

Crazyflie 2.1 - ROS (uned_crazyflie_ros_pkg)

Francisco J. Mañas-Álvarez

4 de mayo de 2021

Índice

1. Introducción	2
1.1. Estructura	2
2. Crazyflie	2
2.1. Características	3
2.1.1. Especificaciones mecánicas	3
2.1.2. Radio	3
2.1.3. Microcontroladores	3
2.1.4. IMU	3
2.1.5. Especificaciones de vuelo	3
2.1.6. Conectores	3
2.2. Proyectos anteriores	4
2.3. Modelado	4
2.3.1. Ecuaciones del equilibrio de fuerzas	6
2.3.2. Equilibrio de Momentos	7
3. Control	8
3.1. PID continuo	9
3.2. PID discreto	10
3.3. MPC	10
3.4. Control basado en eventos	10
4. Instalación	10
4.1. ROS	11
4.2. rosbridge_suite	11
4.3. Matlab	11
4.4. Dependencias	12
4.5. Ubuntu 18.04 - ROS Melodic Morenia	12
4.6. Ubuntu 20.04 - ROS Noetic Ninjemys	12
5. Simulador	13
5.1. C++	13
5.2. Matlab	16
6. Hardware-in-the-Loop	18
6.1. Crazyflie_ros	18

1. Introducción

Este repositorio¹ contiene los paquetes de ROS y ficheros de configuración para la teleoperación y simulación del dron crazyflie 2.1 en ROS, 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.

1.1. Estructura

- **doc.** Contiene un fichero *.tex* que aborda más en detalle toda la información relacionada con el repositorio: esquemas de ROS, búsquedas bibliográficas, enlaces de interés, etc.
- **scripts.** Contiene aquellos ficheros auxiliares que no forman parte de ningún paquete de ROS. Por ejemplo, ficheros *.sh* para automatizar procesos repetitivos como la conversión de los ficheros *.bag* a *txt* o los scripts de Matlab para representar datasets.
- **submodules.** En este directorio están vinculados otros repositorios que se reutilizan, o se toman de base, para tareas ya abordadas por otros usuarios.
- **uned_crazyflie_config.** Paquete de ROS. 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. Comprende los nodos propios desarrollados para la integración de drones en el sistema. Incluye toda la información asociada para su correcta puesta en marcha.
- **uned_crazyflie_test.** Paquete de ROS. 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, 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 quadrotor que por su reducido tamaño entra dentro de la categoría de "nanoquadrotor". Su diseño está especialmente pensado para vuelo en interiores. A continuación se detallan sus especificaciones técnicas, disponibles también en la web del dron².



Figura 1: Crazyflie 2.1.

¹Github: https://github.com/FranciscoJManasAlvarez/uned_crazyflie_ros_pkg

²Bitcraze: <https://store.bitcraze.io/products/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. Proyectos anteriores

En este apartado se describen los trabajos ya realizados sobre crazyflie que se han consultado y reutilizado parcialmente para la herramienta desarrollada.

CrazyS ³

TO-DO

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin euismod, erat ultrices hendrerit consequat, ligula nisi semper felis, id euismod ex erat eget diam. Vestibulum ut leo condimentum, ullamcorper orci id, suscipit arcu. Suspendisse suscipit purus tincidunt ex eleifend, sed tristique eros maximus. Donec vitae nisl congue, gravida orci vitae, consectetur sapien. Interdum et malesuada fames ac ante ipsum primis in faucibus. Duis ante orci, blandit rutrum urna in, mollis commodo lorem. Donec blandit, velit sed aliquam auctor, mi nunc lobortis arcu, in imperdiet sapien est vitae libero. Donec tincidunt quam ipsum, vitae rhoncus dolor efficitur id. Proin laoreet, ipsum quis vehicula condimentum, ex ipsum tincidunt est, bibendum iaculis enim est vel risus.

Crazyflie_ros ⁴

TO-DO

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin euismod, erat ultrices hendrerit consequat, ligula nisi semper felis, id euismod ex erat eget diam. Vestibulum ut leo condimentum, ullamcorper orci id, suscipit arcu. Suspendisse suscipit purus tincidunt ex eleifend, sed tristique eros maximus. Donec vitae nisl congue, gravida orci vitae, consectetur sapien. Interdum et malesuada fames ac ante ipsum primis in faucibus. Duis ante orci, blandit rutrum urna in, mollis commodo lorem. Donec blandit, velit sed aliquam auctor, mi nunc lobortis arcu, in imperdiet sapien est vitae libero. Donec tincidunt quam ipsum, vitae rhoncus dolor efficitur id. Proin laoreet, ipsum quis vehicula condimentum, ex ipsum tincidunt est, bibendum iaculis enim est vel risus.

2.3. 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.

³Github: <https://github.com/gsilano/CrazyS>

⁴Github: https://github.com/whoenig/crazyflie_ros

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:

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

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

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

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

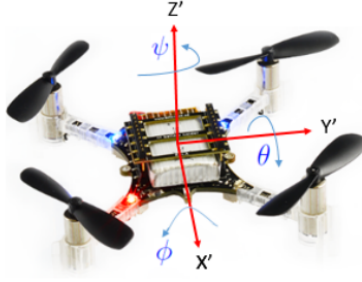


Figura 2: Ejes y ángulos.

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.

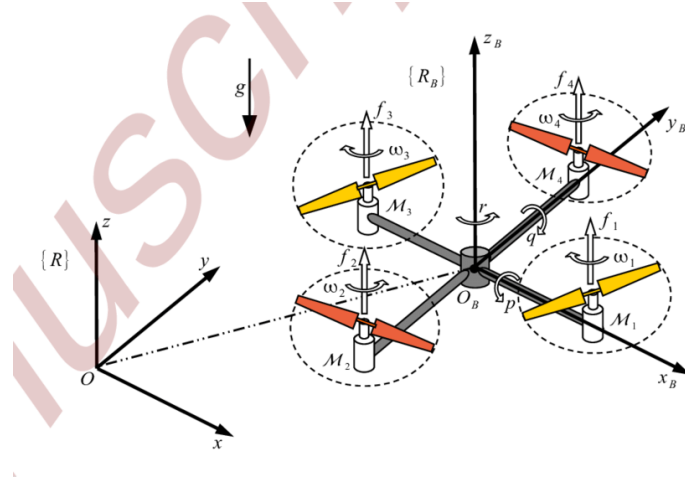


Figura 3: Esquema de un quadrotor.

2.3.1. 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 Coriolis⁵ [1]

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

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.

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

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

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.

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

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

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

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

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

⁵Explicación: <https://www.youtube.com/watch?v=-0yRCgv-hPs>

2.3.2. 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:

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

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

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

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

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

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

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

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

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

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

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

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

$$M = \Sigma P_i \times F_i + \Sigma \tau_i \quad (17)$$

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:

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

Conociendo la fuerza que genera cada motor a partir de la ecuación 2.3.1, el momento producto de esa fuerza se puede determinar según la siguiente expresión:

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

Aplicando la conservación del momento angular:

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

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

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

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

$$Q = k_D \rho n^2 D^5 \quad (21)$$

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

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].

3. Control

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.

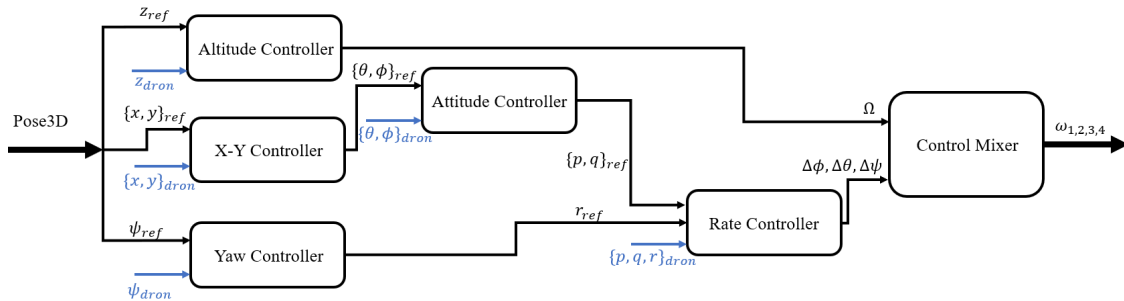


Figura 4: Esquema de control para el dron crazyflie 2.1.

3.1. 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.

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

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.

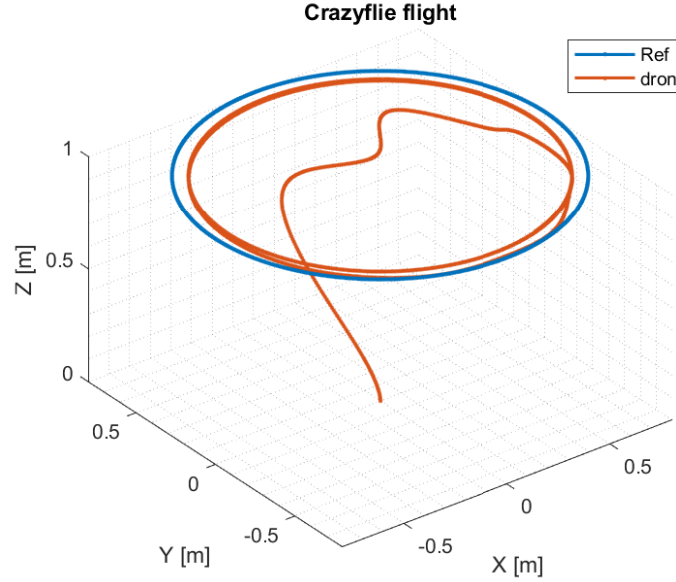


Figura 5: PID continuo. Resultados.

3.2. 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.

3.3. MPC

3.4. Control basado en eventos

4. 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).

4.1. 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.

4.2. 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
...
commentstyle export ROS_MASTER_URI = http://xxx.xxx.x.xx:11311
commentstyle export ROS_HOSTNAME = xxx.xxx.x.xx
```

Para el lanzamiento del paquete, se emplea el comando

```
roslaunch rosbridge_server rosbridge_websocket.launch
```

4.3. 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 `roshuttdown` al final del script para evitar dejar el nodo en el aire y al principio por si se nos ha olvidado cerrarlo anteriormente, que no de problemas.

4.4. Dependencias

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

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

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

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

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

```
sudo apt-get install ros-<rostdistro>-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.

4.5. 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-rtt-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).

4.6. 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_ws/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.

5. Simulador

La simulación se puede hacer tanto de forma exclusiva en c++ como empleando Matlab.

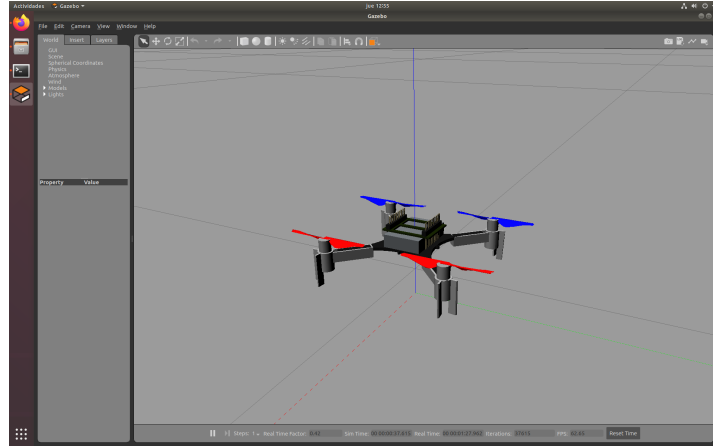


Figura 6: Crazyflie 2.1 en Gazebo

5.1. C++

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin euismod, erat ultrices hendrerit consequat, ligula nisi semper felis, id euismod ex erat eget diam. Vestibulum ut leo condimentum, ullamcorper orci id, suscipit arcu. Suspendisse suscipit purus tincidunt ex eleifend, sed tristique eros maximus. Donec vitae nisl congue, gravida orci vitae, consectetur sapien. Interdum et malesuada fames ac ante ipsum primis in faucibus. Duis ante orci, blandit rutrum urna in, mollis commodo lorem. Donec blandit, velit sed aliquam auctor, mi nunc lobortis arcu, in imperdiet sapien est vitae libero. Donec tincidunt quam ipsum, vitae rhoncus dolor efficitur id. Proin laoreet, ipsum quis vehicula condimentum, ex ipsum tincidunt est, bibendum iaculis enim est vel risus.

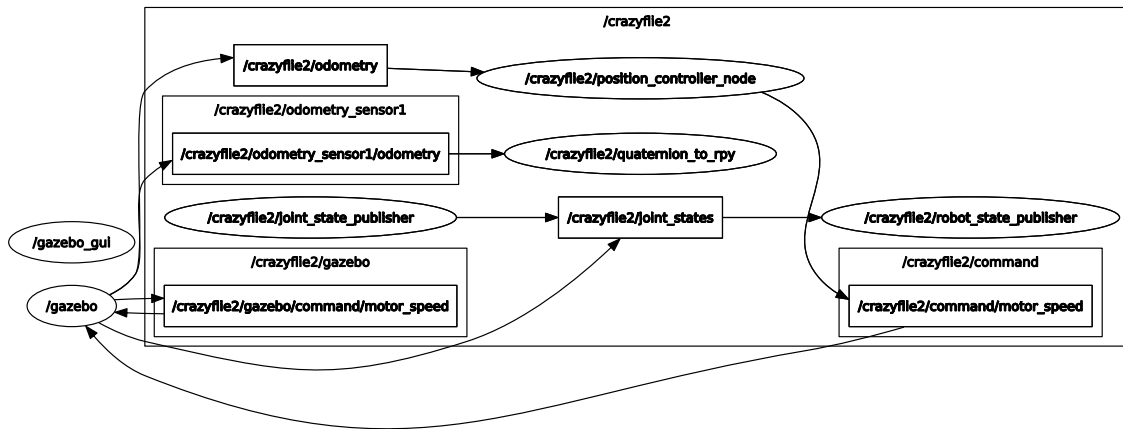


Figura 7: Captura de rqt_graph durante la ejecución de la simulación C++ (simple).

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin euismod, erat ultrices hendrerit consequat, ligula nisi semper felis, id euismod ex erat eget diam. Vestibulum ut leo condimentum, ullamcorper orci id, suscipit arcu. Suspendisse suscipit purus tincidunt ex eleifend, sed tristique eros maximus. Donec vitae nisl congue, gravida orci vitae, consectetur sapien. Interdum et malesuada fames ac ante ipsum primis in faucibus. Duis ante orci, blandit rutrum urna in, mollis commodo lorem. Donec blandit, velit sed aliquam auctor, mi nunc lobortis arcu, in imperdiet sapien est vitae libero. Donec tincidunt quam ipsum, vitae rhoncus dolor efficitur id. Proin laoreet, ipsum quis vehicula condimentum, ex ipsum tincidunt est, bibendum iaculis enim est vel risus.

In hac habitasse platea dictumst. Aenean justo tortor, congue non congue ut, egestas a sem. Mauris egestas diam id nulla dictum eleifend. Nulla aliquam vulputate dapibus. Morbi tellus dolor, ornare sed iaculis ac, sodales sed nisl. Sed elementum, ligula eget venenatis congue, nisi nulla finibus mauris, sed hendrerit dui odio et massa. Cras nec enim ut nunc rhoncus cursus. Vestibulum ex ipsum, commodo a dolor a, bibendum fermentum lacus. Praesent et ultricies sem. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. In iaculis elit eget mattis auctor. Duis vel gravida magna. Cras egestas cursus rutrum. Duis consectetur et ex et commodo. Suspendisse dignissim magna ac mi porta bibendum. Praesent sollicitudin aliquet malesuada.

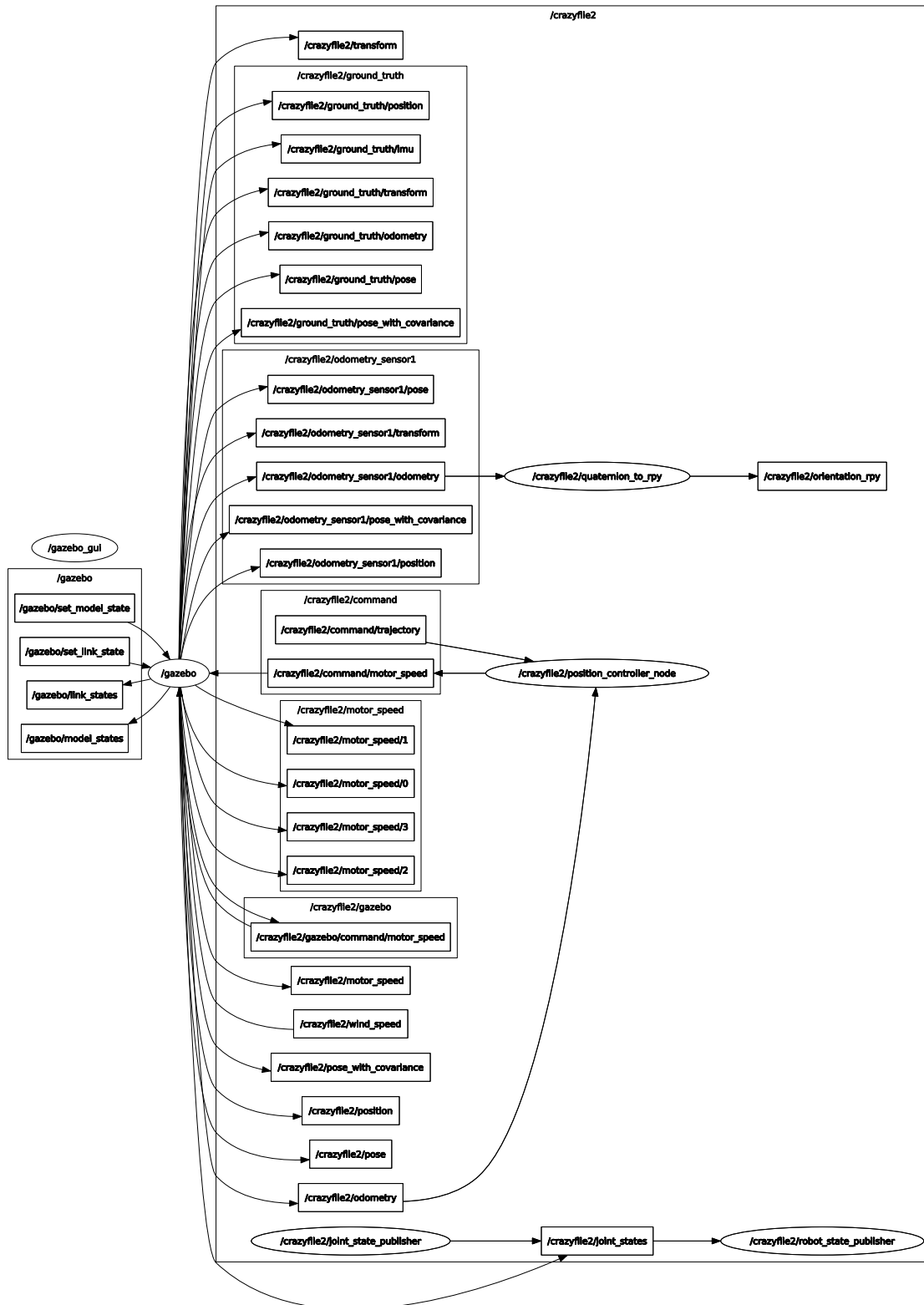


Figura 8: Captura de rqt_graph durante la ejecución de la simulación C++.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin euismod, erat ultrices hendrerit consequat, ligula nisi semper felis, id euismod ex erat eget diam. Vestibulum ut leo condimentum, ullamcorper orci id, suscipit arcu. Suspendisse suscipit purus tincidunt ex eleifend, sed tristique eros maximus. Donec vitae nisl congue, gravida orci vitae, consectetur sapien. Interdum et malesuada fames ac ante ipsum primis in faucibus. Duis ante orci, blandit rutrum urna in, mollis commodo lorem. Donec blandit, velit sed aliquam auctor, mi nunc lobortis arcu, in imperdiet sapien est vitae libero. Donec tincidunt quam ipsum, vitae rhoncus dolor efficitur id. Proin laoreet, ipsum quis vehicula condimentum, ex ipsum tincidunt est, bibendum iaculis enim est vel risus.

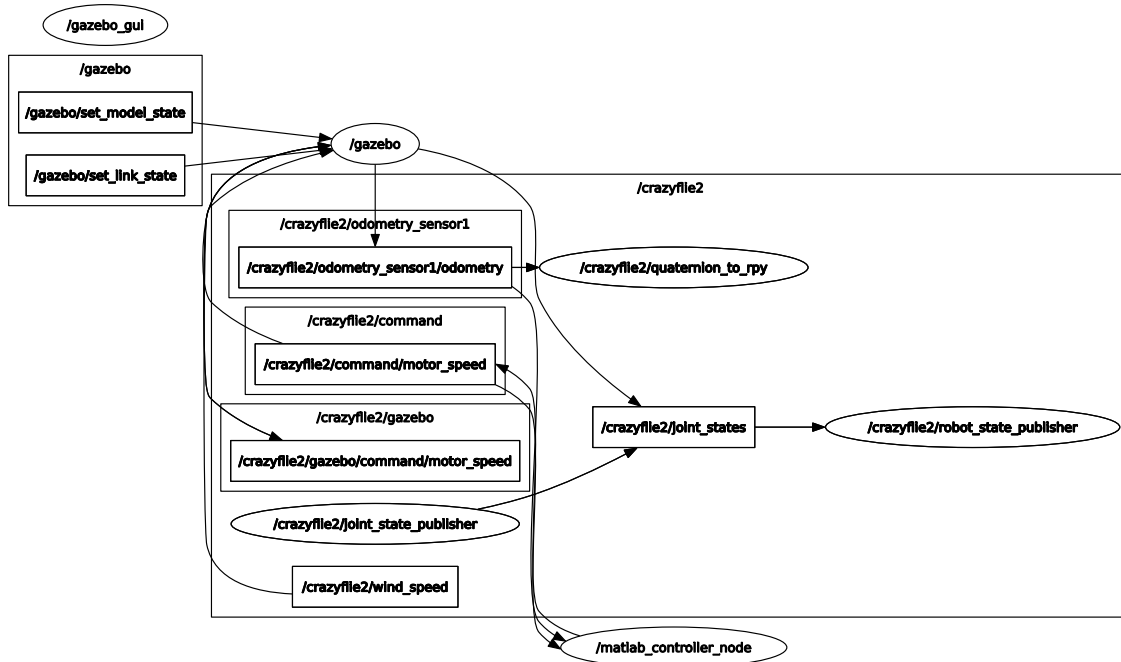


Figura 9: Captura de `rqt_graph` durante la ejecución de la simulación Matlab (simple).

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin euismod, erat ultrices hendrerit consequat, ligula nisi semper felis, id euismod ex erat eget diam. Vestibulum ut leo condimentum, ullamcorper orci id, suscipit arcu. Suspendisse suscipit purus tincidunt ex eleifend, sed tristique eros maximus. Donec vitae nisl congue, gravida orci vitae, consectetur sapien. Interdum et malesuada fames ac ante ipsum primis in faucibus. Duis ante orci, blandit rutrum urna in, mollis commodo lorem. Donec blandit, velit sed aliquam auctor, mi nunc lobortis arcu, in imperdiet sapien est vitae libero. Donec tincidunt quam ipsum, vitae rhoncus dolor efficitur id. Proin laoreet, ipsum quis vehicula condimentum, ex ipsum tincidunt est, bibendum iaculis enim est vel risus.

In hac habitasse platea dictumst. Aenean justo tortor, congrue non congrue ut, egestas a sem. Mauris egestas diam id nulla dictum eleifend. Nulla aliquam vulputate dapibus. Morbi tellus dolor, ornare sed iaculis ac, sodales sed nisl. Sed elementum, ligula eget venenatis congrue, nisi nulla finibus mauris, sed hendrerit dui odio et massa. Cras nec enim ut nunc rhoncus cursus. Vestibulum ex ipsum, commodo a dolor a, bibendum fermentum lacus. Praesent et ultricies sem. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. In iaculis elit eget mattis auctor. Duis vel gravida magna. Cras egestas cursus rutrum. Duis consectetur et ex et commodo. Suspendisse dignissim magna ac mi porta bibendum. Praesent sollicitudin aliquet malesuada.

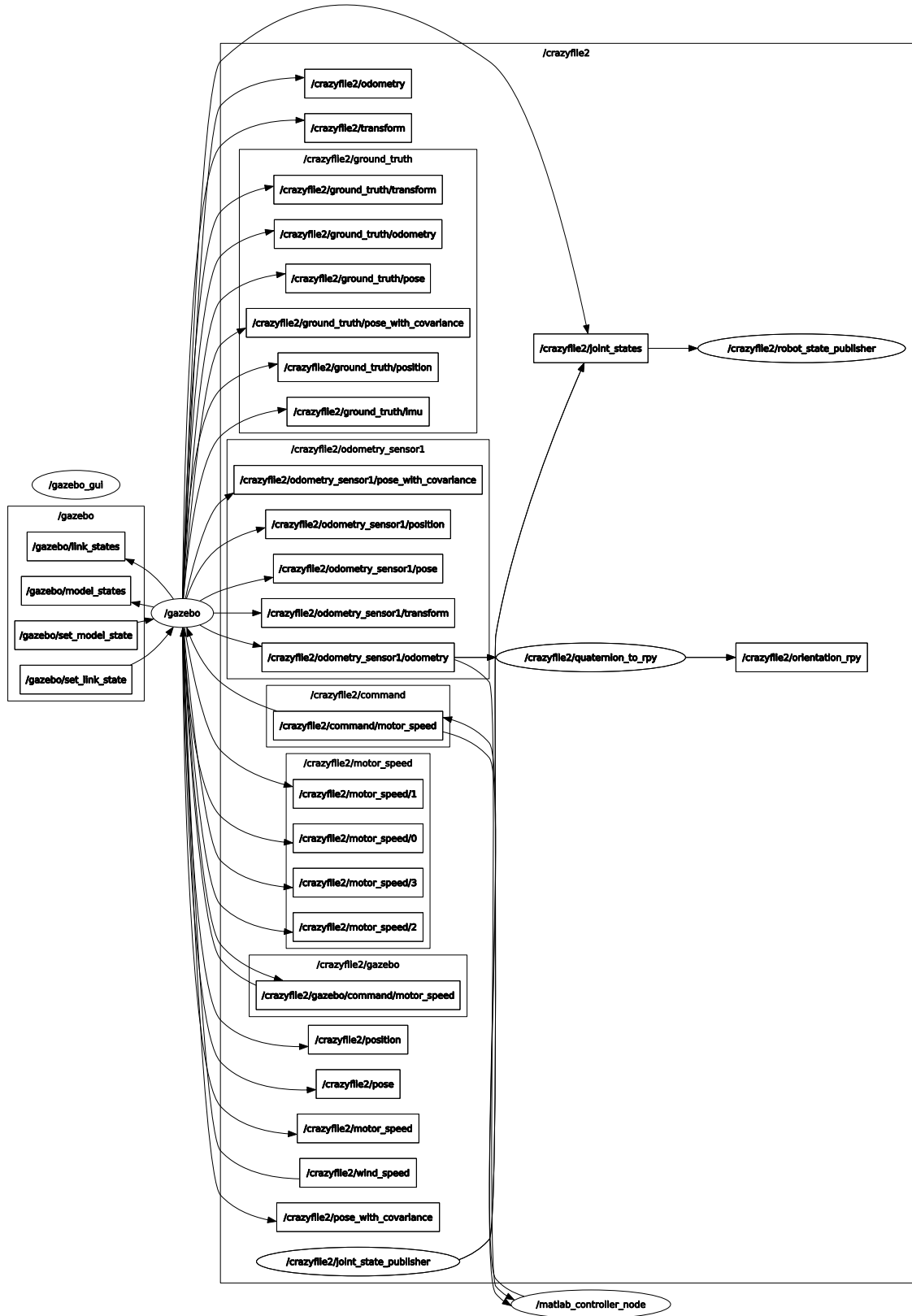


Figura 10: Captura de rqt_graph durante la ejecución de la simulación Matlab.

6. Hardware-in-the-Loop

6.1. Crazyflie_ros

TO-DO

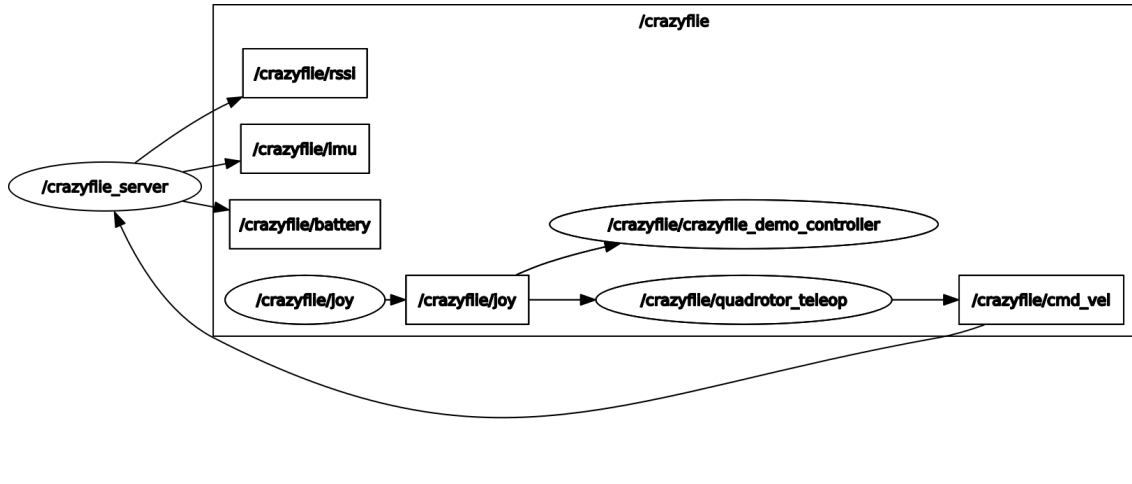


Figura 11: rosgaph.

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.