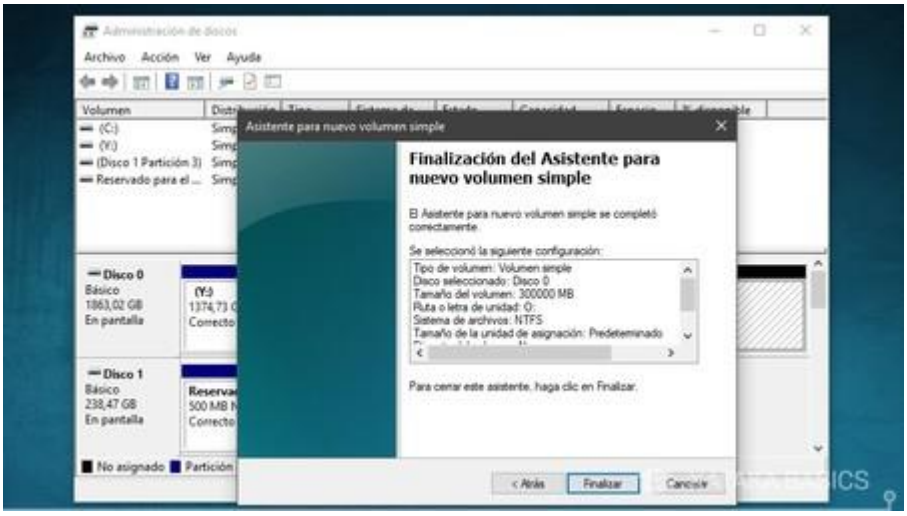


Figura II - 8: Ultimo Paso

Descargar Ubuntu

Lo primero será acudir al sitio web de ubuntu para obtener una copia del sistema en formato .ISO, que podremos seguidamente grabar en un DVD o preparar un pendrive de instalación que es lo que haremos y veremos a continuación.

http://www.ubuntu.com/download/desktop

En esta página nos detalla lo imprescindible que necesita nuestro ordenador para que Ubuntu 16.04 LTS funcione de forma óptima, los requisitos mínimos que debe cumplir nuestro ordenador para poder ejecutar Ubuntu 16.04 de forma correcta, no son especialmente altos y prácticamente cualquier ordenador con 10 años o menos va a ser apto.

Requisitos mínimos:

- Procesador Dual Core.
- 2GB de memoria RAM.
- 16GB de disco duro.
- Acceso a internet.
- Pendrive de 2 GB o más que usaremos como medio de instalación.

- El siguiente paso es descargar una imagen ISO de Ubuntu 16.04 LTS, puedes hacerlo desde los siguientes enlaces. Recuerda elegir la versión de 32 bits o 64 bits según las características de tu procesador, elegiremos 64 bits y Haremos clic en Download.

Figura II - 9: web de Ubuntu 16

- A continuación, vamos a preparar un pendrive para instalar nuestro sistema Ubuntu 16.04, lo que haremos será usar la aplicación YUMI para generar un medio de instalación en nuestro pendrive con una capacidad de al menos 2 GB.

- Preparamos nuestro PC para que arranque desde el pendrive. Normalmente se accede a la BIOS y se modifica el apartado Boot Sequence, para seleccionar el medio de arranque solo hay que pulsar F12 o esc, dependiendo del ordenador puede que sea incluso otra tecla, repetidamente al encender nuestro ordenador y nos aparecerá el menú de selección de medio donde elegiremos nuestro pendrive.

Preparación del pendrive

Una vez hecha la descarga de una imagen ISO de Ubuntu 16.04 LTS, continuamos preparando un pendrive para instalar nuestro sistema Ubuntu 16.04

Lo que haremos será usar la aplicación rufus para generar un medio de instalación en nuestro pendrive con una capacidad de al menos 2 GB.

- En primer lugar, necesitamos descargar la herramienta rufus para preparar nuestro Pendrive, una vez más se trata de una aplicación gratuita y descargarla en su página web. No necesitarás instalar nada, ya que se iniciará automáticamente. Ahora, conecta al ordenador el USB que quieras utilizar, y comprueba que el USB aparece en el campo Dispositivo (1) que tienes arriba del todo. Ahora pulsa sobre la opción Seleccionar (2) para elegir la imagen ISO con la que quieres crear el USB de arranque.

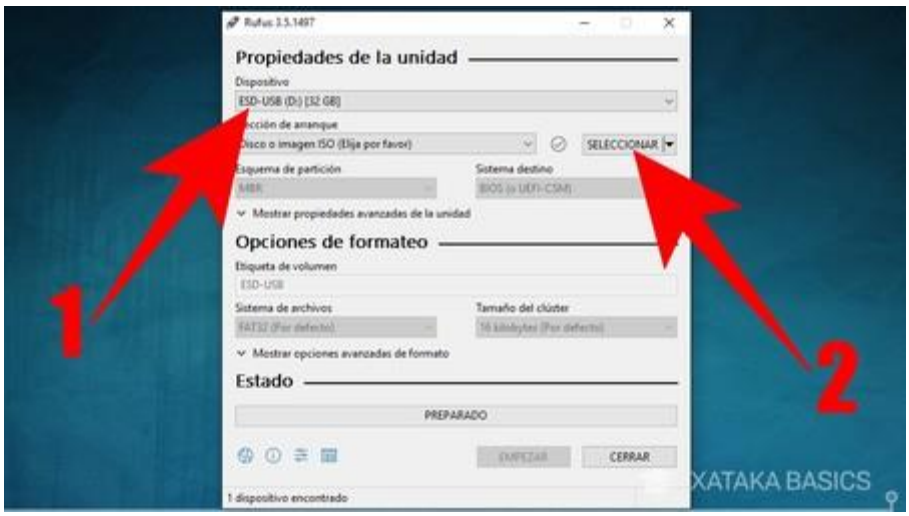


Figura II - 10: Rufus

- Cuando pulses en Seleccionar, se abrirá un explorador de archivos. En él tienes que buscar y seleccionar el archivo .ISO de la distro que hayas descargado, y pulsar el botón Abrir para que quede seleccionado en Rufus.

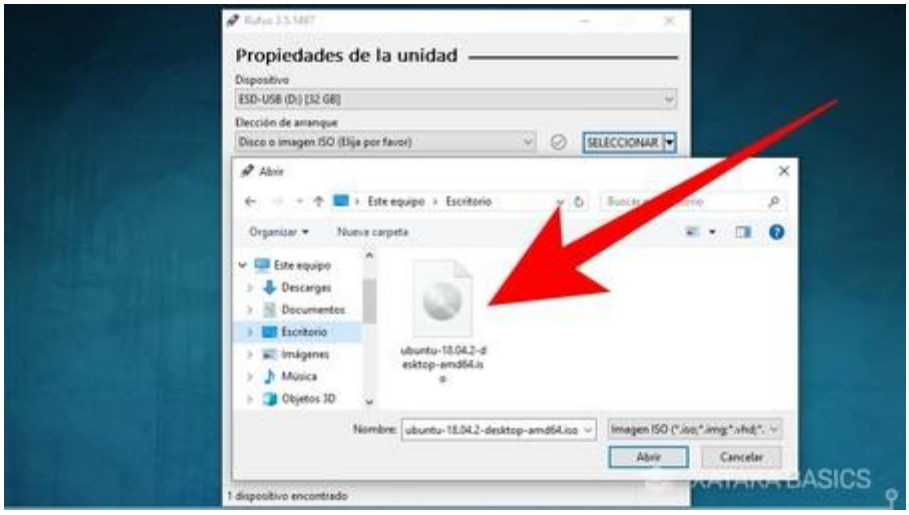


Figura II - 11: seleccionar ISO

- Una vez hayas seleccionado el archivo ISO, el resto de las opciones por defecto son las correctas para prácticamente cualquier caso u ordenador. Por lo tanto, a no ser que tengas unos conocimientos avanzados y quieras cambiar algo por las especificaciones concretas de tu ordenador, con dejarlo todo como está y pulsar el botón Empezar ya es suficiente para crear tu USB.

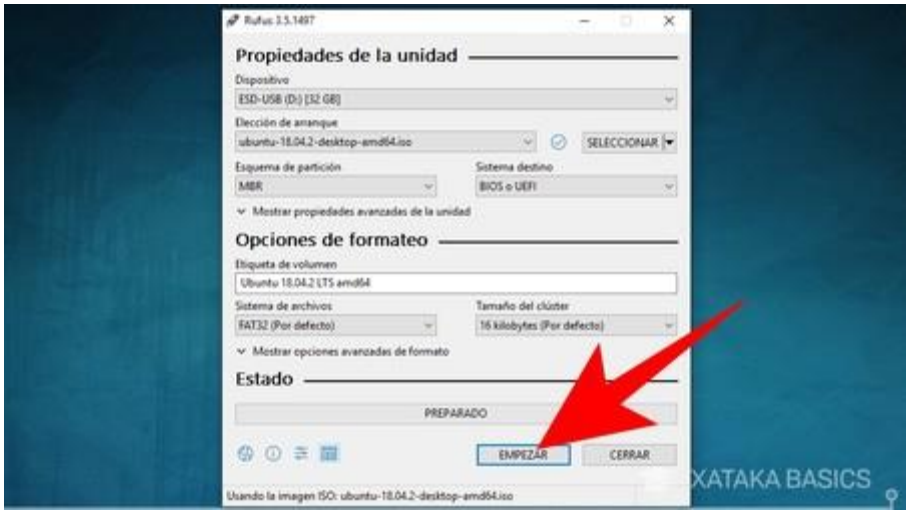


Figura II - 12: crear USB

- Cuando pulses en Empezar, Rufus te lanzará un aviso diciéndote que la versión del gestor de arranque syslinux que utiliza es más antigua que la que solicita la ISO. Por lo que debes pulsar el botón Sí para que Rufus se conecte a Internet y descargue automáticamente la versión que necesita.

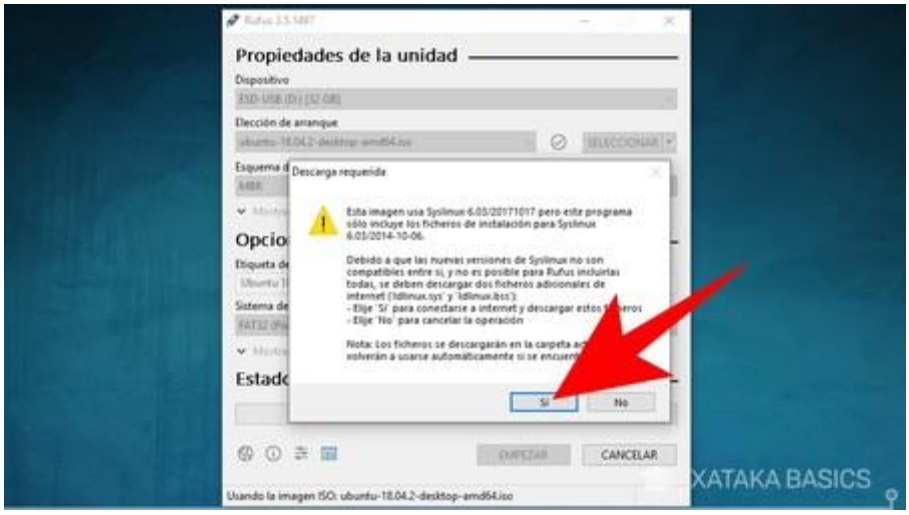


Figura II - 13: Actualizar Rufus

- Tras ese trámite, te aparecerá otra ventana en la que se informa de que la ISO que has descargado puede ser escrita de dos maneras en tu USB. Aquí, lo recomendable es que dejes seleccionada la opción Escribir en modo Imagen ISO y pulsas el botón OK.

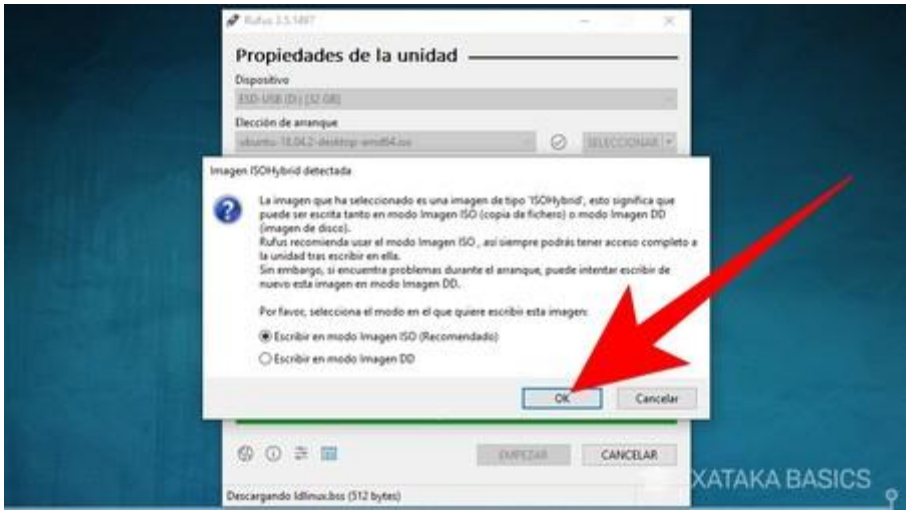


Figura II - 14: Escribir ISO

- Y, por último, Rufus te advertirá de que al realizar este proceso perderás todos los datos que tengas en el USB que vayas a utilizar. Si estás conforme, pulsa en el botón Aceptar y se empezará a preparar el USB de arranque. Espera a que termine, y una vez se complete el proceso ya podrás sacar el USB y arrancar con él el nuevo ordenador.

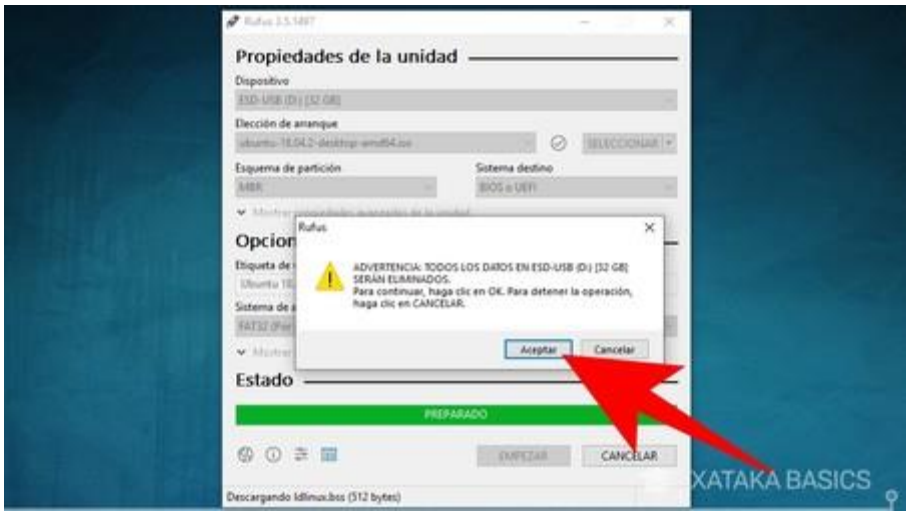


Figura II - 15: USB de arranque

Instalación Ubuntu

Una vez que tenemos listo nuestro pendrive para instalar Ubuntu 16.04 LTS en nuestro ordenador tan solo nos queda reiniciar el PC con el conectado y usar lo como medio de inicio. Cuando veas el menú, selecciona la unidad del USB de arranque y pulsa Enter para arrancar el ordenador a través de él.

Figura II - 16: select boot

El proceso de instalación puede variar dependiendo de la distribución, aunque muchas de ellas comparten las mismas líneas generales. En cualquier caso, nosotros nos vamos a ceñir al proceso de Ubuntu, que es el mismo que verás en muchas distribuciones basadas en Debian y en ella misma.

- La instalación comienza con una pantalla de bienvenida que a su vez hace la función de asistente de instalación.

Figura II - 17: Asistente

- Después debemos seleccionar el idioma a usar y pulsar en “instalar Ubuntu”.

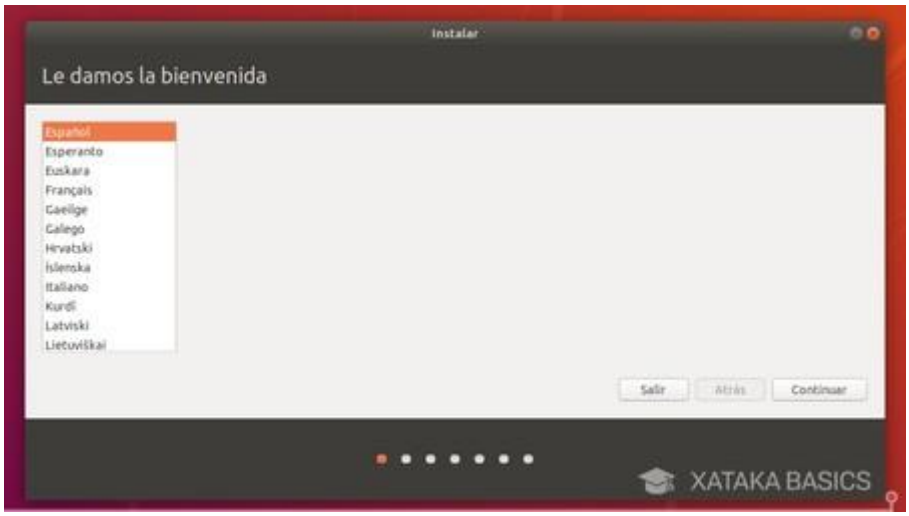


Figura II - 18: Idioma

- Ahora, llegarás a una pantalla en la que vas a poder conectarte a Internet configurando tu WiFi, yo he selecionado no conectarme debido a que por cuestion de que el sistema operativo es antiguo y mi ordenador tiene poco tiempo la tarjeta del wifi no la detecta y he necesitado una antena wifi externa para poder hacer las actualizaciones, por tanto, no nos da la opción de descargar las actualizaciones e instalar software de terceros para reproducir archivos multimedia y otros. Es recomendable marcar ambas opciones para que nuestro sistema Ubuntu esté actualizado.

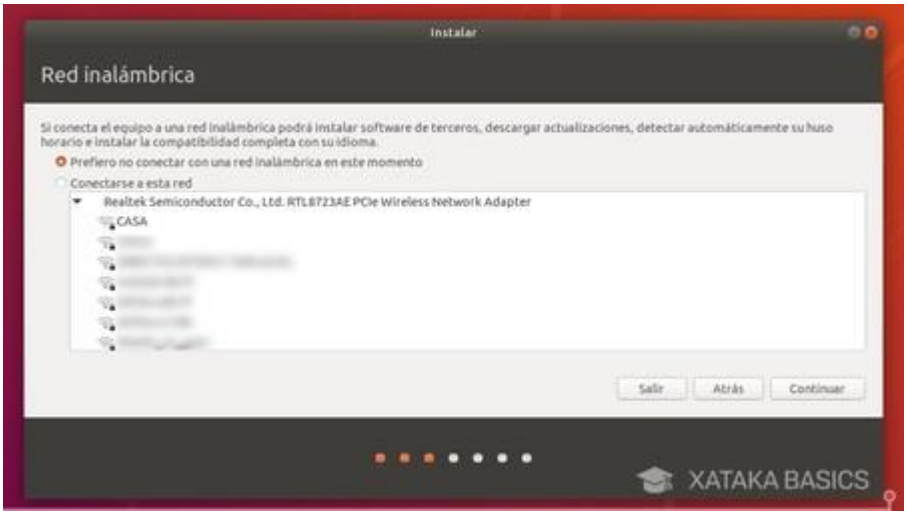


Figura II - 19: Red inalámbricas

- El siguiente paso vas a tener que elegir cómo quieres que sea la instalación. Lo recomendable es elegir la instalación normal.

Figura II - 20: Actualizaciones

- Y ahora llegamos a la parte importante, ya que tienes que decidir cómo quieres hacer la instalación. Ubuntu detectará que tienes Windows, o sea que es suficiente con elegir la opción de Instalar Ubuntu junto a Windows 10, y el sistema de instalación se encargará de hacerlo sin problemas. De hecho, al confirmar esta selección la instalación comenzará automáticamente.

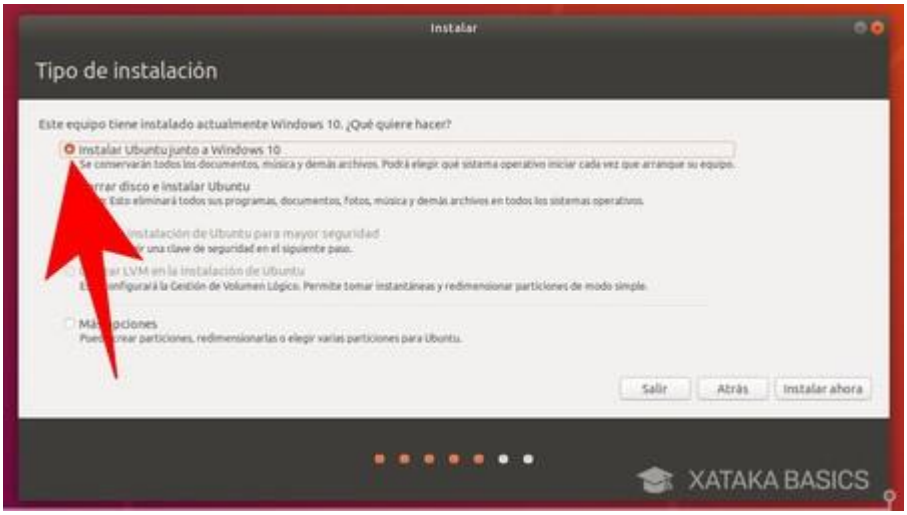


Figura II - 21: Instalación junto a Windows

Debemos aceptar el mensaje de confirmación.

- Seleccionamos nuestra zona horaria y pulsamos en “continuar”.

Figura II- 22: Zona horaria

- Elegimos nuestra distribución del teclado y “continuar“. Existe un espacio para que podamos escribir y comprobar que hemos elegido la opción correcta.

Figura II - 23: Teclado

- Luego nos saldrá una última pantalla en la que deberemos configurar un nombre de usuario y contraseña para crear tu primer usuario en el equipo. A este, también le puedes poner tu nombre real, y puedes elegir otro nombre para el ordenador. La contraseña es obligatoria, pero tras ponerla puedes marcar la casilla de Conectarse automáticamente sin pedir la contraseña para que no tengas necesidad de utilizarla. Cuando lo hagas, pulsa Siguiente.

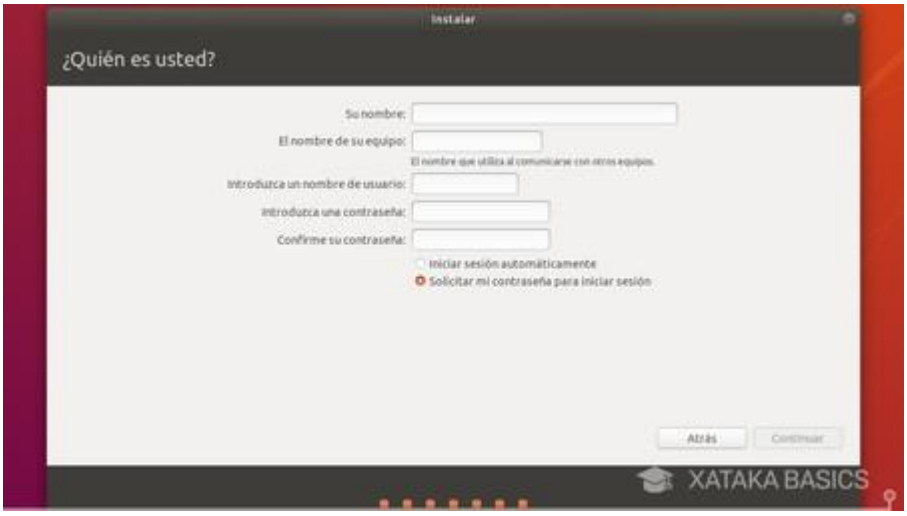


Figura II- 24: Usuario

- Y una vez hayas dado estos dos últimos pasos, la distribución que hayas elegido empezará a instalarse junto a Windows 10.

Figura II - 25: Bienvenido

- Finalmente reiniciamos nuestro sistema, y ya podremos acceder al escritorio de Ubuntu.

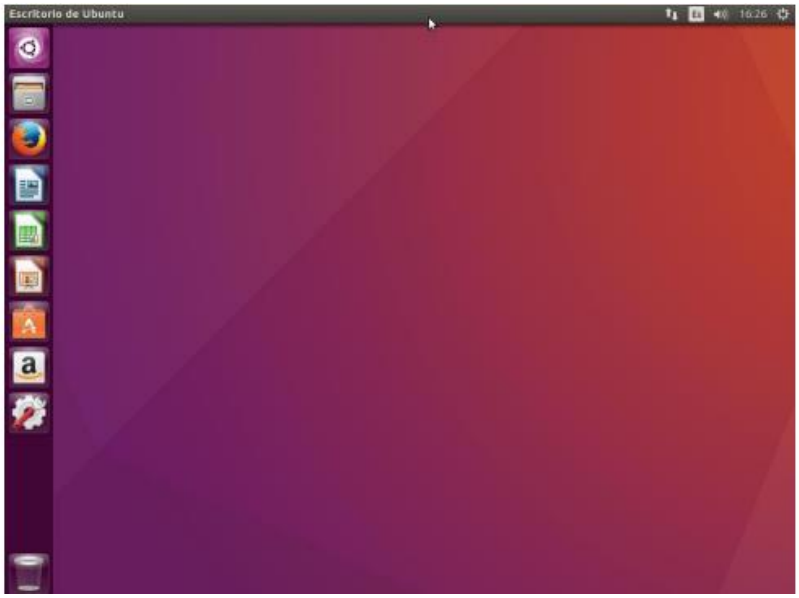


Figura II - 26: Escritorio

- Si todo ha ido bien, la próxima vez que enciendas el ordenador aparecerá una pantalla en la que podrás elegir qué sistema operativo iniciar, lo que te permitirá utilizar sin problemas tanto Windows como Ubuntu u otra distro que hayas escogido.

Anexo III

Instalación de ROS

Instalación

Las instrucciones que indico a continuación pueden encontrarse en la web de ROS, ROS Kinetic solo es compatible con Wily (Ubuntu 15.10), Xenial (Ubuntu 16.04) y Jessie (Debian 8) para paquetes de Debian (17, s.f.).

Configure sus repositorios de Ubuntu

Configure sus repositorios de Ubuntu para permitir "restringido", "universo" y "multiverso". Puede seguir la guía de Ubuntu para obtener instrucciones sobre cómo hacerlo.

Configura tu sources.list

configuramos nuestro PC para que reciba paquetes y acepte el software de packages.ros.org.
Dicho lo cual, vamos a abril un Terminal y tecleando:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb
_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Configura tus llaves

A continuación, establecemos nuestras “llaves” para asegurarnos que descargamos el software sin fraudes:

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' -
-recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Instalación

Primero, nos aseguramos de que la lista de paquetes Debian esté actualizado:

```
sudo apt-get update
```

Hay muchas bibliotecas y herramientas diferentes en ROS. Proporcionamos cuatro configuraciones predeterminadas para que pueda comenzar. También puede instalar paquetes ROS individualmente.

Una vez actualizada la lista, descargamos la versión completa de ROS

- **Instalación completa de escritorio: (Recomendado)** : ROS, rqt , rviz , bibliotecas genéricas de robots, simuladores 2D / 3D, navegación y percepción 2D / 3D

```
sudo apt-get install ros-kinetic-desktop-full
```

Configuración del entorno

Establecemos las variables de entorno para que no haga falta hacerlo cada vez que abrimos una ventana de Terminal, es conveniente si las variables de entorno ROS se agregan automáticamente a su sesión de bash cada vez que se lanza un nuevo shell:

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Si tiene más de una distribución ROS instalada, ~ / *.bashrc* solo debe obtener *setup.bash* para la versión que está utilizando actualmente.
Si solo desea cambiar el entorno de su shell actual, en lugar del anterior, puede escribir:

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

Dependencias para construir paquetes

Hasta ahora, ha instalado lo que necesita para ejecutar los paquetes ROS principales. Para crear y administrar sus propios espacios de trabajo ROS, existen varias herramientas y requisitos que se distribuyen por separado.

Por ejemplo, rosinstall es una herramienta de línea de comandos de uso frecuente que le permite descargar fácilmente muchos árboles de origen para paquetes ROS con un solo comando.

Para instalar esta herramienta y otras dependencias para construir paquetes ROS, ejecute:

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
```

Inicializar rosdep

Antes de que pueda utilizar muchas herramientas ROS, deberá inicializar rosdep y actualizarlo. rosdep le permite instalar fácilmente las dependencias del sistema para la fuente que desea compilar y es necesario para ejecutar algunos componentes centrales en ROS. Si aún no ha instalado rosdep , hágalo de la siguiente manera.

```
sudo apt install python-rosdep
```

Con lo siguiente, puede inicializar rosdep .

```
sudo rosdep init
rosdep update
```