Crazyflie 2.1 - ROS/ROS2 (`uned_crazyflie_ros_pkg`)

Francisco J. Mañas-Álvarez

28 de agosto de 2021

Índice

1. Introducción

Este repositorio[1] contiene los paquetes de ROS y ficheros de configuración para la teleoperación y simulación del dron crazyflie 2.1 en ROS/ROS2, Gazebo y Matlab. Se busca obtener una herramienta Hardware-in-the-Loop que sea fácilmente escalable y mantenible. A partir de esta herramienta, se pretende integrar estos robots en un sistema distribuido mayor con más variedad de robots. El proyecto se inició en octubre de 2021 en ROS (*ros-noetic*) mientras que en agosto de 2021 se procede a su actualización a ROS2 (*ros2-galactic*). Se hace un esfuerzo por mantener ambas plataformas en ramas distintas de forma de obtener la mayor versatilidad posible.

1.1. Estructura

- **doc.** Contiene un fichero *.tex* que aborda más en detalle toda la información relacionada con el repositorio: esquemas de ROS/ROS2, búsquedas bibliográficas, enlaces de interés, etc. Se persigue reunir toda la información recogida en un manual para que cualquier usuario pueda ponerse al día de la forma más autónoma posible.
- **scripts.** Contiene aquellos ficheros auxiliares que no forman parte de ningún paquete de ROS/ROS2. Por ejemplo, ficheros *.sh* para automatizar procesos repetitivos como la conversión de los ficheros *.bag* a txt o scripts de Matlab/Simulink para visualización de datos y simulaciones.
- **submodules.** En este directorio están vinculados otros repositorios que se reutilizan, o se toman de base, para tareas ya abordadas por otros usuarios. Si no hay dependencias con otros repositorios, esta carpeta no se generará.
- **uned_crazyflie_config.** Paquete de ROS/ROS2. Contiene aquellos elementos auxiliares para la configuración del entorno, así como los *launch* para la ejecución en bloque de las diferentes estructuras del sistema.
- **uned_crazyflie_drone.** Paquete de ROS/ROS2. Comprende los nodos propios desarrollados para la integración de drones en el sistema. Incluye toda la información asociada para su correcta puesta en marcha. Aún no se dispone de una versión para ROS. En el caso de ROS2 se está trabajando para hacerlo a través de microROS, sistema basado en ROS2. MicroROS está especialmente diseñado para este tipo de robots y ya hay un caso aplicado al crazyflie. El problema es que está diseñado para FreeRTOS.
- **uned_crazyflie_test.** Paquete de ROS/ROS2. Paquete en el que se incluyen todos los elementos destinados a realizar comprobaciones en el sistema de forma rápida. Por ejemplo los nodos *talker* y *listener* que se desarrollan al empezar a usar ROS/ROS2, que en este caso se usan para comprobar la correcta comunicación entre máquinas en el sistema distribuido.

2. Crazyflie

El dron crazyflie 2.1, figura 1, se trata de un cuadricóptero que por su reducido tamaño entra dentro de la categoría de "nano-cuadricóptero". Su diseño está especialmente pensado para vuelo en interiores. A continuación se detallan sus especificaciones técnicas, disponibles también en la web del dron[2].

[1] Github: `https://github.com/FranciscoJManasAlvarez/uned_crazyflie_ros_pkg`
[2] Bitcraze: `https://store.bitcraze.io/products/crazyflie-2-1`

Figura 1: Crazyflie 2.1.

2.1. Características

2.1.1. Especificaciones mecánicas

- Peso: $27g$
- Dimensiones: $92x92x29mm$

2.1.2. Radio

- 2.4GHz ISM band radio
- Increased range with 20 dBm radio amplifier, tested to > 1 km range LOS with Crazyradio PA (environmentally dependent)
- Bluetooth Low Energy support with iOS and Android clients available
- Dual antenna support with both on board chip antenna and U.FL connector

2.1.3. Microcontroladores

- STM32F405 main application MCU (Cortex-M4, 168MHz, 192kb SRAM, 1Mb flash)
- nRF51822 radio and power management MCU (Cortex-M0, 32Mhz, 16kb SRAM, 128kb flash)
- uUSB connector
- On-board LiPo charger with 100mA, 500mA and 980mA modes available
- Full speed USB device interface
- Partial USB OTG capability (USB OTG present but no 5V output)
- 8KB EEPROM

2.1.4. IMU

- 3 axis accelerometer / gyroscope (BMI088)
- high precision pressure sensor (BMP388)

2.1.5. Especificaciones de vuelo

- Tiempo de vuelo con batería completa: 7 minutos
- Tiempo de carga hasta batería completa: 40 minutos
- Máxima carga recomendada: 15g

2.1.6. Conectores

- VCC (3.0V, max 100mA)
- GND
- VCOM (unregulated VBAT or VUSB, max 1A)
- VUSB (both for input and output)
- I2C (400kHz)
- SPI
- 2 x UART
- 4 x GPIO/CS for SPI
- 1-wire bus for expansion identification
- 2 x GPIO connected to nRF51

2.2. Extras

2.2.1. Motion capture marker deck

TO-DO

2.2.2. Multi-ranger deck

TO-DO

2.2.3. Carga inalámbrica

TO-DO

2.3. Proyectos anteriores

En este apartado se describen los trabajos ya realizados sobre crazyflie que se han consultado y reutilizado parcialmente para la herramienta desarrollada. Ampliar con imágenes y referencias a sus publicaciones, no sólo sus webs.

CrazyS [3]

Se trata de un proyecto basado en el paquete RotorS [4]. Su finalidad es obtener un simulador en Gazebo/ROS para los crazyflie 2.1. En la versión la versión para ROS Noetic de *uned_crazyflie_ros_pkg* está basada en este repositorio. Los parámetros empleados presentan discrepancias respecto a los empleados en otras publicaciones (dinámica más lenta). Todos los parámetros son configurables desde un fichero *.xacro*. TO-DO

[3] Github: `https://github.com/gsilano/CrazyS`
[4] Github: `https://github.com/ethz-asl/rotors_simulator`

Crazyflie_ros [5]

Este repositorio ha quedado archivado por el autor y ahora todos los esfuerzos se centran en el el proyecto crazyswarm. TO-DO

Crazyswarm [6]

Este proyecto trabaja en ROS. Es un trabajo muy extendido y consolidado. Toda la información se encuentra disponible en `https://crazyswarm.readthedocs.io/en/latest/index.html`. TO-DO

Gym-pybullet-drones [7]

Este proyecto contempla trabajar en ROS2, no obstante, no emplea Gazebo como herramienta de simulación. TO-DO

OFERA [8]

Se trata de un proyecto europeo desarrollado entre 01/18-12/2020. Presenta el sistema micro-ROS, basado en ROS2. TO-DO

2.4. Modelado

El modelado se ha basado en el capítulo 4 del trabajo [6]. Para el modelado, se comprenderá la configuración del crazyflie en "X", ya que presenta menos inconvenientes en el caso de querer agregar sensores, como cámaras.

Para la obtención del modelo dinámico del sistema, se asumen tres hipótesis, que simplifican el modelo pero dan una buena aproximación para trabajar. Las simplificaciones sobre el dron son:

- Se comporta como un sólido-rígido sin deformaciones.
- Es simétrico, tanto en masa como en el sistema de propulsión (todos los motores presentan el mismo comportamiento).
- La masa es constante.

Para ubicar el marco de referencia del dron, se emplea la representación extrínseca de Tait-Bryan. La transformación desde el centro de referencia del sistema global al dron está determinada por tres rotaciones sucesivas: Yaw (ψ) $\rightarrow$ Pitch (θ) $\rightarrow$ Roll (ϕ). Las rotaciones en los ejes X e Z se toman en sentido horario mientras que el giro en Y se adopta con sentido antihorario [4]. En [3] se toman estas rotaciones de la misma forma, pero sus matrices de rotación individuales están mal indicadas. La matriz de rotación resultante sería:

$$R_x(-\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} R_z(-\psi) = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

$$R_o^b = \begin{pmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{pmatrix} \tag{2}$$

Al tratarse de una matriz ortonormal, se cumple la siguiente propiedad:

$$\left(R_o^b\right)^{-1} = \left(R_o^b\right)^T = R_b^o \tag{3}$$

[5] Github: `https://github.com/whoenig/crazyflie_ros`
[6] Github: `https://github.com/USC-ACTLab/crazyswarm`
[7] Github: `https://github.com/utiasDSL/gym-pybullet-drones`
[8] Github: `http://www.ofera.eu/index.php`

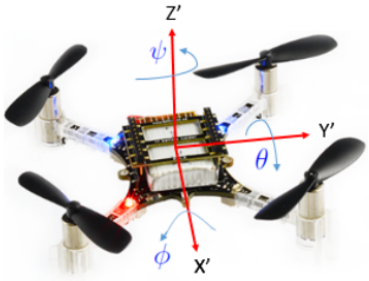

Figura 2: Ejes y ángulos.

A continuación se realiza el modelado basado en espacios de estado para un quadrotor genérico. El vector de estados está compuesto por cuatro vectores asociados a los 6 grados de libertad del dron y sus respectivas velocidades tal como se describe en la tabla 1.

Vector	Estado	Descripción
p	x	Posición X
	y	Posición Y
	z	Posición Z
Φ	ϕ	Ángulo Roll
	θ	Ángulo Pitch
	ψ	Ángulo Yaw
V	u	Velocidad Lineal X
	v	Velocidad Lineal Y
	w	Velocidad Lineal Z
ω	p	Velocidad Angular X
	q	Velocidad Angular Y
	r	Velocidad Angular Z

Tabla 1: Vector de estados.

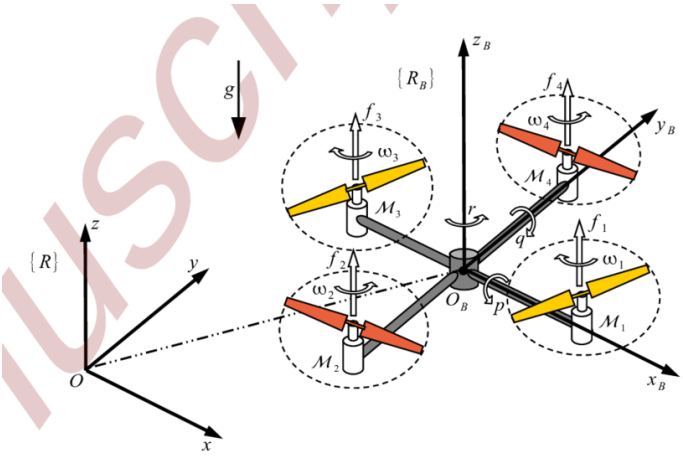

Figura 3: Esquema de un quadrotor.

2.4.1. Ecuaciones del equilibrio de fuerzas

Para determinar las ecuaciones relacionadas con los parámetros lineales del dron, se aplica la segunda ley de Newton, determinando la derivada de la velocidad según la ecuación de Coriolis[9] [1]

$$\Sigma F = m \cdot \dot{V}_{CG}^{o} = m \cdot \left(\dot{V}_{CG}^{b} + \omega_{b/o} \times V_{CG} \right) \tag{4}$$

Considerando que en estacionario, tanto el Roll como el Pitch son nulos, el equilibrio de fuerzas permite aislar la derivada de la velocidad lineal del dron.

$$\begin{bmatrix} 0 \\ 0 \\ F_z \end{bmatrix} - R_o^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = m \left(\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \right) \tag{5}$$

$$\boxed{\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{F_z}{m} \end{bmatrix} - R_o^b \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix}} \tag{6}$$

Para determinar las derivadas de los espacios de estado referentes a la posición, se proyectan las velocidades lineales del dron sobre el marco general mediante la correspondiente matriz de transformación, ecuación 2.

$$\dot{p}^o = R_b^o \cdot V^b \rightarrow \boxed{\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = R_b^o \begin{bmatrix} u \\ v \\ w \end{bmatrix}} \tag{7}$$

Para determinar la fuerza que generan las hélices del dron se emplea la siguiente expresión:

$$F_i^b = \begin{bmatrix} 0 \\ 0 \\ T_i \end{bmatrix} \rightarrow T_i = C_T \omega_i^2 \rightarrow \boxed{\Sigma F_i^b = \begin{bmatrix} 0 \\ 0 \\ C_T \left(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2 \right) \end{bmatrix}} \tag{8}$$

T_i es el empuje expresado en Newtons y C_T es el coeficiente de empuje que se determina mediante la siguiente expresión:

$$C_T = k_T \rho \frac{(2r)^4}{3600} \tag{9}$$

2.4.2. Equilibrio de Momentos

Para determinar los elementos asociados a los componentes angulares de los estados del sistema, se emplea el equilibrio de momentos. Aplicando la sumatoria de momentos equivalente al momento angular del dron usando la equivalencia de la ecuación de Coriolis:

$$\Sigma M^o = {}^o\dot{h} \rightarrow \Sigma M^o = {}^b\dot{h} + \omega_{b/o} \times h \tag{10}$$

Aplicado al marco del dron, las ecuaciones de cantidad de movimiento se calculan más fácilmente, como se explica en [7, 8]

$$\Sigma M^b = J^b \dot{\omega}_{b/o} + \omega_{b/o} \times J \omega_{b/o} \tag{11}$$

donde J representa la matriz de inercia del dron. Al asumirse la hipótesis de simetría del dron, la matriz sería:

$$J = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \rightarrow J = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{12}$$

[9] Explicación: `https://www.youtube.com/watch?v=-0yRCgv-hPs`

$$(J)^{-1} = \frac{adj\begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix}}{\begin{vmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{vmatrix}} = \frac{\begin{pmatrix} I_{yy}I_{zz} & 0 & 0 \\ 0 & I_{xx}I_{zz} & 0 \\ 0 & 0 & I_{xx}I_{yy} \end{pmatrix}}{I_{xx}I_{yy}I_{zz}} \tag{13}$$

$$(J)^{-1} = \begin{pmatrix} \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{pmatrix} \tag{14}$$

Por lo tanto, se puede realizar el despeje del término $\dot{\omega}_{b/o}$, obteniendo la siguiente expresión:

$$\dot{\omega}_{b/o} = \boxed{\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = (J)^{-1}\left(\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times J \begin{bmatrix} p \\ q \\ r \end{bmatrix}\right)} \tag{15}$$

La última ecuación de estado determina la relación entre el vector $\omega_{b/o}$ y la derivada de Φ.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \rightarrow \boxed{\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}} \tag{16}$$

Para determinar la correlación entre la velocidad de giro de las hélices y el momento resultante, el proceso es el siguiente:

$$M = \Sigma P_i \times F_i + \Sigma \tau_i \tag{17}$$

donde P_i representa la distancia de cada motor al centro de gravedad y τ_i representa el momento inducido en el dron por cada motor. Este momento es consecuencia de la tercera ley de newton aplicada al giro de la hélice.

Al emplear una configuración en "X", la posición de cada motor es:

$$P_1 = \begin{bmatrix} \frac{d}{\sqrt{2}} \\ -\frac{d}{\sqrt{2}} \\ 0 \end{bmatrix}; P_2 = \begin{bmatrix} -\frac{d}{\sqrt{2}} \\ -\frac{d}{\sqrt{2}} \\ 0 \end{bmatrix}; P_3 = \begin{bmatrix} -\frac{d}{\sqrt{2}} \\ \frac{d}{\sqrt{2}} \\ 0 \end{bmatrix}; P_4 = \begin{bmatrix} \frac{d}{\sqrt{2}} \\ \frac{d}{\sqrt{2}} \\ 0 \end{bmatrix} \tag{18}$$

Conociendo la fuerza que genera cada motor a partir de la ecuación 2.4.1, el momento producto de esa fuerza se puede determinar según la siguiente expresión:

$$P_1 \times F_1 = \begin{bmatrix} -\left(C_T\omega_1^2\right) d/\sqrt{2} \\ -\left(C_T\omega_1^2\right) d/\sqrt{2} \\ 0 \end{bmatrix} \quad P_2 \times F_2 = \begin{bmatrix} -\left(C_T\omega_2^2\right) d/\sqrt{2} \\ \left(C_T\omega_2^2\right) d/\sqrt{2} \\ 0 \end{bmatrix}$$
$$P_3 \times F_3 = \begin{bmatrix} \left(C_T\omega_3^2\right) d/\sqrt{2} \\ \left(C_T\omega_3^2\right) d/\sqrt{2} \\ 0 \end{bmatrix} \quad P_4 \times F_4 = \begin{bmatrix} \left(C_T\omega_4^2\right) d/\sqrt{2} \\ -\left(C_T\omega_4^2\right) d/\sqrt{2} \\ 0 \end{bmatrix}$$

Aplicando la conservación del momento angular:

$$\Sigma\tau_i^b = \begin{bmatrix} 0 \\ 0 \\ C_D\left(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2\right) \end{bmatrix} \tag{19}$$

Por lo tanto, en el cálculo de los momentos, el resultado sería:

$$M^b = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} dC_T/\sqrt{2}\left(-\omega_1^2 - \omega_2^2 + \omega_3^2 + \omega_4^2\right) \\ dC_T/\sqrt{2}\left(-\omega_1^2 + \omega_2^2 + \omega_3^2 - \omega_4^2\right) \\ C_D\left(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2\right) \end{bmatrix} \tag{20}$$

El coeficiente de torque, C_D, se determina según las expresiones siguientes como se especifica en [5]:

$$Q = k_D \rho n^2 D^5 \tag{21}$$

$$C_D = k_D \rho \left(2r\right)^5 / 3600 = 7{,}9379 \cdot 10^{-12} \left[Nm/rpm^2\right] \tag{22}$$

En la ecuación de la cantidad de movimiento total hay ciertos términos que incluyen aceleraciones angulares que se han omitido, ya que tienden a ser pequeños en comparación con los otros términos de la ecuación. Los momentos de los giroscópicos también se han omitido utilizando el argumento de que el momento de inercia de cada motor tiende a ser pequeño, por lo que su contribución en el momento total también es despreciable [9, 2].

2.4.3. Parámetros físicos

TO-DO: En esta sección se indicará una relación de aquellos parámetros empleados en las simulaciones para establecer un criterio fijo que permita comparar resultados entre herramientas.

3. Instalación

Se plantea inicialmente el desarrollo del sistema distribuido en ROS. El objetivo de desarrollo de la plataforma es implementar todo el sistema en ROS Noetic Ninjemys y Ubuntu 20.04 LTS (Focal Fossa) a fin de prolongar el mantenimiento y vigencia de la plataforma. Ambos tienen el mantenimiento previsto de 5 años. No obstante, el lanzamiento de estas versiones se ha realizado en el año 2020, por lo que, junto con el fin del mantenimiento de la versión 2.7 de Python, implica que mucho del material ya desarrollado para versiones anteriores, no se puede migrar con facilidad. Por tanto, se plantea la reutilización de gran parte del material ya disponible en la web, trabajando con parte del sistema en la actual y otra parte en la versión anterior, ROS Melodic Morenia y Ubuntu 18.04 LTS (Bionic Beaver).

En agosto de 2021, con las simulaciones operativas en ROS Noetic, se opta por llevar el sistema a ROS2 Galactic Geochelone. Se opta por intentar llevar a cabo un mantenimiento de ambas versiones para facilitar su uso entre los usuarios ya familiarizados con ROS. Al igual que con las versiones de Ubuntu, se presenta el problema de que muy pocos paquetes se encuentran disponibles en ROS2, por lo que el esfuerzo de mudanza es mayor. No obstante habilita la posibilidad de emplear el proyecto OFERA, micro-ROS, diseñado específicamente para microcontroladores, con un ejemplo específico para los crazyflie.

3.1. ROS

Lo primero debe ser tener instalada la correspondiente versión de ROS para el sistema operativo del dispositivo (Noetic, Melodic). La máquina donde se ejecuten los paquetes reutilizados debe trabajar con ROS Melodic. No hay problema de compatibilidad en la interconexión de distintas máquinas siempre que los topics no presenten incompatibilidades entre versiones.

3.1.1. rosbridge_suite

La conexión común de todos los componentes de la red de ROS se realiza a través del paquete rosbridge_suite, instalado mediante el comando sudo apt-get install ros-¡rosdistro¿-rosbridge-suite (debe estar instalado previamente ROS en el dispositivo). Para la correcta identificación y conexión de cada máquina, se debe configurar en cada una los parámetros *ROS_MASTER_URI* y *ROS_HOSTNAME*, dados por la ip de cada dispositivo.

```
sudo nano ~/.bashrc
...
commentstyleexport ROS_MASTER_URI = http://xxx.xxx.x.xx:11311
commentstyleexport ROS_HOSTNAME = xxx.xxx.x.xx
```

Para el lanzamiento del paquete, se emplea el comando

```
roslaunch rosbridge_server rosbridge_websocket.launch
```

3.1.2. Matlab

Matlab debe disponer del toolbox de ROS instalado. En este caso, se trabaja con la versión de Matlab 2020b.

Se deben configurar los parámetros *ROS_MASTER_URI* y *ROS_HOSTNAME* para que pueda conectarse a la red que se ejecute en el dispositivo principal (ROS_MASTER_URI) y sea identificado dentro de la red. Estas acciones se llevan a cabo en línea de comandos mediante las instrucciones:

```
setenv('ROS_MASTER_URI','http://192.168.1.xx:11311')
setenv('ROS_HOSTNAME','192.168.1.xx')
```

La versión de Python que emplea el toolbox de Matlab es la 2.7. En principio no supone un problema porque no influye en el desempeño del resto de la red de dispositivos. Se puede descargar esta versión desde la web oficial. Se debe configurar la versión de Python Matlab mediante el comando

```
pyversion folder
```

donde *folder* es el directorio donde se ha instalado previamente la versión de python. Esto se emplea para integrar posteriormente los mensajes no estándares que se emplean en el proyecto. El compilador que debe estar fijado en Matlab debe ser Microsoft Visual C++ 2017. Para realizar esta comprobación se puede ejecutar el comando

```
mex -setup cpp
```

Una vez asegurada la versión de python y del compilador se deben agregar las nuevas tipologías de mensajes al directorio de Matlab. Para ello, se deben ejecutar los siguientes comandos:

```
folderpath = 'C:\folder con los nuevos mensajes\'
rosgenmsg(folderpath)
addpath('C:\folder con los nuevos mensajes\matlab_msg_gen_ros1\win64\install\m')
savepath
clear classes
rehash toolboxcache
rosmsg list
```

Recordar ejecutar siempre el comando rosshutdown al final del script para evitar dejar el nodo en el aire y al proncipio por si se nos ha olvidado cerrarlo anteriormente, que no de problemas.

3.1.3. Dependencias

- **Octomap**. TO-DO: Especificar los paquetes que dependen

  ```
  sudo apt-get install ros-<rosdistro>-octomap
  ```

- **Xacro**. TO-DO: Especificar los paquetes que dependen

  ```
  sudo apt-get install ros-<rosdistro>-xacro
  ```

- **Joy**. Paquete para realizar la lectura del joystick para teleoperación.

  ```
  sudo apt-get install ros-<rosdistro>-joystick-drivers
  ```

- **mav_comm**. TO-DO. Este paquete se emplea como complemento a gran parte de los paquetes ya desarrollados de Crazyflie y es compatible con ambas versiones de ROS y Ubuntu por lo que se puede alojar en el espacio de trabajo del dispositivo y compilarlo como cualquier otro paquete.

3.1.4. Ubuntu 18.04 - ROS Melodic Morenia

A continuación se detalla la instalación en el entorno de trabajo de ROS para el paquete CrazyS, de donde se reutiliza gran parte de la arquitectura de simulación.

```
mkdir -p catkin_ws/src
cd crazyflie/src
catkin_init_workspace
cd ..
catkin init
cd src
git clone https://github.com/gsilano/CrazyS.git
git clone https://github.com/gsilano/mav_comm.git

rosdep install --from-paths src -i
sudo apt install ros-melodic-rqt-rotors ros-melodic-rotors-comm \
ros-melodic-mav-msgs ros-melodic-rotors-control

sudo apt install ros-melodic-rotors-gazebo ros-melodic-rotors-evaluation \
ros-melodic-rotors-joy-interface

sudo apt install ros-melodic-rotors-gazebo-plugins ros-melodic-mav-planning-msgs \
ros-melodic-rotors-description ros-melodic-rotors-hil-interface

rosdep update
catkin build

echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

En la sección de instalación de CrazyS se detalla este proceso y posibles soluciones en caso de fallos con gazebo (como que no se inicie la simulación).

3.1.5. Ubuntu 20.04 - ROS Noetic Ninjemys

La configuración del entorno de trabajo para el paquete desarrollado se muestra a continuación.

```
mkdir -p crazyflie_ws/src
cd crazyflie/src
git clone https://github.com/FranciscoJManasAlvarez/uned_crazyflie_ros_pkg
cd uned_crazyflie_ros_pkg
git submodules init
git submodules update
cd ..
git clone https://github.com/ethz-asl/mav_comm.git
cd ../..
catkin build
echo "source devel/setup.bash" >> ~/.bashrc
```

Este paquete compila correctamente en ambas versiones de ROS y Ubuntu.

3.2. ROS2

TO-DO

3.2.1. Ubuntu 20.04 - ROS Noetic Ninjemys

TO-DO

3.2.2. Windows

TO-DO

3.2.3. Rosbridge

TO-DO

3.2.4. Conexión múltiples máquinas

TO-DO

3.2.5. Matlab

TO-DO

3.2.6. Dependencias

TO-DO

3.3. micro-ROS

TO-DO

3.3.1. FreeRTOS

TO-DO

4. Control

TO-DO: Rehacer con los nuevos parámetros y los últimos datos.

En esta sección se documentarán los controladores propuestos y operativos que se han implementado, así como sus resultados en Matlab, Gazebo y experimentos reales. A continuación se muestra el esquema de control genérico implementado.

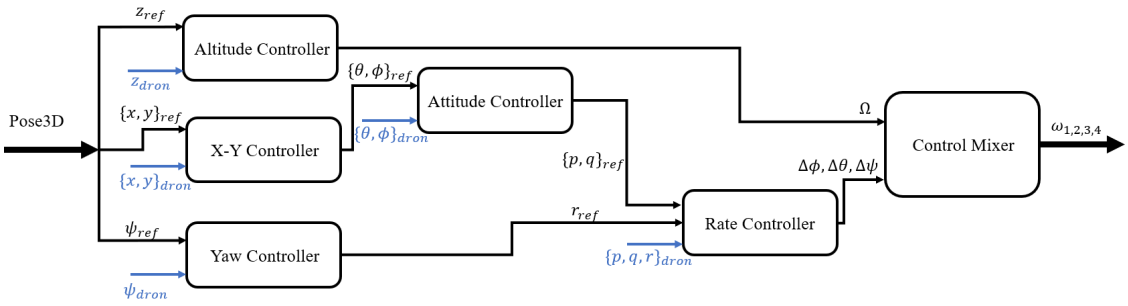

Figura 4: Esquema de control para el dron crazyflie 2.1.

4.1. PID continuo

El primer controlador probado sobre el sistema se trata de una arquitectura de control basada en controladores del tipo PID en tiempo continuo. A continuación se detallan los valores establecidos para los distintos parámetros de cada controlador. La configuración de los controladores PID se trata del modelo paralelo. ecuación 23, donde N es el coeficiente del filtro del término derivativo y toma el valor 100.

$$C(s) = \frac{U(s)}{E(s)} = K_p + K_i \cdot \frac{1}{s} + K_d \cdot \frac{N}{1 + N \cdot \frac{1}{s}} \tag{23}$$

Controlador	Objetivo	K_p	K_i	K_d
Altitude	z	15000,0	3500,0	9000,0
	w	1,0	0,0	0,0
X-Y	x	1,0	0,0	0,0
	u	30,0	2,0	0,0
	y	1,0	0,0	0,0
	v	$-30,0$	$-2,0$	0,0
Attitude	ϕ	3,5	2,0	0,0
	θ	3,5	2,0	0,0
Yaw	ψ	3,0	0,0	0,0
Rate	p	70,0	0,0	0,0
	q	70,0	0,0	0,0
	r	70,0	16,7	0,0

Tabla 2: Controladores PID. Parámetros.

El resultado global para esta configuración se muestra en la figura 5.

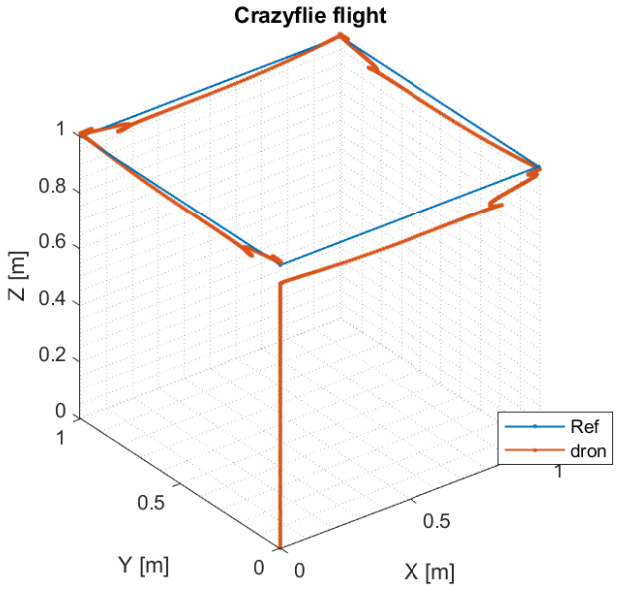

Figura 5: PID continuo. Resultados.

4.2. PID discreto

En el caso del controlador PID discreto, se toma como referencia los valores del caso continuo. El sistema se ejecuta a tres frecuencias distintas. Los sistemas más rápidos se ejecutan a $500Hz$ ($T = 0{,}002s$) y se corresponden con el "*Rate Controller*" y el "*Yaw Controller*". En el caso del "*Attitude Controller*", la frecuencia de funcionamiento se reduce a $250Hz$ ($T = 0{,}004s$). Finalmente, el resto de controladores operan a una frecuencia de funcionamiento de $100Hz$ ($T = 0{,}010s$). Para la discretización de los controladores, se ha estimado que la aproximación más adecuada es la trapezoidal (*Tustin's approximation* ó *the bilinear transformation*) [10], ecuación 25.

$$U(s) = \left(K_p + K_i \cdot \frac{1}{s} + K_d \cdot \frac{N}{1 + N \cdot \frac{1}{s}}\right) E(s) \rightarrow u(t) = K_p e(t) + K_i \int e(t)\, dt + K_d \frac{de(t)}{dt} \quad (24)$$

$$s = \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} \rightarrow z = e^{sT} \approx \frac{1 + (sT/2)}{1 - (sT/2)} \quad (25)$$

$$C(z) = \frac{U(z)}{E(z)} = \frac{q_0 + q_1 z^{-1} + q_2 z^{-2}}{1 - z^{-1}} \begin{cases} q_0 & = K_p + \frac{K_i T}{2} + \frac{K_d}{T} \\ q_1 & = -K_p + \frac{K_i T}{2} - 2\frac{K_d}{T} \\ q_2 & = \frac{K_d}{T} \end{cases} \quad (26)$$

Controlador	Objetivo	q_0	q_1	q_2
Altitude	z	915000	-1815000	900000,0
	w	1,0	$-1,0$	0,0
X-Y	x	1,0	$-1,0$	0,0
	u	30,01	$-29,99$	0,00
	y	1,0	$-1,0$	0,0
	v	$-30,01$	29,99	0,00
Attitude	ϕ	3,504	$-3,4960$	0,0
	θ	3,5040	$-3,4960$	0,0
Yaw	ψ	3,0	$-3,0$	0,0
Rate	p	70,0	$-70,0$	0,0
	q	70,0	$-70,0$	0,0
	r	70,0167	$-69,9833$	0,00

Tabla 3: Controladores PID discreto. Parámetros.

El resultado global para esta configuración se muestra en la figura 6.

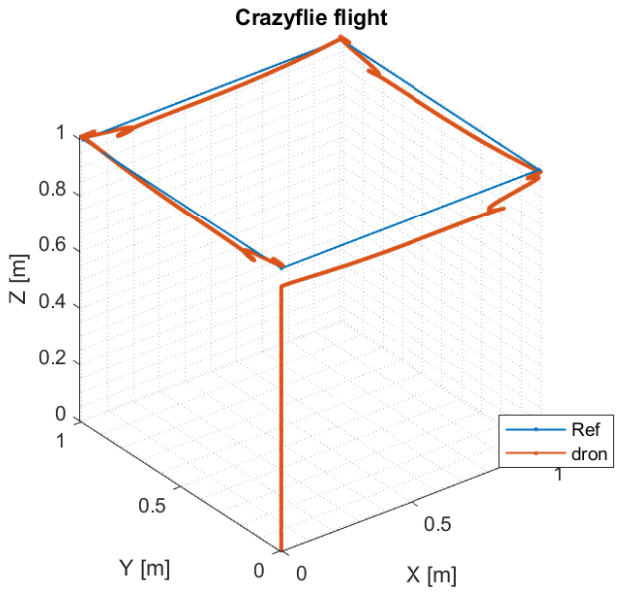

Figura 6: PID discreto. Resultados.

4.3. MPC

4.4. Control basado en eventos

A partir de los controladores diseñados en secciones anteriores, se ha procedido a montar la correspondiente estructura de control basado en eventos. En este caso, tras por la velocidad que demandan el nivel inferior, comprendido por el *Yaw Controller*, *Attitude Controller* y *Rate Controller*, $500[Hz]$, se opta por implementar el control basado en eventos para los controladores de posición exclusivamente. El generador de eventos se rige por tres leyes:

- Ciclo límite con una frecuencia menor que la empleada en el control en tiempo discreto.
- Cota de error inferior a los $10[cm]$. Para evitar un sobremuestreo en la simulación, se emplea esta cota a una frecuencia de $100[Hz]$.
- Implementación *send-on-delta* simple con una banda de $3[cm]$.

El resultado global para esta configuración se muestra en la figura 7. LOS RESULTADOS SON MALÍSIMOS. LÓGICO PORQUE NO SE HAN AJUSTADO LOS PARÁMETROS DE LOS CONTROLADORES. A LO MEJOR IMPLEMENTANDO ALGUNA REGLA COMO SIMC O EL MÉTODO AMIGO PUEDE MEJORAR.

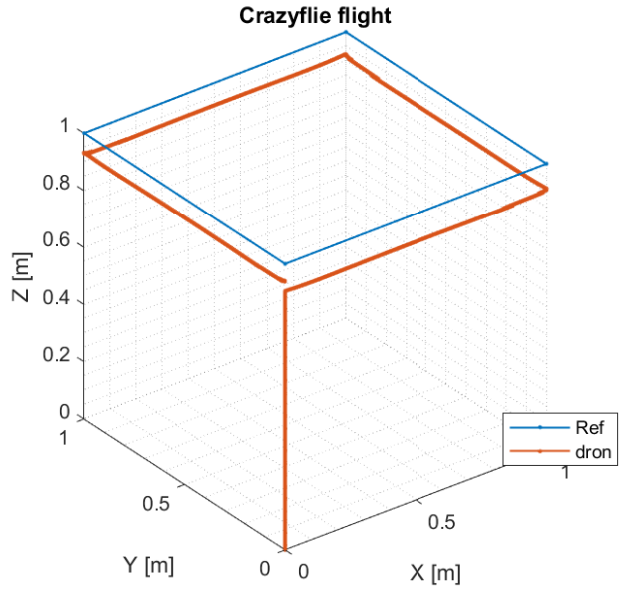

Figura 7: PID basado en eventos. Resultados.

5. Simulador

Como herramienta de simulación se trabaja con ROS/Gazebo y Matlab/Simulink. TO-DO

5.1. Gazebo

TO-DO

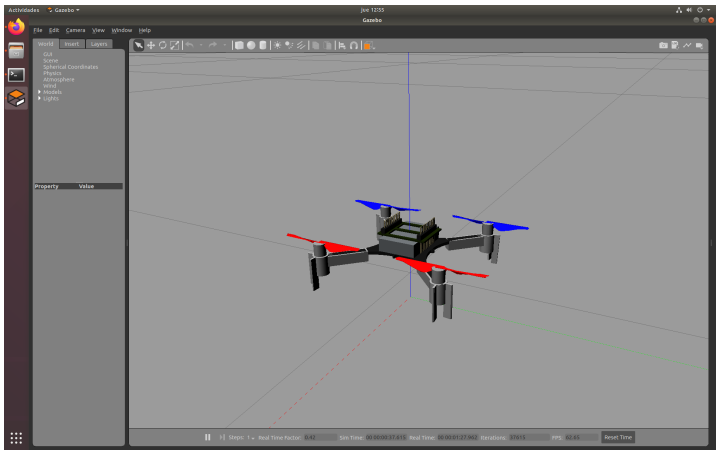

Figura 8: Crazyflie 2.1 en Gazebo

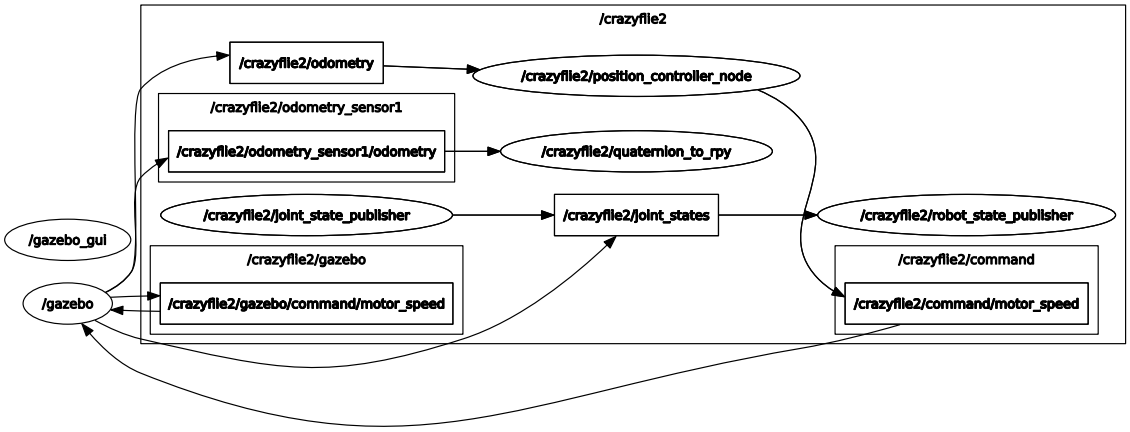

Figura 9: Captura de rqt_graph durante la ejecución de la simulación C++.

5.2. Matlab

TO-DO

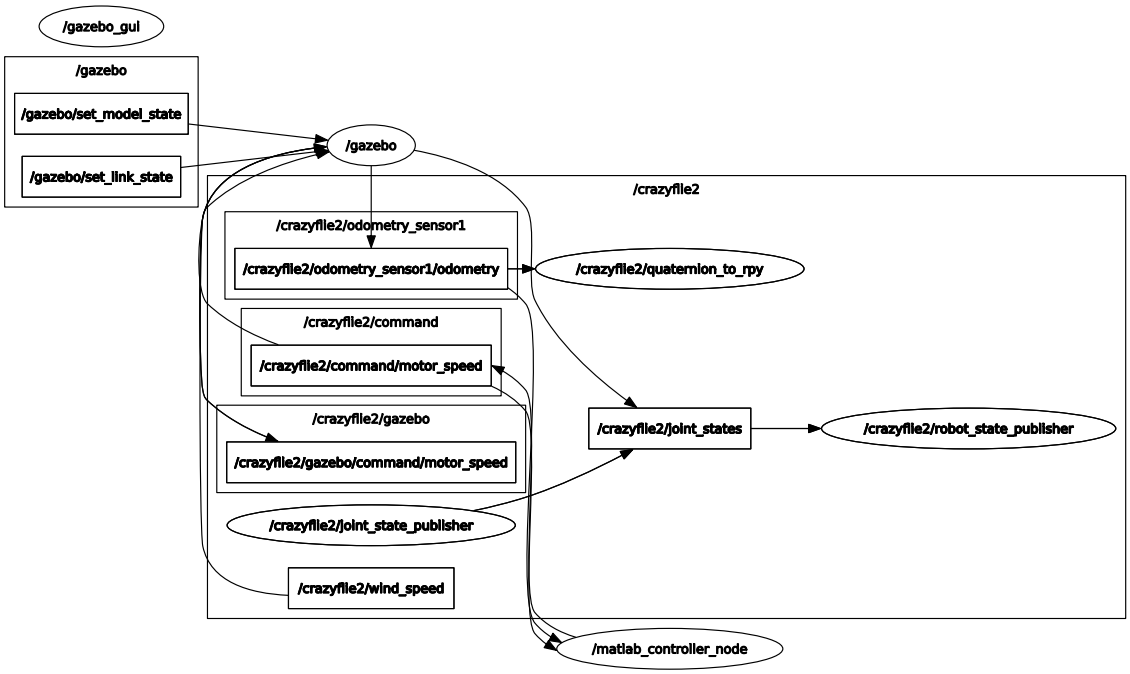

Figura 10: Captura de rqt_graph durante la ejecución de la simulación Matlab.

6. Hardware-in-the-Loop

TO-DO

6.1. ROS

TO-DO

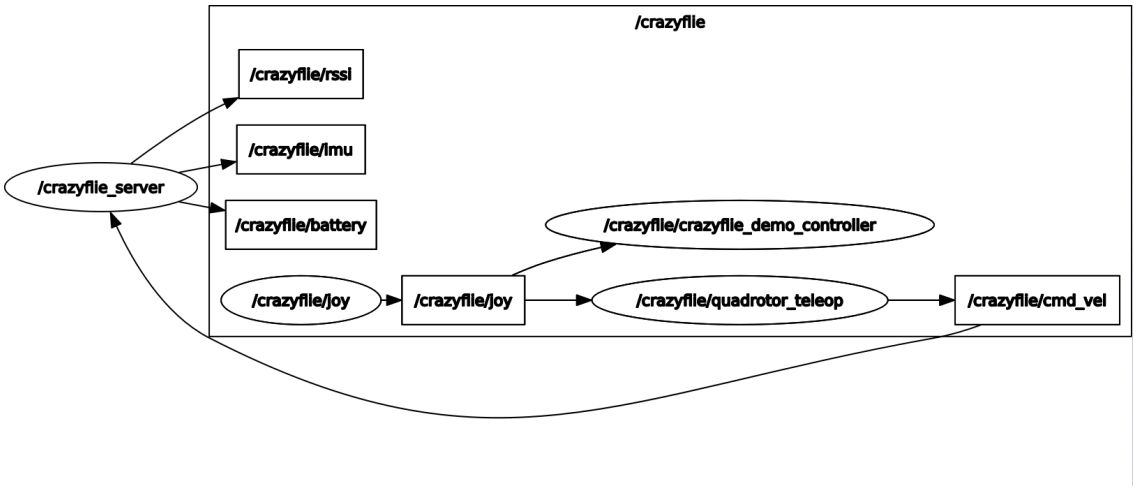

Figura 11: rosgraph.

6.2. ROS2

TO-DO

6.3. micro-ROS

TO-DO

7. Publicaciones

TO-DO

8. Líneas de trabajo

TO-DO

Referencias

[1] John H Blakelock. *Automatic control of aircraft and missiles*. John Wiley & Sons, 1991.

[2] Samir Bouabdallah, Andre Noth, and Roland Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2451–2456. IEEE, 2004.

[3] Sebastian Green and Pontus Månsson. Autonomous control of unmanned aerial multi-agent networks in confined spaces. 2019.

[4] Marcus Greiff. Modelling and control of the crazyflie quadrotor for aggressive and autonomous flight by optical flow driven state estimation. 2017.

[5] EM Greitzer, ZS Spakovszky, and IA Waitz. Thermodynamics and propulsion. *Mechanical Engineering, MIT*, 2006.

[6] Carlos Luis and Jérôme Le Ny. Design of a trajectory tracking controller for a nanoquadcopter. *arXiv preprint arXiv:1608.05786*, 2016.

[7] J Peraire and S Widnall. Lecture l28 - 3d rigid body dynamics: Equations of motion. *Dynamics*, 2009.

[8] J Peraire and S Widnall. Lecture l29-3d rigid body dynamics. *Dynamics*, 2009.

[9] Francesco Sabatino. Quadrotor control: modeling, nonlinearcontrol design, and simulation, 2015.

[10] Björn Wittenmark, Karl Johan Åström, and Karl-Erik Årzén. Computer control: An overview. *IFAC Professional Brief*, 1:2, 2002.