Leader-Follower Decentraliced Control of a Nanoquadrotor Swarm

CRISTINA ESCRIBANO

Automatic and Electronic Department
Date: March 4, 2019
Supervisor: Ernesto Gambao and Christos Verginis
Examiner: Tribunal
Universidad Politécnica de Madrid

Abstract

This thesis presents an adaptive control coordination of a swarm of aerial robots in order to be able to develop cooperative tasks. The aim is to test a decentralized algorithm in drones to check its behavior and illustrate theoretical findings. More detailed, the algorithm purpose is to track a trajectory while guarantee collision avoidance between agents and, also, keep certain robots inside their sense area, within the meaning of a connected sensing graph that has to remain connected. The codification of the control has been done in Python and ROS. Simulation results are provided in order to be able to make comparisons between different graph and trajectories and, even more, to make future comparisons with new algorithms.

Contents

Agradecimientos

Primero quiero darle las gracias a mi familia, en especial a mi madre y a mi padre. Ellos son los que han hecho posible no solo que haya tenido la oportunidad de hacer este TFG, sino que hoy esté acabando la carrera ya que me han apoyado siempre en mis aspiraciones y me han ayudado en todos los momentos del camino.

Diego, has sido mi apoyo diario durante estos meses y tu ayuda y motivación han sido indispensables durante todo el proyecto. Gracias a ti he aprendido mucho más de lo establecido y aunque en ocasiones haya sido duro, ha merecido la pena. Muchísimas gracias.

Gracias también a Pedro y a Christos. Gracias por haberme dado la oportunidad de trabajar con vosotros, sino hoy no estaría terminando mi TFG en KTH. Pero, sobre todo, gracias por ayudarme y responder a todas mis dudas con paciencia.

Finalmente, gracias a todos mis compañeros de laboratorio, en especial a Nicola, Elia, Umar, Aldo, Frank y Kuba que han hecho que las interminables horas en el laboratorio hayan sido muy divertidas y que los 'No funciona nada' provocasen mas risas que desesperación. Y, por supuesto, gracias a Joaquín por todos los descansos de una hora y media que han contribuido tanto a este proyecto.

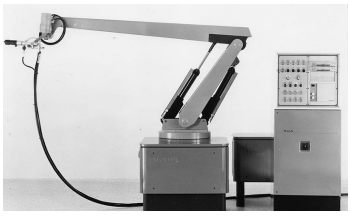
Chapter 0

Resumen

0.1 Introducción

Durante los últimos años ha habido un creciente interés por la robótica. Un gran numero de empresas han reemplazado a los hombres por robots no solo en tareas peligrosas o aburridas, sino también en tareas que no pueden ser realizadas por humanos.

Historia de la robótica



(a)

(b)

Figure 1

Los primeros robots tal cual los concebimos hoy en día fueron creados en los 50 por Goerge Devol. Fue a través de *Unimate* (Fig, 1a), un

brazo robótico reprogramable. Mas adelante, en 1978, se creo el primer robot tipo *SCARA* que conllevo una gran mejora en trabajos de 'Pick and Place' y se introdujo en las cadenas de montaje.

En junio de 1997, *The Pathfinder Mission* aterrizó en Marte y recopiló información hasta septiembre del mismo año, podemos verlo en la Figura 1b. En 2005 empezó la revolución de la robótica, se avanzó mucho en su tecnología y muchos más robots fueron comercializados. Un ejemplo de ello es el robot *Starfish*, creado en 2006.

Robots móviles

Los robots en general, y los robots móviles en particular han cambiado la historia reciente de muchas maneras. Existe actualmente un gran variedad de robots móviles especializados en distintos ambientes y con diferentes grados de complejidad. En la Figura 2 podemos ver el robot *Atlas* creado por *Boston Dynamics*, un humanoide que puede correr y saltar con gran agilidad. También podemos destacar al dron *Pluto Plus*, el cual se usa en el ámbito militar para detección y destrucción de minas bajo el agua.

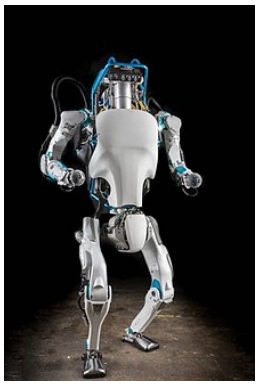


Figure 2

Durante los últimos años, una gran parte de la investigación en robótica móvil ha estado focalizada en los drones o UAV (Unmanned Aerial Vehicle). Se caracterizan por su gran versatilidad, pudiendo utilizarse en multitud de aplicaciones. Además, proporcionan una nueva y amplia área de investigación.

Estado del arte

Los drones tienen un futuro prometedor en muchos campos. Creando este tipo de robots de forma que sean intuitivos, amigables, fáciles de usar e incluso de fabricar, como por ejemplo con impresoras 3D, podemos crear cientos de nuevas aplicaciones donde pueden ayudar en nuestro día a día.

Dentro de la familia de los drones encontramos el quadrotor, también llamado cuadricóptero. Se trata de un helicóptero multirrotor que es levantado y propulsado por cuatro motores. Su popularidad se debe a su agilidad y simplicidad. Ademas, pueden ser tan grandes como para cargar cámaras ordenadores o incluso personas, o extremadamente pequeños y básicos. Por esta razón, se ha convertido en el vehículo favorito en investigación.

En la Figura 3a vemos el dron acuático *HexH2O* lanzado en 2017. Su atractivo principal es su alta resistencia al agua que hace posible la toma de imágenes tanto dentro como fuera del agua. En la Figura 3b aparece el dron creado por *AirBobotics*. Esta empresa ha propuesto una ingeniosa solución ante el problema de la poca duración de las baterías que existen actualmente. Se trata de una sistema completo que consta de una caja de aterrizaje donde se cambia de forma automáticamente la batería del dron en el momento que se requiere. Ademas, el dron actúa de forma autónoma eliminando la necesidad de un operador.

Como ejemplo innovador podemos exponer el prototipo lanzado en 2017 por *Italdesign* y *AirBus* llamado *PopUp*. Se trata de un módulo de trasporte de dos personas que se puede unir tanto a un chasis para ir por tierra o a un módulo de hélices para convertirse en un quadrotor. Ademas, es totalmente autónomo y eléctrico.

(a)

(b)

Figure 3

Robótica de enjambres

Hay situaciones en las que un único dron no es suficiente para la tarea requerida. Estas limitaciones se superan utilizando un conjunto de robots, o lo que también se llama robótica de enjambres. En la naturaleza los comportamientos colectivos permiten a los animales alcanzar logros que serían imposibles de lograr mediante un único individuo. Colonias de bacterias, bancos de peces, colmenas de abejas o hormigueros han inspirado a ingenieros para desarrollar esta misma idea en robots.

La inteligencia de enjambre consiste en un grupo de individuos simples y autónomos controlados por determinadas reglas cuya interacción solo se da a nivel local. Estos individuos no son necesariamente no-inteligentes, pero su rol es relativamente simple en comparación con la meta global. Ademas, al actuar cada uno de forma autonoma el control no esta centralizado y no se necesita un modelado del enjambre, esto proporciona una solución muy sencilla y sofisticada a muchos problemas de larga escala.

Por otro lado, ningún individuo tiene acceso al toda la información del enjambre. Típicamente, solo tienen acceso a determinada información de agentes cercanos. La inteligencia de conjunto nace de las pequeñas decisiones de cada individuo y de la comunicación entre ellos de forma local y no globalizada.

La coordinación de enjambres ha recibido una amplia atención recientemente. Este interés dentro de la investigación ha sido, en parte, provocado por el rápido desarrollo del control de sistemas multi-agentes como muestra [1]. Ademas, esta coordinación tiene numerosas aplicaciones, un ejemplo típico es el relativo al problema de actitud de interferometría tratado en [2] [3].

En [4] podemos ver una sincronización cooperativa tipo Leader-Follower donde hay una variación del líder con respecto del tiempo. Retardos en la comunicación y distintas tipologías en dinámicas se muestran en [5] donde se resalta el trabajo realizado en la prevención de colisión.

Dispersión y cohesión son comportamientos normalmente muy im-

portantes cuando los referimos a sistemas multi-agentes. Una estructura variable se estudia en [6] para garantizar un cooperación en enjambre para seguir una trayectoria con o sin líder. Diferentes formas de conservación de la conectividad entre agentes se plantean en [7] y [8].

Motivación

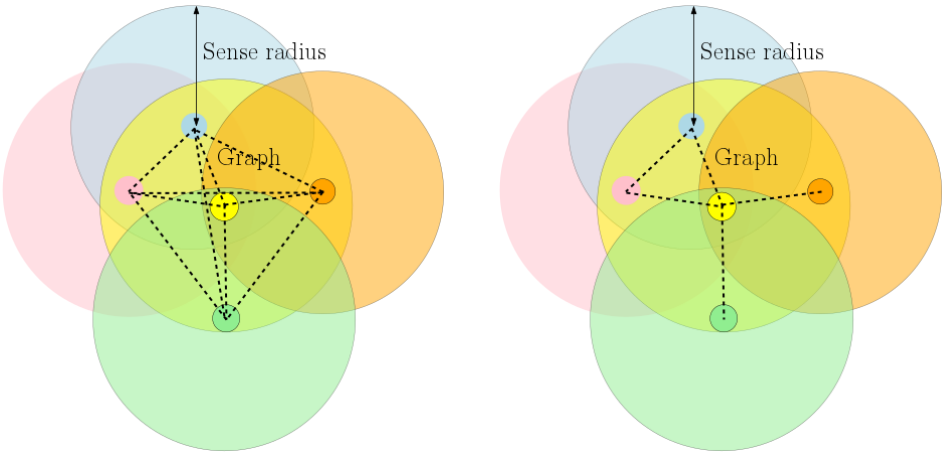
La principal motivación que me lleva a elaborar este trabajo de fin de grado es profundizar en la teoría de control y tecnología de los robots aéreos. Los drones son, en este momento, una tecnología innovadora y quiero ampliar mi conocimiento en el área.

Las ciencias me han fascinado desde muy pequeña. El interés en matemáticas y física me lleva a apreciar la tecnología, y en especial, la implementación de estas ciencias en los robots. Investigar como se puede replicar un comportamiento de la naturaleza en un conjunto de robots para construir sistemas mas optimizados me ha fascinado desde el primer momento.

0.2 Objetivo

El objetivo de este trabajo de fin de grado es implementar un control adaptativo en un enjambre de quadrotors para crear un comportamiento Leader-Follower, en el que el agente líder es el que determina que es lo que debe hacerse y el resto deben seguirle. Cada dron tiene un radio de sensibilidad dentro del cual puede detectar a drones vecinos. Consideramos que el líder es el único robot que conoce el punto que se ha de alcanzar y el único que debe converger a el, creando así toda la trayectoria.

Dentro del seguimiento, se deben cumplir varias condiciones. Primeramente, evitar la colisión entre robots, lo que se traduce en no exceder la distancia mínima de separación entre ellos. Esta distancia varia con el tamaño y la inercia del dron elegido. Para garantizarla diseñamos la grafica $G(x)$ (fig. 4a) donde están todo los agentes conectados en-



(a) (b)

Figure 4

tre ellos y en el momento que entran dentro de zona de peligro por colisiones intentan alejarse.

Por otro lado, deben mantenerse determinadas conexiones entre ellos, es decir, deben mantenerse en el radio de sensibilidad de determinados drones durante la ejecución de la trayectoria para lograr que todos ellos vayan moviéndose con la nube. Para ello, diseñamos una gráfica $G_o(x)$ (fig. 4b) donde definimos cómo queremos que sean estas conexiones, siendo todos los agentes parte de al menos una linea de la gráfica. Esto implica que, hay casos en los que un seguidor sigue a otro seguidor y no directamente al líder, hecho que dificulta notablemente la ejecución de trayectoria.

Formalmente, el problema tratado es el siguiente:

Consideramos $N > 1$ robots autónomos de forma esférica, con $N = \{1, \ldots, N\}$, operando en R y descritos por las esferas $A_i(x_i) = \{y \in R : \|x_i - y\| < r_i\}$, con $x_i \in R$, siendo i su centro de masas y $r_i \in R_{>0}$ su radio de sensibilidad.

Sin perdida de generalidad, asumimos que el robot $i = 1$ corresponde al líder, siendo $i > 1$ los seguidores, que pertenecen al grupo $F = \{2, \ldots, N\}$. La labor del líder es navegar hasta el punto deseado, denotado como $x_d \in R_n$, y el resto del equipo es responsable de mantener las conexiones y evitar las colisiones.

El radio límite de sensibilidad se expresa por medio de $d_{conn,i} \in R_{>0}$, con $d_{conn,i} > max_{j \in N}\{r_i + r_j\}$. Basándonos en esto, modelamos la topografía de la red de robots a través de $G(x) = (N, \mathscr{E}(x))$, con $\mathscr{E}(x) = \{(i,j) \in N^2 : \|x_i - x_j\| \leq min\{d_{conn,i}, d_{conn,j}\}\}$. Además, definimos $M(x) = |\mathscr{E}(x)|$. Dado el m-th borde dentro del conjunto de bordes de $\mathscr{E}(x)$, usamos la notación $(m_1, m_2) \in N^2$, donde m_1 es la cola y m_2 es la cabeza del borde $m \in \mathscr{M}(x)$, donde $\mathscr{M}(x) = \{1, \ldots, M(x)\}$ es una numeración arbitraria de los bordes $\mathscr{E}(x)$.

También debemos garantizar que determinados bordes, denominados $\mathscr{E}_o \subset \mathscr{E}(x)$, son preservados para mantener la conexión. Denotando $M_o(x) = |\mathscr{E}_o(x)|$ y siendo $\mathscr{M}_o(x) = \{1, \ldots, M_o(x)\}$ una numeración arbitraria de los bordes $\mathscr{E}_o(x)$. Llamamos l al borde del conjunto de bordes $\mathscr{E}_o$, siendo $(l_1, l_2) \in N^2$, donde l_1 es la cola y l_2 es la cabeza del borde $l \in \mathscr{M}_o(x)$.

Teniendo en cuenta estas definiciones podemos resumir el problema propuesto:

Considerando N robots autónomos y sus respectivas dinámicas el objetivo es realizar una estrategia de control descentralizado que consiga 1) convergencia del líder al punto deseado, 2) prevención de colisión entre robots, y 3) mantenimiento de la conexión en el subconjunto de robots inicialmente conectados:

1. $$\lim_{t \to \infty} x_1(t) - x_d = 0$$

2. $$A_i(x_i(t)) \cap A_j(x_j(t)) = \oslash, \forall t \in \mathrm{R}_{\geq 0}, \mathrm{i}, \mathrm{j} \in \mathrm{N}, \mathrm{i} \neq \mathrm{j}$$

3. $$\|x_{l_1}(t) - x_{l_2}(t)\| \leq min\{d_{conn,l_1}, d_{conn,l_2}\}, \forall t \in \mathrm{R}_{\geq 0}, \mathrm{l} \in \mathscr{M}_0$$

0.3 Crazyflie 2.0. Nano Quadcopter

Hardware

El problema propuesto se va a realizar en una nube de Crazyflies 2.0. Nanoquadrotors. Hemos elegido este dron debido a que es uno de los mas pequeños del mercado, y es un perfecto candidato para aplicaciones que necesitan ser probada y usadas cerca de personas y testeadas en áreas pequeñas. Además, el precio es relativamente bajo, lo cual es conveniente debido a que se necesitaran muchos de ellos. Mide 92 milímetros, pesa 34 gramos y tiene un tiempo de vuelo de aproximadamente 5 minutos.

El Crazyflie está equipado con dos microcontroladores, una IMU con giróscospo y acelerómetro en los tres ejes, además contiene un magnetógramo y un barómetro.

Software

Una de las mayores ventajas del Crazyflie 2.0. es que el software producido por la compañía que lo comercializa, Bitcraze, es totalmente libre. Esto nos proporciona un total control sobre el firmware, código que permite el control a bajo nivel del hardware dentro del quadrotor.

El firmware de nuestro quadrotor esta codificado en C y esta diseñado para optimizar el vuelo del dron, de forma que maneja la coordinación de los procesos y lleva a cabo el control en el que interviene el Kalman Filter, variables internas de lectura y escritura en tiempo real, PIDs, estimaciones de posicion, etc. Además todos los cálculos necesarios para la estabilización del Crazyflie se encuentran en él.

Esta estabilización del vuelo del dron consta de tres partes. En un primer lugar, la localización, que proporciona la posición absoluta a través un Extended Kalman Filter y de otro filtro complementario. Después se realiza la orden, que manda la posición a la que se quiere ir al regulador. Esta última parte es la encargada de mover al dron a la posición deseada, se realiza mediante una serie de PIDs en

cascada que tienen como salida los PWM aplicados a cada rotor del dron (Fig. 5).

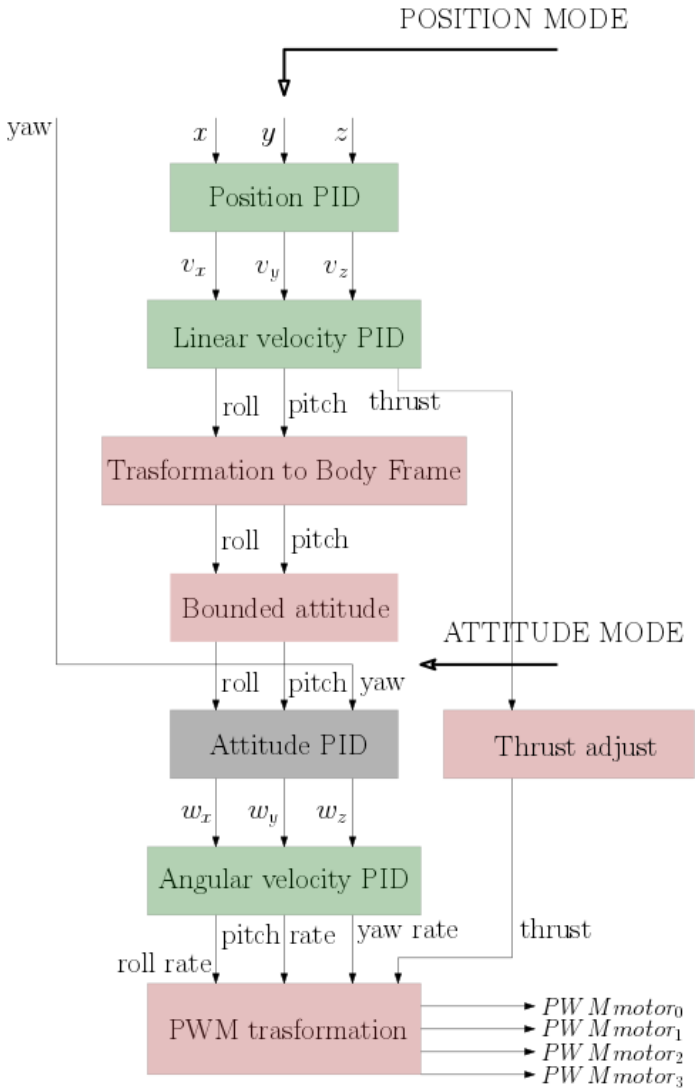


Figure 5

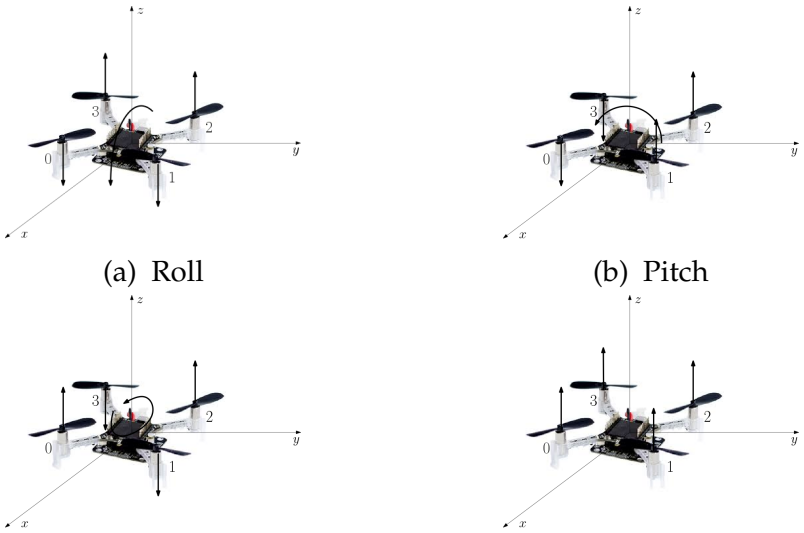
Modelo dinámico

El modelo dinámico es la descripción matemática de los movimientos del dron, es una parte clave dentro del estudio de robots, debido a que nos permite saber el comportamiento que tendrá en cada momento ante nuestras entradas.

Nociones preliminares

Antes de nada, es preciso describir los sistemas de coordenadas a los que referenciaremos nuestras ecuaciones. Basándonos en la documentación proporcionada por Bitcraze, utilizaremos un sistema de coordenadas del mundo, que es inercial y común para todos los Crazyflies, y un sistema de coordenadas no inercial en el cuerpo de cada dron.

Para describir el movimiento de una nave se necesitan seis grados de libertad, tres rotaciones y tres translaciones. La posición y velocidad del quadrotor cambia según se apliquen distintas velocidades de rotación a los motores. Con distintas combinaciones de estas rotaciones se puede conseguir cabeceo o roll, alabeo o pitch, guiñada o yaw y empuje o thrust según vemos en la Figura 6.



(a) Roll (b) Pitch

(c) Yaw (d) Thrust

Figure 6: Movimientos del dron

Ecuaciones dinámicas

En esta sección proponemos las ecuaciones dinámicas suponiendo determinadas hipótesis, basadas en propiedad físicas de nuestro dron, que son buenas aproximaciones para simplifican el estudio y la comprehensión de la respuesta de este tipo de vehículos. Las hipótesis son

las siguientes:

1. El quadrotor es un cuerpo rígido que no puede ser deformado, por lo tanto podemos conocer con total certeza sus ecuaciones dinámicas.
2. El quadrotor tiene una geometría, masa y propulsión simétrica.
3. La masa del quadrotor es constante, su derivada es 0.

Matriz de rotación

Es necesario definir una matriz transformación rígida para poder expresar las ecuaciones en el sistema de referencia óptimo. Para ello creamos la matriz R_o^b, que cambia del sistema fijo al no inercial y corresponde a una rotación en cada eje como vemos en la ecuación 1.

$$R_o^b = R_z R_y R_x \tag{1}$$

Una vez se calculan las tres matrices se obtiene

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

Y sus propiedades son las siguientes.

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

Ecuaciones de fuerza

Acorde con la Segunda Ley de Newton

$$\sum F^o = m\dot{V}_{CG}^o \tag{4}$$

Siendo la derivada de la velocidad, usando Coriolis, la siguiente expresión.

$$\dot{V}_{CG}^o = \dot{V}_{CG}^b + \omega \times V_{CG} \tag{5}$$

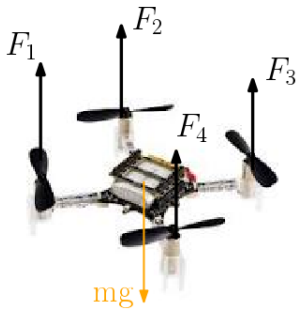


Figure 7

La fuerza es definida entonces como

$$\sum F = m(\dot{V}_{CG}^{b} + \omega \times V_{CG}) \tag{6}$$

Cada hélice genera un fuerza aerodinámica que empuja en dirección z del eje de coordenadas del cuerpo (Fig. 7).

$$\begin{bmatrix} 0 \\ 0 \\ F_z \end{bmatrix} - R_o^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = m \left(\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} R \\ P \\ Y \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \right) \tag{7}$$

Despejando $\dot{V}_{CG}$ obtenemos la ecuación de la velocidad linear.

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ F_z/m \end{bmatrix} - R_o^b \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} - \begin{bmatrix} R \\ P \\ Y \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{8}$$

Para la posición del quadrotor aplicamos la expresión 9.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = R_b^o \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{9}$$

Para calcular la expresión 8 y 9 se necesita especificar como es la fuerza generada por cada hélice.

$$F_i^b = \begin{bmatrix} 0 \\ 0 \\ T_i \end{bmatrix} \tag{10}$$

Donde el empuje generado es función de la velocidad angular

$$T_i = C_T \omega^2 \tag{11}$$

Como tenemos cuatro hélices, obtendremos la siguiente expresión.

$$\sum F_i^b = \begin{bmatrix} 0 \\ 0 \\ C_T(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \end{bmatrix} \tag{12}$$

Ecuación de momentos

El teorema de momento angular dice que

$$\sum M^o = \dot{h}^o \tag{13}$$

siendo

$$\dot{h}^o = \dot{h}^b + \omega \times hh = J\omega \tag{14}$$

Al sustituir obtenemos la siguiente expresión.

$$\sum M^b = J^b \dot{\omega} + \omega \times J\omega \tag{15}$$

Donde J es la matriz de inercia del quadrotor.

$$J = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{16}$$

Despejando $\dot{\omega}$ obtenemos

$$\begin{bmatrix} \dot{R} \\ \dot{P} \\ \dot{Y} \end{bmatrix} = J^{-1} \left(\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} - \begin{bmatrix} R \\ P \\ Y \end{bmatrix} - \begin{bmatrix} R \\ P \\ Y \end{bmatrix} \times J \begin{bmatrix} R \\ P \\ Y \end{bmatrix} \right) \tag{17}$$

Teniendo en cuenta la relación entre ω y $\dot{\Phi}$ obtenemos la ecuación de la actitud.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi \tan\theta & \cos\phi \tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} R \\ P \\ Y \end{bmatrix} \quad for\ \theta \neq \frac{\pi}{2} \tag{18}$$

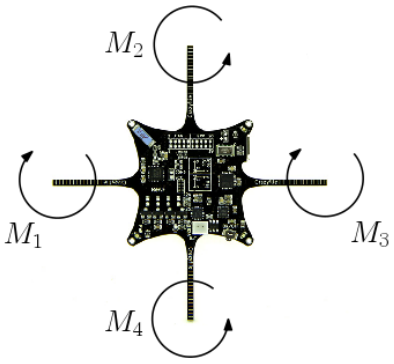


Figure 8

El momento creado por cada hélice es

$$M^b = \sum P_i^b \times F_i^b + \sum \tau_i \tag{19}$$

Si d expresa la distancia entre el centro de masas y el centro de cada rotor, la posición de cada motor es

$$P_1 = \begin{bmatrix} d/\sqrt{2} \\ -d/\sqrt{2} \\ 0 \end{bmatrix}, \quad P_2 = \begin{bmatrix} -d/\sqrt{2} \\ -d/\sqrt{2} \\ 0 \end{bmatrix}, \quad P_3 = \begin{bmatrix} -d/\sqrt{2} \\ d/\sqrt{2} \\ 0 \end{bmatrix}, \quad P_4 = \begin{bmatrix} d/\sqrt{2} \\ d/\sqrt{2} \\ 0 \end{bmatrix}, \tag{20}$$

Calculando los momentos inducidos, que solo actúan en el eje z como aparece en la Figura 8, obtenemos la expresión

$$\sum \tau = \begin{bmatrix} 0 \\ 0 \\ C_D(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2) \end{bmatrix} \tag{21}$$

Finalmente, el momento es

$$\sum M_i^b = \begin{bmatrix} dC_D/\sqrt{2}(-\omega_1^2 - \omega_2^2 + \omega_3^2 + \omega_4^2) \\ dC_D/\sqrt{2}(-\omega_1^2 + \omega_2^2 + \omega_3^2 - \omega_4^2) \\ C_D/(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2) \end{bmatrix} \tag{22}$$

0.4 Algoritmo de control

En este capítulo expondremos el algoritmo llevado acabo para conseguir el comportamiento de Líder-Follower deseado.

El control debe ser descentralizado, por lo tanto el proceso de toma de decisiones tiene que estar distribuido entre todos los agentes y cada uno de ellos tiene el control de si mismo de forma independiente. El líder es únicamente el agente que conoce el punto que hay que alcanzar, pero no manda ni organiza al resto de la nube.

Por todo esto, la idea es desarrollar un algoritmo que corra dentro de cada uno de ellos, siendo únicamente compartida la información de posición en los momentos en lo que están suficientemente cerca.

El código esta escrito en Python y la comunicación se llevara a cabo a través de ROS.

Para garantizar la conexión entre agentes definimos la función $\beta_{conn}(\eta)$: $R_{\geq 0} \rightarrow \left[0, \overline{\beta}_{conn}\right]$

$$\beta_{conn}(\eta) = \begin{cases} 0 & \eta < 0 \\ \vartheta_{conn}(\eta) & 0 \leq \eta < d_{conn}^2 \\ \overline{\beta}_{conn} & d_{conn}^2 < \eta \end{cases} \tag{23}$$

donde $\overline{\beta}_{conn}$ es una constante positiva y $\vartheta_{conn}(\eta)$ es un polinomio.

Esta función depende de la variable η, la cual es definida tal que

$$\eta = d_{conn}^2 - \|p_1 - p_2\|^2 \tag{24}$$

donde p_i es la posición de agente i.

El polinomio $\vartheta_{conn}(\eta)$ garantiza que $\beta_{conn}(\eta)$ sea dos veces diferenciable para todo m en el grupo $\mathscr{M}_o$ y debe cumplir que $\vartheta_{conn}(0) = 0$ y $\vartheta_{conn}(d_{conn}^2) = \overline{\beta}_{conn}$.

Como podemos deducir de la función $\beta_{conn}(\eta)$, cuando la distancia entre drones es mayor que el radio de sensibilidad, los robots que forman parte del borde están desconectados, η es negativa y $\beta_{conn}(\eta)$ vale

0, entonces $\beta_{conn}(\eta)$ no tiene efecto en el control. En el caso de que la distancia entre agentes sea mayor que 0, $\beta_{conn}(\eta)$ tiene el valor que le da $\vartheta_{conn}(\eta)$. Por otro lado, si la distancia es menor que cero, $\beta_{conn}(\eta)$ tiene un valor constante ya que $\beta_{conn}(\eta)$ no tiene que lidiar con las colisiones.

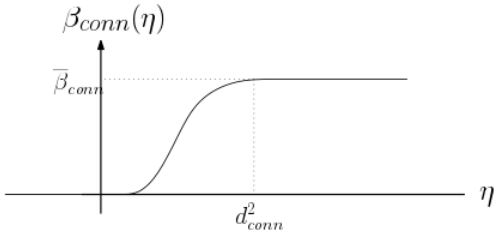


Figure 9

Cuanto mas lejos estén los agentes, mayor sera la fuerza que queremos que se apliquen entre ellos para mantenerse conectados dentro del radio de sensibilidad. Además, queremos que la traslación de los robots sea suave y progresiva, evitando movimientos rápidos que pueden desestabilizar el sistema. Por eso, necesitamos una función que vaya de 0 a $\overline{\beta}_{conn}$ de forma suave, esto se consigue a través de un polinomio de grado alto.

$$\vartheta_{conn}(\eta) = a\eta^5 + b\eta^4 + c\eta^3 \qquad a, b, c = constante \tag{25}$$

Podemos ver la forma final de la función $\beta_{conn}(\eta)$ en la Figura 9.

Por otro lado, para implementar la no colisión entre agentes, aplicamos un algoritmo similar.

Primeramente, definimos la función $\beta_{coll}(\iota) : R_{\geq 0} \rightarrow \left[0, \overline{\beta}_{coll}\right]$

$$\beta_{coll}(\iota) = \begin{cases} 0 & \iota < 0 \\ \vartheta_{coll}(\iota) & 0 \leq \iota < d_{coll} \\ \overline{\beta}_{coll} & d_{coll} < \iota \end{cases} \tag{26}$$

donde $\overline{\beta}_{coll}$ es una constante positiva y $\vartheta_{coll}(\iota)$ es un polinomio. Siendo d_{coll} la siguiente expresión

$$d_{coll} = d^2_{conn} - (r_{m_1} + r_{m_2})^2 \tag{27}$$

Esta función depende de ι, la cual se define como

$$\iota = \|p_1 - p_2\|^2 - (r_{m_1} + r_{m_2})^2 \tag{28}$$

La función $\vartheta_{coll}(\iota)$ garantiza que $\beta_{coll}(\iota)$ sea dos veces diferenciable para topo m en el conjunto $\mathscr{M}$. $\vartheta_{coll}(\iota)$, además, debe cumplir que $\vartheta_{coll}(0) = 0$ y $\vartheta_{coll}(d_{coll}) = \overline{\beta}_{coll}$.

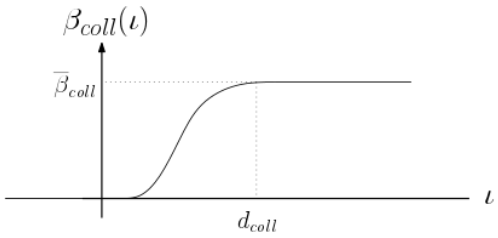


Figure 10

Cuanto mas cerca están los drones, mayor es la fuerza con la que queremos que se repelan para evitar la colisión. Siguiendo la misma idea de antes definimos $\overline{\beta}_{coll}$ como

$$\vartheta_{coll}(\iota) = a'\iota^5 + b'\iota^4 + c'\iota^3 \qquad a', b', c' = constant \tag{29}$$

Podemos observar la forma final de la función $\beta_{coll}(\iota)$ en la Figura 10.

Finalmente, definimos

$$\beta'_{conn} = \frac{\partial}{\partial \eta}\left(\frac{1}{\beta_{conn}(\eta)}\right) \ \forall l \in \mathscr{M}_o \tag{30}$$

$$\beta'_{coll} = \frac{\partial}{\partial \iota}\left(\frac{1}{\beta_{coll}(\iota)}\right) \ \forall m \in \mathscr{M} \tag{31}$$

Las cuales divergen a infinito en una pérdida de conexión o en una colisión entre los agentes l_1, l_2 o m_1, m_2, respectivamente.

Después de todo esto, el control tiene la siguiente forma

$$u = \sum_{l \in \mathscr{M}_o} \alpha_{i,l}\ \beta'_{conn} \frac{\partial \eta}{\partial x_{l_1}} + \sum_{m \in \mathscr{M}} \alpha_{i,m}\ \beta'_{coll} \frac{\partial \iota}{\partial x_{m_1}} \tag{32}$$

donde $\alpha_{i,l}$ y $\alpha_{i,m}$ toman los valores -1, 1 o 0 en los casos siguientes

$$\alpha_{i,l} = \begin{cases} -1 & i = l_1 \\ 1 & i = l_2 \\ 0 & i \neq l_1, l_2 \end{cases} \tag{33}$$

$$\alpha_{i,m} = \begin{cases} -1 & i = m_1 \\ 1 & i = m_2 \\ 0 & i \neq m_1, m_2 \end{cases} \tag{34}$$

Comprobamos que los términos de no colisión y conexión implementados actualmente en la señal de control no son suficientes ya que obtenemos comportamientos oscilatorios, por lo que añadimos un término disipativo, ξ. Este termino es, básicamente, un PI de velocidad para lograr un comportamiento estable.

$$\xi = k_{p_i} e_{v_i} + k_{v_i} \int_{t=0}^{t} v_i \, dt \tag{35}$$

donde $e_{v_i} = v_i - v_{desired_i}$.

Para los seguidores, la velocidad deseada se obtiene con la siguiente expresión.

$$v_{desired_i} = k_i \left(\sum_{l \in \mathscr{M}_o} \alpha_{i,l} \, \beta'_{conn} \frac{\partial \eta}{\partial x_{l_1}} + \sum_{m \in \mathscr{M}} \alpha_{i,m} \, \beta'_{coll} \frac{\partial \iota}{\partial x_{m_1}} \right) \tag{36}$$

Sin embargo, para el líder necesitamos añadir ademas un PI de posición.

$$v_{desired_1} = k_p e_p - k_i \int_{t=0}^{t} x_1 \, dt + k_1 \left(\sum_{l \in \mathscr{M}_o} \alpha_{i,l} \, \beta'_{conn} \frac{\partial \eta}{\partial x_{l_1}} + \sum_{m \in \mathscr{M}} \alpha_{i,m} \, \beta'_{coll} \frac{\partial \iota}{\partial x_{m_1}} \right) \tag{37}$$

Además, para garantizar que el líder converge a la posición deseada añadimos un termino más, $\tilde{e}$.

$$\tilde{e} = e_{pos} + k_{pos} \int_{t=0}^{t} x_1 \, dt \tag{38}$$

$$e_{pos} = x_1 - x_{desired} \tag{39}$$

Entonces, finalmente, el algoritmo de control consta de los siguientes términos.

$$u = \sum_{l \in \mathscr{M}_o} \alpha_{i,l}\, \beta'_{conn} \frac{\partial \eta}{\partial x_{l_1}} + \sum_{m \in \mathscr{M}} \alpha_{i,m}\, \beta'_{coll} \frac{\partial \iota}{\partial x_{m_1}} - \xi + \lambda \tilde{e} \tag{40}$$

donde $\lambda = 0$ if $i = 2, 3, \ldots$ y $\lambda = 1$ if $i = 1$.

0.5 Simulación

Para testar nuestro control tenemos que llevar a cabo la simulación.

Simulador

Hemos trabajado con un simulador escrito en C++ y manejado a través de ROS. La visualización se lleva a cabo mediante R-VIZ.

Los archivos C++, en otras palabras, el propio simulador, emula el interior del Crazyflie. Consiste en PIDs previos al envio de la información a los motores más las ecuaciones dinámicas, que permiten que el simulador devuelva los estados que tendría el Crazyflie real.

PIDs implementados

Como está descrito en el Capítulo 2, los Crazyflie tienen cuatro PIDs en cascada. Mirando la Figura 5.1 podemos entender como están implementado en el código C++. Los Crazyflie pueden ser controlados enviado la posición deseada utilizando el *POSITION MODE* o enviando la actitud usando el *ATTITUDE MODE*. En la primera opción la información pasa a través de todos los PIDs y es necesario determinar el valor de la propia posición y del yaw deseado. Por otro lado, en la segunda opción la información entra directamente al PID de actitud y han de fijarse los valores del roll, pitch, yaw y thurst.

Algunas aclaraciones son necesarias. En la caja de *Bounded attitude* limitamos el valor de las actitudes, debido a las limitaciones físicas del hardware. Por otro lado, la caja de *Transformation to Body Frame* recibe la actitud en el sistema de coordenadas inercial ya que es el utilizado cuando mandamos la posición, y la transforma al no inercial ya que es la que Crazyflie utiliza. Además, en la caja de *Thrust adjust* replicamos la alteración que hace el firmware al thrust, consiste en sumarte un término que corresponde aproximadamente a $masa \times gravedad$ para conseguir que el dron flote de forma mucho más estable. Finalmente, la caja de *PWM transformation* es la encargada de distribuir el PWM en los cuatro motores siguiendo las siguientes ecuaciones.

$$\begin{aligned} PWM_{motor_0} &= Thrust - Roll\ rate + Pitch\ rate + Yaw\ rate \\ PWM_{motor_1} &= Thrust - Roll\ rate - Pitch\ rate - Yaw\ rate \\ PWM_{motor_2} &= Thrust + Roll\ rate - Pitch\ rate + Yaw\ rate \\ PWM_{motor_3} &= Thrust + Roll\ rate + Pitch\ rate - Yaw\ rate \end{aligned} \tag{41}$$

Ecuaciones dinámicas implementadas

Debemos implementar las ecuaciones dinámicas mostradas en el Capítulo 3 en nuestro simulador.

Tras el regulador explicado en el párrafo anterior disponemos de un PWM, que debemos convertir en el input que demanden las ecuaciones dinámicas, es decir, fuerzas y momentos. Estas fuerzas y momentos, a su vez, dependen de la velocidad angular. Por lo tanto, tendremos que calcular estas velocidades angulares con la ayuda de la ecuación 42.

$$RPM = 0.2685 \times PWM + 4070.3 \tag{42}$$

Después, únicamente tendremos que implementar las ecuaciones ya conocidas y sustituir las fuerzas y los momentos obtenidos en base a las velocidades angulares.

ROS

Teniendo el código de control y el simulador podemos simular nuestra nube de quadrotors. Cada Crazyflie correrá el código de control y el

código de simulador, esto se realiza a través de un archivo launch que nos permite lanzar todo de forma coordinada.

El flujo de información sera manejado por ROS, el cual se basa en nodos y topics. Podemos ver el esquema ROS de nuestra simulación en la Figura 11.

Los nodos están representados por medio de círculos, las flechan indican el flujo de información entre nodos y topics y estos últimos están posicionados en medio de cada flecha. Cada nodo y topic tiene un nombre propio para poder *llamarlo*. Además, los nodos y topics de un Crazyflie en concreto van precedidos de su nombre, por ejemplo *crazyflie_1/Control* y *crazyflie_1/Forces*.

Para cada Crazyflie (Fig. 12), en nuestro caso seis, tenemos un nodo *Control*, otro llamado *ForcesToRollPitchYaw* y un último llamado *DynamicalMoldel*. El nodo *ForcesToRollPitchYaw* es el encargado de traducir la fuerza que entrega el nodo *Control* a una actitud, que es el input del nodo de simulación.

Por otro lado, cada agente tiene cuatro topics. En el topic *crazyflie_i-/Forces* el control publica la fuerza deseada y el nodo *crazyflie_i/Forces-ToRollPitchYaw* esta subscrito para recibir la informaciónn. Además, este último nodo publica en *crazyflie_i/AttitudeCommand* la actitud y el thrust calculados, topic en el que el simulador esta subscrito.

Para enviar información entre Crazyflies creamos el topic *crazyflie-_i/OutPosition* donde cada uno de ellos publica su posición y el resto esta subscrito. Dentro del algoritmo de control es donde se discrimina que información podemos utilizar y que no, dependiendo de si se encuentran dentro de el radio de sensibilidad o no.

Por último, tenemos el nodo de la visualización *R-VIZ* el cual también está subscrito a *crazyflie_i/OutPosition* para conocer la posición de todos los drones y poder imprimirla.

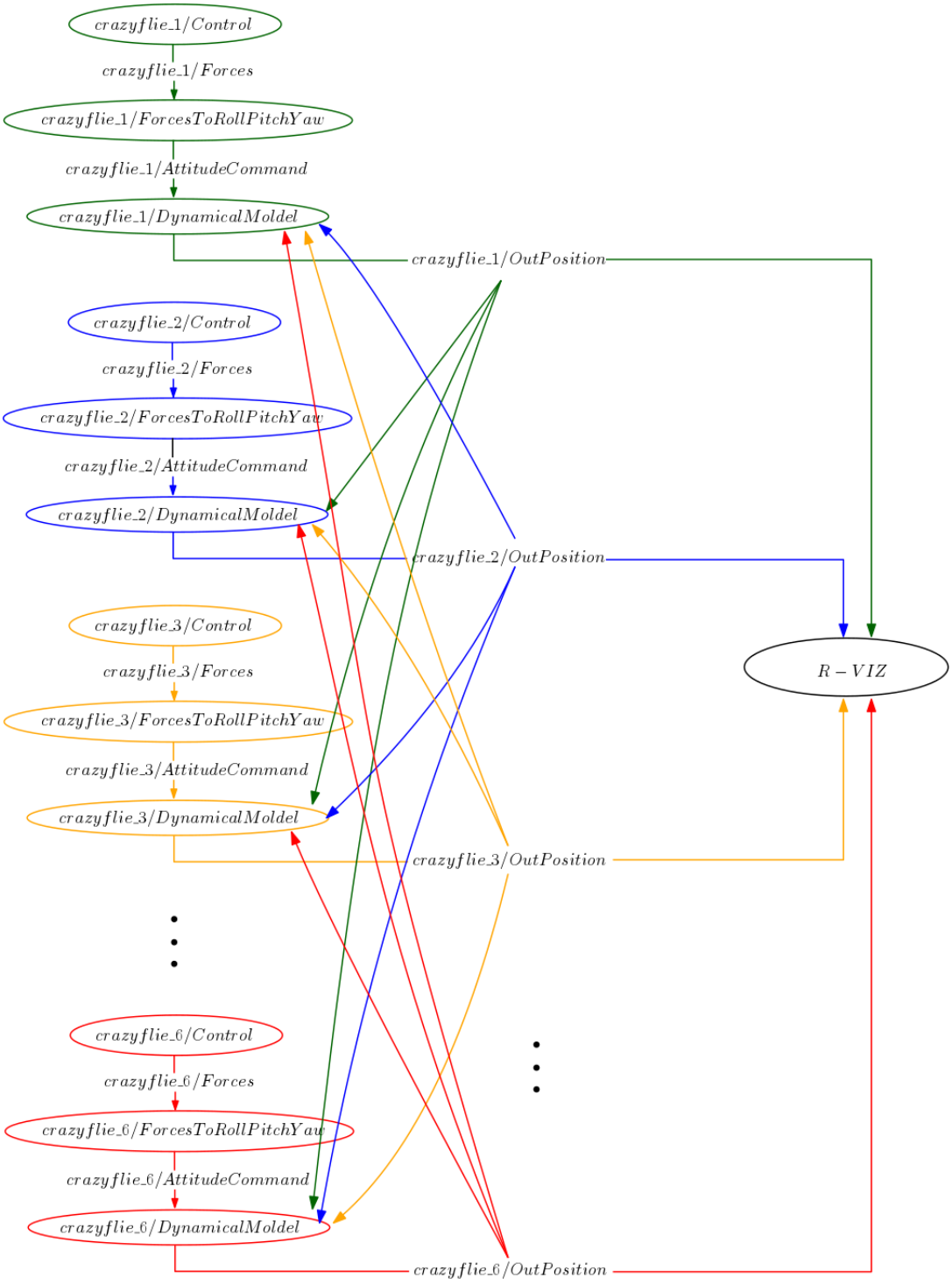

crazyflie_1/Control
crazyflie_1/Forces
crazyflie_1/ForcesToRollPitchYaw
crazyflie_1/AttitudeCommand
crazyflie_1/DynamicalMoldel
crazyflie_1/OutPosition
crazyflie_2/Control
crazyflie_2/Forces
crazyflie_2/ForcesToRollPitchYaw
crazyflie_2/AttitudeCommand
crazyflie_2/DynamicalMoldel
crazyflie_2/OutPosition
crazyflie_3/Control
crazyflie_3/Forces
crazyflie_3/ForcesToRollPitchYaw
crazyflie_3/AttitudeCommand
crazyflie_3/DynamicalMoldel
crazyflie_3/OutPosition
R − VIZ
crazyflie_6/Control
crazyflie_6/Forces
crazyflie_6/ForcesToRollPitchYaw
crazyflie_6/AttitudeCommand
crazyflie_6/DynamicalMoldel
crazyflie_6/OutPosition

Figure 11

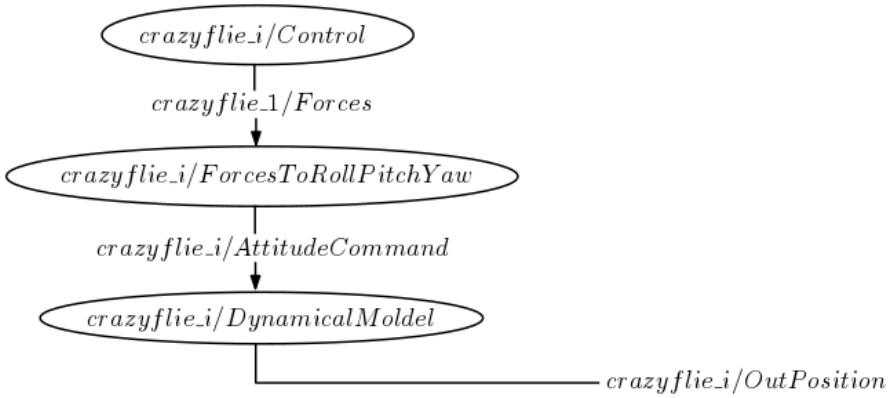


Figure 12

Experimento

Para nuestros experimentos utilizaremos seis Crazyflies posicionados como se muestra en la Figura 13.

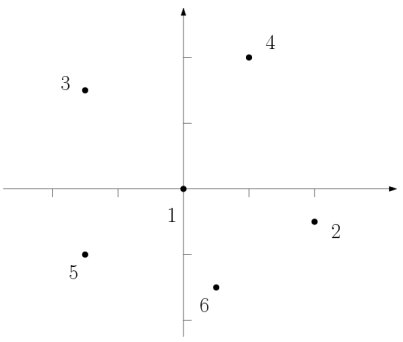


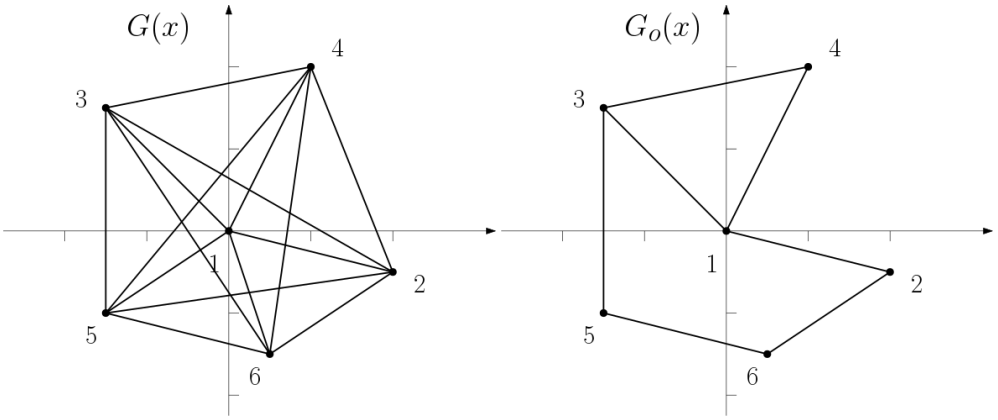
Figure 13

Para el *Experimento A* definimos la gráfica de colisiones (Fig. 14a), y la gráfica de conectividad (Fig. 14b).

Dentro de este experimento, vamos a testar dos trayectorias diferentes para analizar como se comporta el algoritmo en distintas situaciones. Ambas trayectorias constan de siete puntos:

I (0, 0, 0.2), (0, 0, 2), (0, 0, 5), (4, 5, 3), (-2, 4, 2), (3, -2, 3), (1, -1, 2)

II (0, 0, 0.2), (0, 0, 2), (-3, -3, 2), (-1, 4, 4), (4, -2, 1), (3, 3, 3), (1, 1, 1)

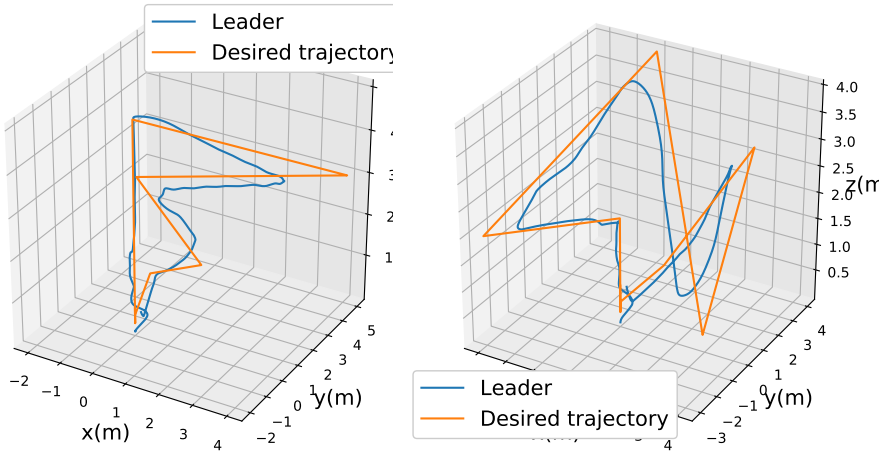


(a) Gráfica de colisiones (b) Gráfica de conectividad

Figure 14: Experimento A

Resultados

En este capítulo se exponen los resultados obtenidos en simulación de los experimentos previamente descritos.



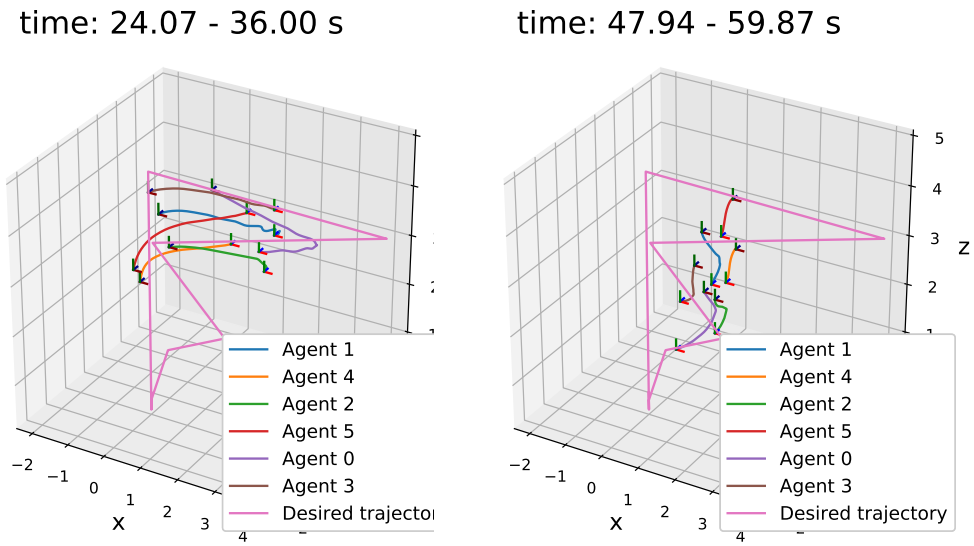
(a) Traj.I (b) Traj.II

Figure 15: Ruta seguida por el líder en el *Experimento A*

Analizamos la ruta que sigue el líder durante la ejecución de la trayectoria (Figura 15). Como podemos apreciar, tanto en la trayectoria I como en la II su comportamiento es suave y sin oscilaciones. Sin embargo, la trayectoria que realiza no coincide exactamente con

la trayectoria deseada porque el líder también cuenta con el efecto de los demás drones de la nube. Es decir, los términos de conectividad y colisión no permiten total libertad al líder, que tiene que adaptarse a los requisitos de cada momento. Por otro lado, es cierto que eso puede solucionarse ajustando los parámetros de control solo para el líder, pero hemos decidido no hacerlo para lograr la máxima generalidad posible.

Para dar una idea de como evoluciona el sistema añadimos la Figura 16.



(a) (b)

Figure 16: Comportamineto del enjambre en el *Experimento A* Traj. I

A fin de clarificar la conducta de los robots durante el experimento, analizamos las distancias entre drones (Figura 17). Estas distancias nos permiten comprobar si el algoritmo funciona como esperamos, es decir, si no se produce ninguna desconexión o colisión entre drones. Y como podemos observar, las distancias no exceden ningún limite.

Por otra parte, analizamos la señal de salida del control entregada para lograr el comportamiento deseado. Para ello analizamos un Crazyflie aleatorio.

Para el agente 3 imprimimos el término de conectividad y el de

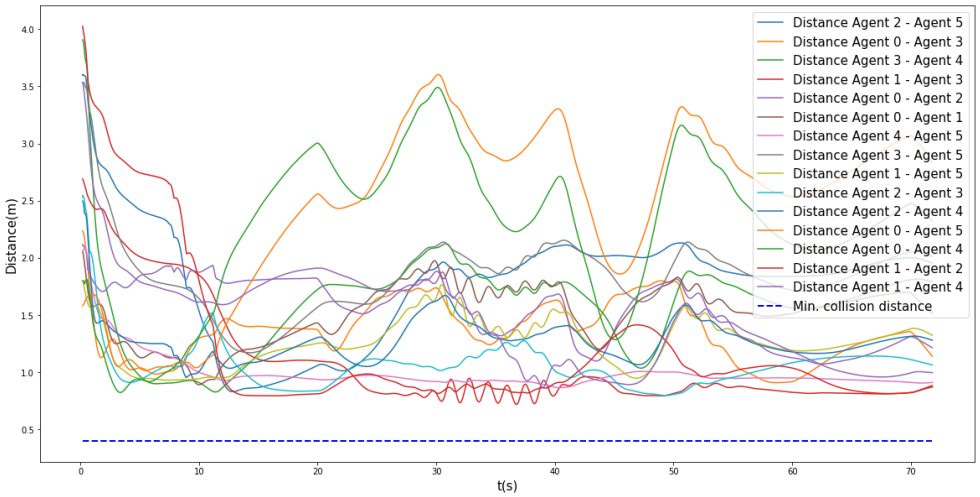
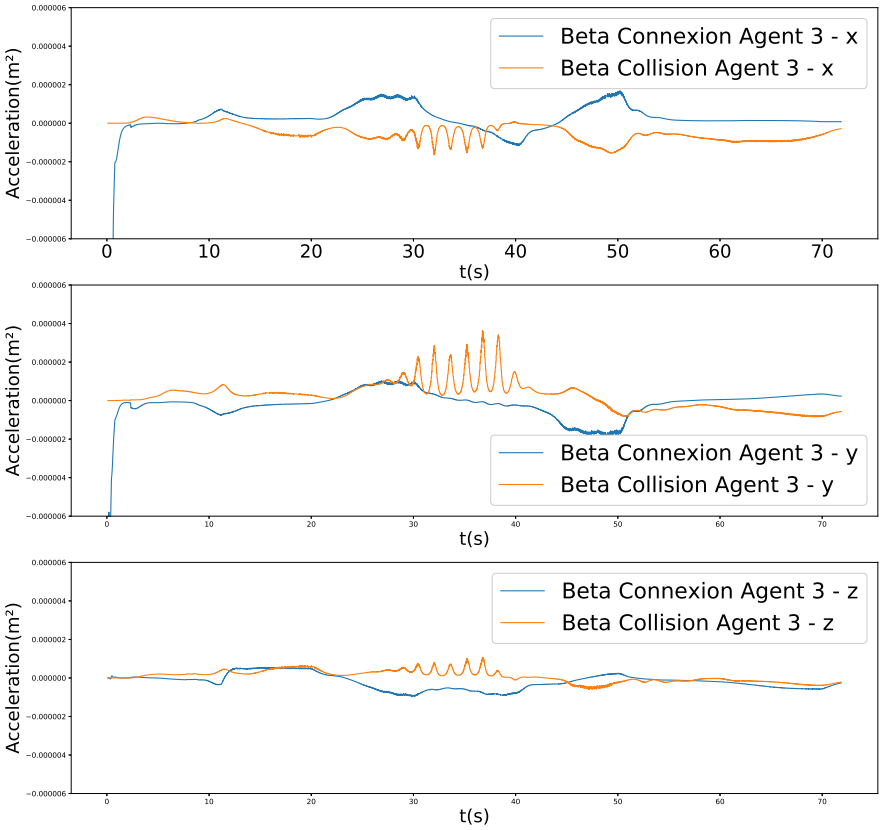


Figure 17: Distancias en el enjambre en el *Experiment A* Tray. I

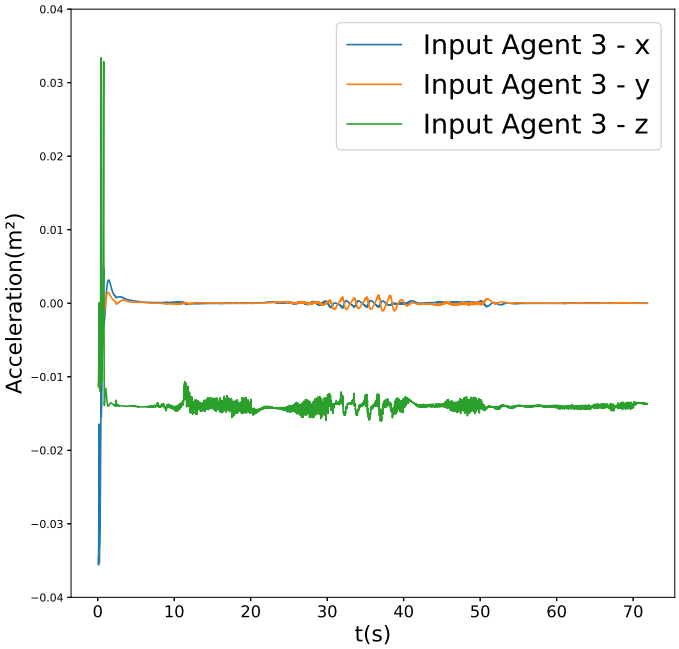
no-colisión en la Figura 18a y la señal de fuerza en la Figura 18b.

Que el término de conexión o el de colisión sean prácticamente constantes se traduce en que no se ha dado ninguna aproximación o alejamiento problemático durante el experimento. Un cambio de valor en el término de conexión quiere decir que el dron esta intentando mantener la conexión con otro dron que se está alejándose. Por otro lado, las oscilaciones que observamos en el término de no colisión nos muestran como, al estar ante peligro de choque, la señal cambia de valor de forma muy rápida para intentar solventar el problema, y comprobamos que lo logra al estabilizarse otra vez la señal.

En relacion a la fuerza, su valor en el eje z es significativamente distinto al de los ejes x y y ya que el efecto de la gravedad está presente y la dinámica es muy distinta y más compleja. Por otro lado, podemos observar que aún con las variaciones que puedan darse el los términos explicados en el anterior párrafo, la fuerza se mantiene muy estable. La linealidad de la fuerza de entrada en un robot aéreo es una característica muy buscada ya que la gran inercia que presentan estos vehículos hace muy difícil o casi imposible el control si existen oscilaciones en ella.



(a) Términos de colisión y conexión en el *Exp. A* Tray. I para el agente 3.



(b) Fuerza en el *Exp. A* Tray. I para el agente 3.

Figure 18

0.6 Conclusión

Hemos estudiado un problema Lider-Follower en un sistema multi-agente de drones donde la comunicación entre agentes esta solo permitida cuando se encuentran lo suficientemente cerca.

Empezamos con la formulación del problema y sus restricciones. Específicamente, qué información es proporcionada al líder, cuál es su tarea y cómo deben comportarse los seguidores con el objeto de mantenerse juntos formando una nube de drones pero sin chocar entre ellos.

Más adelante, hemos introducido el dron con el que vamos a trabajar en los experimentos y su modelo dinámico. Luego hemos propuesto el algoritmo de control para conseguir el comportamiento deseado.

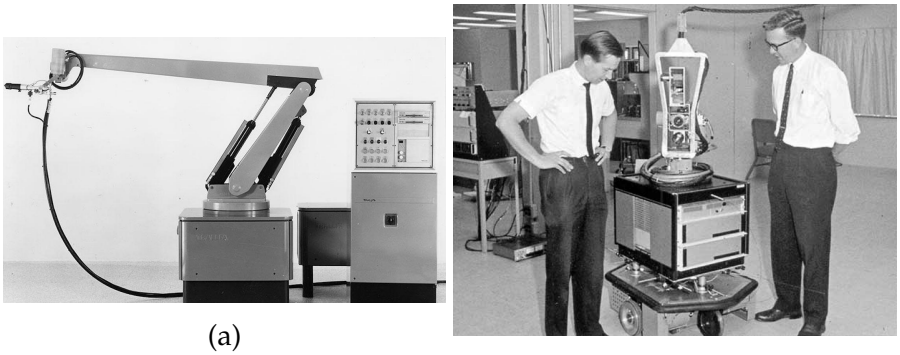
Finalmente, describimos como hemos implementado el control a través de ROS y exponemos los resultados de simulación. Como conclusión podemos decir que el control es suficientemente estable y robusto y que la meta ha sido alcanzada de forma satisfactoria.

Chapter 1

Introduction

A growing interest has been shown in robotics those last years. A wide number of industries require robots to replace men in not only dangerous or boring situations, but also in application that could not be done by humans. A huge part of this research is dedicated to mobile robots.

Robotics is the branch of technology that deals with robots, which are programmable machines which are usually able to carry out a series of actions autonomously, or semi-autonomously [9].



(a) (b)

Figure 1.1

1.1 Robot History

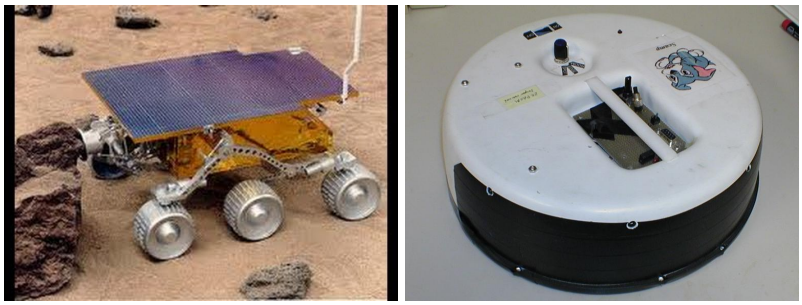
The earliest robots, as we know them, were created in the early 1950s by George Devol, an inventor from Louisville. He invented and patented a reprogrammable manipulator called *Unimate*, from *Universal Automation* (Fig. 1.1a). For the next decade, he attempted to sell his product in the industry, but did not succeed. In the late 1960s, the engineer Joseph Engleberger acquired Devol's robot patent and was able to modify it into an industrial robot and form a company called *Unimation* to produce and market the robots. For his efforts and successes, Engleberger is known in the industry as *The Father of Robotics* [10].

Academia also made much progress creating new robots. In 1958 at the *Stanford Research Institute*, Charles Rosen led a research team in developing a robot called *Shakey* (Fig. 1.1b). *Shakey* was far more advanced than the original *Unimate*, which was designed for specialized industrial applications. *Shakey* could wheel around the room, observe the scene with his television 'eyes', move across unfamiliar surroundings, and to a certain degree, respond to his environment. It was given his name because of his wobbly and clattering movements [11].

In 1974, Victor Scheinman formed his own company and started marketing the *Silver Arm*, that was capable of assembling small parts together using touch sensors, the beginning of the pick and place industrial robots [12]. A few year later, the soft gripper was designed by Shigeo Hirose to wrap around an object. In 1978, the four axis *SCARA* robot arm (Selective Compliance Assembly Robot Arm) was created. It was the best robot used for picking up parts and placing them in another location and it was introduced to assembly lines in 1981 [13].

The *Stanford Cart* built in 1970 was rebuilt by Hans Moravec in 1979 by adding a more robust vision system allowing greater autonomy and being able to do mapping [14]. In 1981, Takeo Kanade built the direct drive arm, that was the first to have motors installed directly into the joints of the arm. This change make his designs much more accurate than previous robotic arms [15].

In 1997, *The Pathfinder Mission* landed on Mars. Its robotic rover *Sojourner*, rolled down a ramp and onto Martian soil in early July (Fig.



(a) (b)

Figure 1.2

1.2a). It continued to broadcast data from the Martian surface until September of the same year [16].

In the beggining of the twenty century, *iRobot* released the first generation of *Roomba* (Fig. 1.2b), the robotic vacuum cleaner, that was a rapprochement of the robots to the ordinary life [17]. In 2005, *The Korean Institute of Science and Technology*, created *HUBO* and claimed it was the smartest mobile robot in the world. This robot is linked to a computer via a high-speed wireless connection; which does all of the thinking for the robot [18].

In 2005, the revolution of robotics started, were a lot of improvements and more robots were developed. As examples we can remark the followings.

In 2006, *Cornell University* revealed its *Starfish* robot (Fig. 1.3a), a four-legged robot capable of self modeling and learning to walk after having been damaged. In 2007, *TOMY* launched the entertainment robot, *i-sobot* [19] (Fig. 1.3b), a humanoid bipedal robot that can walk like a human and performs kicks and punches and also, some entertaining tricks and special actions under 'Special Action Mode'.

Robonaut 2, the latest generation of the astronaut helpers, was launched to the Space Station aboard *Space Shuttle Discovery* on the *STS-133* mission in 2011 [20]. It is the first humanoid robot in space, and although its primary job for now is teaching engineers how robots behave in space; the hope is that through upgrades and advancements, it could one day venture outside the station to help space walkers make

(a) (b)

Figure 1.3

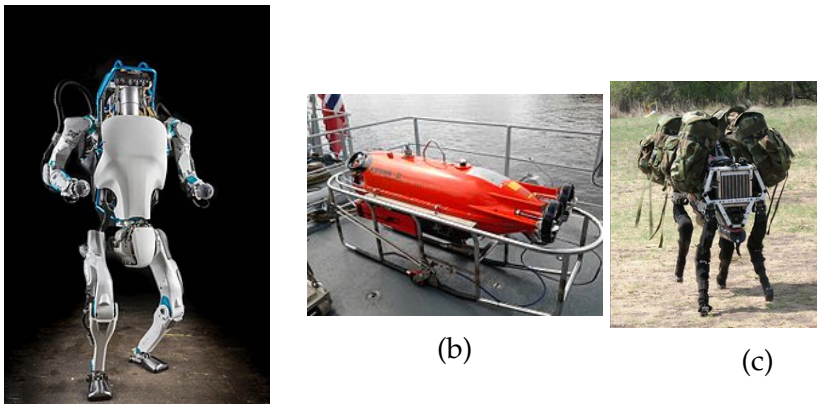
repairs or additions to the station or perform scientific work.

Commercial and industrial robots are now in widespread use performing jobs more cheaply or with greater accuracy and reliability than humans. They are also employed for tasks which are too dirty, dangerous or dull to be suitable for humans. Robots are widely used in manufacturing, assembly and packing, transport, Earth and space exploration, surgery, weaponry, laboratory research and mass production of consumer and industrial goods among others.

1.2 Mobile Robots

As we can see, the robotics, especially the mobile robots, have change the history in multiple ways. As we all know mobile robots can be defined as a platform with large mobility within its environment (air, land, underwater) that follows functional characteristics as perception ability that allow it to sense and react in the environment; tools to be capable to move and certain level of autonomy.

Nowadays, exists a huge variety of mobile robots, specialized for different environments and achieving different grades of complexity. As an example, we expose some of them. In the Figure 1.4a we can see a ground robot developed by *Boston Dynamics* in 2013 [21]. 'Atlas', the *Agile Anthropomorphic Robot*, is a bipedal humanoid robot. He easily runs up 2' high steps onto a platform and performs a number of tasks that would have been impossible for the previous generation of



(a) (b) (c)

Figure 1.4

humanoid robots. In the Figure 1.4b appears an Unmanned Underwater Vehicle called *Pluto Plus* [22] designed for use in underwater mine identification and destruction by militaries. In the Figure 1.4c we can see another creation of *Boston Dynamics* which runs at 10 kilometers per hour, climbs slopes up to 35 degrees, walks across rubble, climbs muddy hiking trails, walks in snow and water and carries up to 150 kilograms loads [23].

During the last years a lot of research has been focused on aerial robot technology, due its versatile application and the wide area that is recently open to develop. Regarding aerial robots, we highlight the unmanned aerial vehicle (UAV), commonly known as drone, define as an aircraft without a human pilot aboard.

Drones are already breaking barriers in the way companies do business. Huge corporations like *Amazon* and *Google* are testing ways to deliver packages with drones. *Facebook* is using drones to provide Internet connections in remote locations. And there's even a start-up that's using unmanned aircraft to deliver tacos to your door [24].

The flight of UAVs may operate with various degrees of autonomy which involves the use of mechatronics, artificial intelligence, multi-agent system and algorithms of control. Drone technology is constantly evolving as new innovation and big investment are bringing more advanced drones to the market every few months.

Figure 1.5

1.3 Drone state of art

Nowadays, unmanned aerial vehicles have a promising future in many sectors. By making these robots intuitive, friendly, easy to use and, even, manufactured by a basic 3D printer, thousands of potential applications and services are made possible. With this rapid growth, most markets have been flooded by all kinds of drones, from fixed-wing aircrafts to multirotor crafts or a combination of the two. Beyond the known set of applications such as monitoring and delivery, these vehicles can perform very difficult tasks due to their high maneuverability and their high level of stability, even losing a propeller during their flight time.

Inside UAV family we find the quadrotor, also called quadcopter, which is a multirotor helicopter that is lifted and propelled by four rotors. They are a popular robotic platform due to their agility and simplicity model. Quadcopters can be large enough to carry cameras and smartphone-grade computers, but also they can be extremely small and basic.

The quadcopter and its simple format with very few moving parts has rapidly become a favorite vehicle for remote control enthusiasts and is widely being used in researcher. A large majority of the quadcopters were originally built by hobbyists who understood the simplicity of the vehicle. By adding four motors and four propellers to a lightweight frame constructed of light wood, carbon fiber, or fiber

glass then connecting it to a remote control transmitter via a small control board fitted with a gyroscopic stabilization system and connected to a battery these craft were relatively simple to construct.

The rapid advances in computing power, the efficiency of coreless or brushless motors, smaller microprocessors, development of batteries and gyroscopic and accelerometer technology has led to a proliferation of quadcopters design. The first quadcopters were not designed for acrobatic flight as the development was concentrated on simple stable flight patterns but now this has all changed. Micro and even nanoquadcopters are produced in this moment can perform intricate aerobatic moves, flips and barrel rolls that years ago would have been unthinkable. In the early days of flight, quadcopters were seen as possible solutions to some of the persistent problems in vertical flight; torque-induced control issues (as well as efficiency issues originating from the tail rotor, which generates no useful lift) can be eliminated by counter-rotation and the relatively short blades are much easier to construct.

Drones and quadcopters, with extensive history and capabilities, are the devices of future. In time to come, humans will find it difficult to imagine life without them.

During these last years progress was being made in a variety of ways. Some of the latest and more developed drones nowadays and its characteristics are exposed in the following paragraphs.

Lily (Fig. 1.5) was launched on 2015 [25]. It is a sleek drone that take off when thrown in the air and could navigate around objects, something no drones were able to do at the time. Each battery offers up to 18 minutes of autonomous flight time of object tracking and automatic return to home.

HexH20 the waterproof multirotor of *Quadh20* was launched on 2017 [26] (Fig. 1.6a). Its main attraction is its powerful waterproof that allow him to take photography above an below the water. It has a *DJI* N3 intelligent controller and a 4K camera supplied by the *DJI Lightbridge 2* giving a range of up to 3.5 kilometers.

The battery autonomy of the drones in average is not high, so *Air-*

(a)

(b)

Figure 1.6

Robotics have created a original solution for automated industrial drones [27]. As we can see in the Figure 1.6b it is not an ordinary drone, it is a entire box that offers an end-to-end automated industrial drone platform. The system contains airbase landing dock, drones and a software that enables mission planning with repeatable pre-programmed flight scheduling, eliminating the need for a drone pilot or operator. It has an innovating charging system, with a set of mechanical devices that allows the drone to land, replace a battery, and take off instantly after.

Regarding 'everyday' drones we can mention *Sora Ruka*, a drone developed by *Rakura* [28]. It was launch in 2016 as a deliver drone. The transported box, with a maximum of 2 kilos, is attached to the drone by an operator who also determines the pickup point. The drone make the trajectory autonomously, release the box in the drop point and returns to the starting point. The first use was in a golf court delivering golf balls and food, however, in march of 2018, it did its first deliver to a private residence. In the future the company want to use it to deliver in scarcy populated or mountain areas. The longest flight has been 12 kilometers.

Finnaly, the cutting edge in drones technology is the transport of people. The companies that are now developing the most impressive technology in this area are *Italdesign* and *Airbus* working together in prototype called *PopUp* [29] (Fig. 1.7). It is a modular system that uses a two passengers capsule that can be attached to a motorized base for driving or to a quadcopter propellers to flight. It is operated by a Artificial Intelligence System that determines the preferable route to follow, being able to pick the driving or flying module to reach de des-

tination. The autonomy as a car is the 100 kilometers and as a quadcopter 130 kilometers with a electric battery. It was first showed in *Geneva Motor Show* in 2017 and the first flying test was on November 2018 with a 1/4 scale prototype.

Figure 1.7

As we can see, there have been a huge development of the drones technology over the last years, however there is a wide avenue to improve, optimize and create in this area.

1.3.1 Example application

The drones are used nowadays in mining, agricultural, military, security, inspection,photograph, research, game or construction industry. They have plenty applications, but, because its innovation and originality we are going to show two of them.

Zipline [30] is a organization stated on October 2016 which operates the world's only drone delivery system at national scale to send urgent medicines, such as blood and vaccines, to those in need. It is placed in Rwanda, Africa, and they carry out 500 delivers per day in 30 minutes, or less, in a radio of 80 kilometers with a drone that can hold 1.8 kilograms of weight. This process start with the necessity of medical resources in a remote clinic, with a text message they ask for the

medical product, those products are stored an the *Ziplice Distribution Center*, enabling immediate access to even the most sensitive os scarce items. They are packaged there and prepared for flight, maintaining cold-chain and product integrity. The box if attached to the drone and it is powered from 0 kilometers per hour to 100 kilometers per hour in 1 second with a launcher. The drone guide himself to the desired point and release the package that fall without danger with the help of a simple parachute. After that, the drone return to the *Distribution Center* for a quick pit stop before taking off again, so no infrastructure is required at remote clinics.

Another interesting application appears when we fuse drones with other way of transport, like, for example, boat. This is the idea of the *Saildrone* launched in November of 2017 [31]. The ocean drone collects data on climate changing all around the ocean and is becoming a invaluable tool for tracking important indicators. *Saildrone* is an autonomous 6-meter-high vessel made of carbon fiber will top off each drone. It moves at only 5-8 kilometers per hour, the ideal speed for data collection. A GPS guides the drones and a remote rudder controls them and is equipped with 16 sensors for testing a number of important variables. One advantage of having an unmanned boat carrying out the duties that are typically performed on manned research ships is the significant cost differences, the company has stated that its drones can operate at only 5 percent of the total operating cost of the manned vessel equivalent. And what is more, they are completely wind and solar powered, and built to endure extreme weather conditions.

1.4 Robot Swarms

There are situations where a single drone is not enough, because its capacities are not suitable for the application or mission requested. These limitations are overcome by using a swarm of drones. In nature collective behaviors enable animals to achieve remarkable, colony-level feats through the distributed actions of millions independent agents. These collective behaviors see in bacteria colonies, fish schools, ant and bee colonies or bird crowdsare are inspiring engineers to build

simple mobile robot swarms, in order to copy its flexibility and robustness. For example, a group of drones could collaborate to build tensile structures, map wide extension, move heavy objects or even for artistic choreographies. Furthermore, strong needs emerge in the field of security for observation in hostile, difficult or large areas, as well as for monitoring of sites and populations during specific events.

A swarm intelligence system usually consist of a group of simple individuals autonomously controlled by rules and local interaction. These individuals are not necessarily unwise, but they role is relatively simple compared with the global activity achieved. Swarm intelligence takes the full advantage of the swarm without the need of centralized control and global model, and provides a great solution for large-scale sophisticated problems. In the field of robotics it could be the coordination of a large number of simple robots in with the desired behavior emerges from the interaction between robots and the interaction of robots with the environment.

The main characteristic of swarm control is the decentralization. No single agent can access to all the information in the network. The agents in the swarm follow their own rules according to local information, typically due to near distance. The group behaviors emerge from these local rules while information is being exchanging between the individuals in the swarm. The rules are also the key component to keep the whole structure flexible and robust even when the sophisticated behaviors are emerged.

Coordination of swarm systems has been receiving a significant attention recently. This research interest is provoked in part by the rapid development of distributed control of multi-agents systems. An example of this is show in [1]. This coordination of robots swarms has numerous practical applications. One typical example of such application is the relative attitude keeping problem in the context of deep space interferometry [2] [3].

Much effort has been made towards coordination of networked systems. In [4] we can see Leader-Follower cooperative attitude synchronization where there exists a time-varying leader. Communication delays and dynamic topologies are considered in [5], where collision avoidance behavior is highlighted.

Group dispersion and cohesion behaviors are often very important for coordination of multi-agent systems. Group dispersion is to ensure minimum safety distance between different agents and group cohesion is to maintain the connectivity once two agents are connected. A variable structure approach is taken in [6] to guarantee cooperative swarm tracking for a group of agents with or without a leader. Connectivity maintenance approaches are proposed in [7] and [8], where a bounded or an unbounded input function is introduced. The authors of [32] designed a so-called region-based shape control algorithm to force a group of mobile robots modeled by Lagrange dynamics to move into a desired region while maintaining a minimum distance among themselves. However, this algorithm relies on the strict assumptions that the minimum distance be small enough and all the followers have access to the information of the desired region.

1.5 Motivation

The main motivation for this thesis is get deeply in the aircraft control theory and technology. Drones are now breakground technology and I wanted to widen my knowledge in the area.

The sciences have fascinated me since childhood. My passionate interest in math and physics stemmed from appreciating technology, in special, the implementation of these sciences in the robotics. Swarm control is, nowadays, a interesting topic to develop in research. Investigate how can we apply natural coordinated behaviors in the robots to build optimize systems has amazed me since the very first moment.

1.6 Outline

At the beginning, in Chapter 0 there is a summary of the whole thesis written in Spanish. In Chapter 1 there is a introduction on robots and, in special, on drones an swarm control. Chapter 2 is where the problem formulation is exposed and in Chapter 3 we explain how is the device used in the experiment and its dynamics. Further more, in

Chapter 4 we can find the algorithm used to achieve the objective and how is it constructed. Finally, in Chapter 5 we show the experiment done with its results and in Chapter 6 we make a conclusion of the whole thesis.

Chapter 2

Objetive

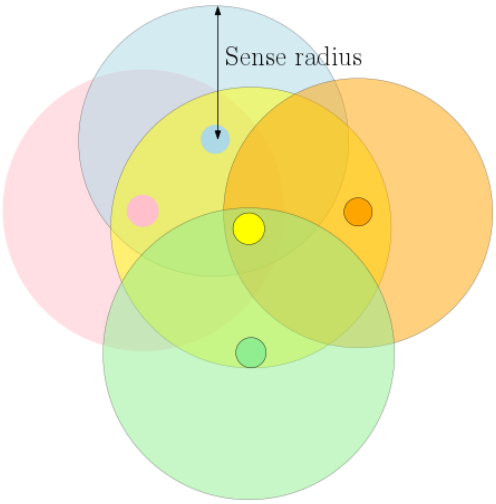
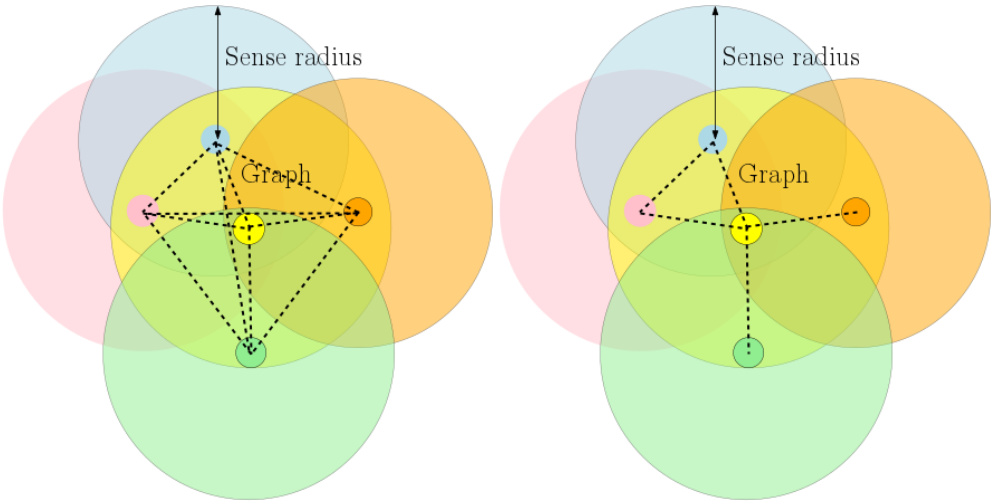


Figure 2.1: Quadrotor swarm

The objective of this thesis is to implement a adaptive control on a swarm of nanoquadrotors (Fig. 2.1) to perform a Leader-Follower behavior. We consider that the leader is the only robot who has to converge to desired points given to perform the whole trajectory. Followers have to follow the leader in compliance with several constraints fixed. Inter-robot collisions must be avoided and some initial connectivity between robots have to remain connected. In addition, we consider that each robot has a limited sensing radius, which implies that the robots can sense each other without colliding but also that can not know the position of those ones that are not in the sense area.



(a) Collision avoidance graph (b) Connectivity maintenance graph

Figure 2.2

The collision avoidance is based in not exceed a minimum distance between robots. This distance depend on the size of the quadcopter used and its inertia, that can change drastically with the drone weight and design. For guarantee collision avoidance we model a graph, $G(x)$ (Fig. 2.2a), in which all of the agents are connect to the rest and in the moment they are too close they try to move away.

Connectivity maintenance takes care of keeping drones in the sense area of another drone, so they are able to know the position of each other and can cooperate together. Is the main reason why the swarm is, actually, a swarm, and the agents does not go away. To achieve this behavior we model the topology of the multi-robot network through a graph, $G_o(x)$ (Fig. 2.2b), that defines the connectivity between drones we want to keep, being all the robots part of an edge. This connectivity translate into a maximum distance between robots involved in the edge that can not be exceed, because the sense radio of each drone is finite, and if they go further connectivity breaks. The initial graph should be kept in the whole performance, which also implies that some of the follower are, actually, following another follower. This fact make the behavior more complicated and sophisticated than the case of all them following the leader directly.

The motivation for that is mainly potential cooperative tasks that

the robots have to accomplish, whose details are provided only to a leader robot. Then, the leader has to navigate the entire team to the point of interest, which is guaranteed via graph connectivity. The majority of the Leader-Follower works treats the leader as an exogenous moving source with bounded velocity and/or acceleration, and they consider that those followers that are connected to the leader have access to its state. In this work, we consider that the leader's velocity is unknown to the rest of agents. In fact, the only available leader information is its pose, only to the follower robots connected to it, as by the limited sensing radius assumption.

Figure 2.3: Cooperative drones

Formally, the problem treated is the following:

Consider $N > 1$ spherical autonomous robots, with $N = \{1, \ldots, N\}$, operating in R and described by the spheres $A_i(x_i) = \{y \in R : \|x_i - y\| < r_i\}$, with $x_i \in R$, being i its mass center and $r_i \in R_{>0}$ its bounding radius.

Without loss of generality, we assume that the robot $i = 1$ corresponds to the team leader, whereas $i > 1$ are the followers, which belong to the set $F = \{2, \ldots, N\}$. The task of the leader is to navigate to a given desired pose, denoted by $x_d \in R_n$, and the entire team is responsible for guaranteeing collision avoidance as well as connectivity maintenance.

The limited sensing radius is denoted $d_{conn,i} \in R_{>0}$, with $d_{conn,i} > max_{j \in N}\{r_i + r_j\}$. Based on this, we model the topology of the multi-robot network, used to guarantee collision avoidance, through the graph

$G(x) = (N, \mathscr{E}(x))$, with $\mathscr{E}(x) = \{(i,j) \in N^2 : \|x_i - x_j\| \leq min\{d_{conn,i}, d_{conn,j}\}\}$. We further denote $M(x) = |\mathscr{E}(x)|$. Given the m-th edge in the edge set $\mathscr{E}(x)$, we use the notation $(m_1, m_2) \in N^2$, where m_1 is the tail and m_2 is the head of edge $m \in \mathscr{M}(x)$, where $\mathscr{M}(x) = \{1, \ldots, M(x)\}$ is an arbitrary numbering of the edges $\mathscr{E}(x)$.

We also need to guarantee that some edges, denoted by $\mathscr{E}_o \subset \mathscr{E}(x)$, are preserved to accomplish with the connectivity maintenance. Denoting $M_o(x) = |\mathscr{E}_o(x)|$ and $\mathscr{M}_o(x) = \{1, \ldots, M_o(x)\}$ an arbitrary numbering os the edges $\mathscr{E}_o(x)$. We call l the edges in the set of edges $\mathscr{E}_o$, being $(l_1, l_2) \in N^2$, where l_1 is the tail and l_2 is the head of edge $l \in \mathscr{M}_o(x)$.

After all defined we can sum up the posed problem:

Consider N autonomous robot with its own dynamics. The objective is to develop a decentralized control strategy that achieves 1) convergence of the leader robot's pose to the desired one, 2) inter-robot collision avoidance, and 3) connectivity maintenance between a subset of the initially connected robots [33]:

1. $$\lim_{t \to \infty} x_1(t) - x_d = 0$$

2. $$A_i(x_i(t)) \cap A_j(x_j(t)) = \oslash, \forall t \in \mathrm{R}_{\geq 0}, \mathrm{i}, \mathrm{j} \in \mathrm{N}, \mathrm{i} \neq \mathrm{j}$$

3. $$\|x_{l_1}(t) - x_{l_2}(t)\| \leq min\{d_{conn,l_1}, d_{conn,l_2}\}, \forall t \in \mathrm{R}_{\geq 0}, \mathrm{l} \in \mathscr{M}_0$$

Chapter 3

Crazyflie 2.0 Nano Quadcopter

3.1 Hardware

The proposed experiment is going to be implemented in Crazyflie 2.0 NanoQuadcopter. We have chosen it because is one of the smallest quadrotors available on the market, which makes it an ideal candidate for swarm research applications that require to be flown near people and be able to test in small flight arenas. Also, the price is relatively cheap, fact that is convenient due to we need several of them in the swarm.

Crazyflie 2.0 (Fig. 3.1) measures 92 millimeters from propeller to propeller and weighs 34 grams with optical markers mounted on it, to be seen by a motion capture system if necessary. It has a flight time of around 5 minutes. The small size, as we said, presents many challenges in using it as a research platform, as well as its small inertia which requires controllers that can react with very little latency. Its payload capacity is also limited, making it difficult to add additional sensors.

The Crazyflie 2.0 is equipped with two microcontrollers. The first one, is used to run the main application and the second to handle power management and the radio for connection with the computer. The quadrotor is also equipped with an IMU, Inertial Measurement

Figure 3.1: Crazyflie, batery and radio

Unit, the *MPU-9250*. It contains a 3 axis gyroscope and a 3 axis accelerometer [34]. The Crazyflie has, as well, an on-board magnetometer and barometer.

The battery used by the Crazyflie 2.0 is a single cell LiPo battery, it supplies 3.7 volts and has a capacity of 240 milliampere-hours. The battery also comes with a *Protection Circuit Module* attached to it that prevents the user from under or over charging the battery or from shorting it [35]. The battery is easily removable from the quadrotor so they could be charged in parallel and reduce delay between experiments.

The Crazyflie has four brushed DC motors. The motors are coreless which in theory provides faster acceleration. They can produce 12000 revolution per minute per volt, with a nominal voltage of 4.2 volts. The propellers are conventional 45 millimeters plastic propellers. They have a tendency to bend during collisions and so experimenters should make sure they regularly change the propellers for new ones in order to give the controller optimal conditions [36].

The Crazyflie is equipped with a *Nordic Semiconductor nRF51822* which handles radio communication. The easiest way to communi-

cate with the chip is to use the Crazyradio. The Crazyradio is a USB dongle that integrates a *Nordic Semiconductor nRF24LU1+*. The chips can communicate with each other over the 2.4 GigaHertz ISM band. Both chips can be reprogrammed [36].

3.1.1 Physical Parameters

The precise measurement of certain physical parameters is the key to create a simulation environment that correctly describes the quadcopter behavior. In the Table 3.1 we assign the name of parameters we will use in the next chapters and their value for the Crazyflie 2.0. [37].

Symbol	Description	Value
m	Total mass	$0.33Kg$
d	Arm length	$39.73 \times 10^{-3}m$
r	Rotor radius	$23.1348 \times 10^{-3}m$
I_{xx}	Principal Moment of Inertia around x	$1.395 \times 10^{-5}Kg\,m^2$
I_{yy}	Principal Moment of Inertia around y	$1.436 \times 10^{-5}Kg\,m^2$
I_{zz}	Principal Moment of Inertia around z	$2.173 \times 10^{-5}Kg\,m^2$
C_T	Thrust coefficient	0.2025
C_D	Aerodynamic drag coefficient	0.11

Table 3.1: Phisical paramenters

3.2 Software

One of the greatest characteristics of the Crazyflie is that all the software produced by the creator company, Bitcraze, is fully open-source. This gives us complete control over the firmware, software that provides hardware low-level control, running on the quadrotor and on the Crazyradio. Bitcraze hosts all of its codebase on Github [38].

The Crazyflie Nano Quadcopter firmware, coded in C, was designed to optimize the flight of a single Crazyflie and it handles the scheduling of processes and control the flight calculations including a

huge amount of features: communication, a Kalman Filter, inner variables values read and write in real time, such as the PID gains, or the state estimation, or a full-state PID controller. All the calculations necessary to stabilize the drone is done by the Crazyflie firmware.

The *stabilizer* is composed of three parts. The first one is the *localization*, that gives the absolute position. Then the *command* part, which gives the positions to go. Finally, the *regulator* moves the drone to the position wanted [39].

The localization code contains both a complementary filter and an Extended Kalman Filter to do an estimation of the drone attitude. However only the latter one can estimate both orientation and position, as the first only estimate the orientation. Kalman filter is an algorithm that obtains reliable estimations on a system state. It works with a sequence of observed measurements, possibly from multiple sensors, and the dynamic model of the system. For the original Kalman filter, the hypotheses are that the system has to be linear and the noises have to be gaussian. But for non-linear ones such as the Crazyflie, it doesn't work properly. The Filter of the firmware is a variant of Kalman filtering that can more easily deals with this type of system. It does a first order Taylor expansion to obtain a linear approximation of the non-linear measurements and dynamic models. The command part is responsible for providing an absolute position to go.

Once the current and desired positions are known the regulation takes part to make the drone move appropriately. Indeed, moving too fast could leads to dangerous behaviors, as flips or crashes. On the opposite, a slow movement could lead to drifts and a low reactivity. A bad regulation will never stabilize the drone on the desired position, so the regulation is crucial for the drone control.

A PID controller is implemented, it calculates an error value from the difference between a desired set point and a measured variable at each control loop. The proportional, P term is the current error value. The further the desired value is, the bigger this term will be. The integral, I term is the sum of error values over time. It is proportional to both the magnitude of the error and the duration of the error. The derivative, D term is the slope of the error over time. The strongest the slop is, the bigger this term will be. The PID regulator of the Crazyflie

is in fact composed of four PID controllers, each of them controls a different aspect of the drone as is explained in the next paragraphs.

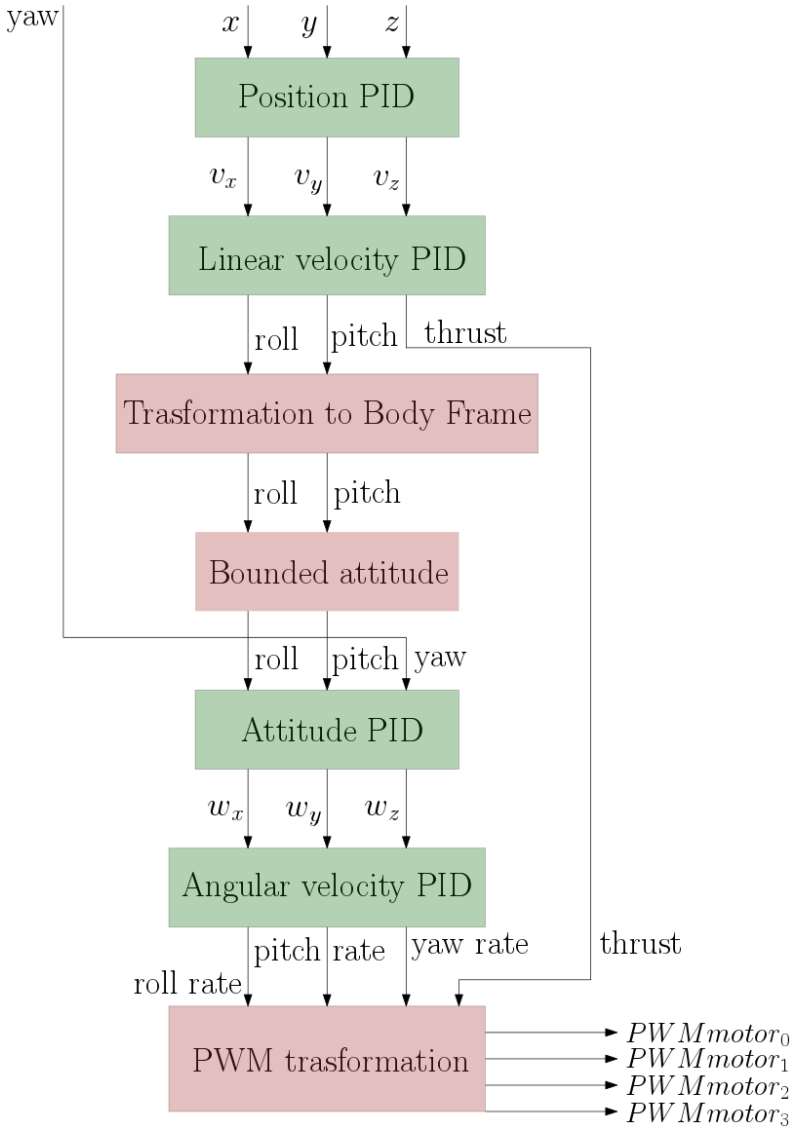
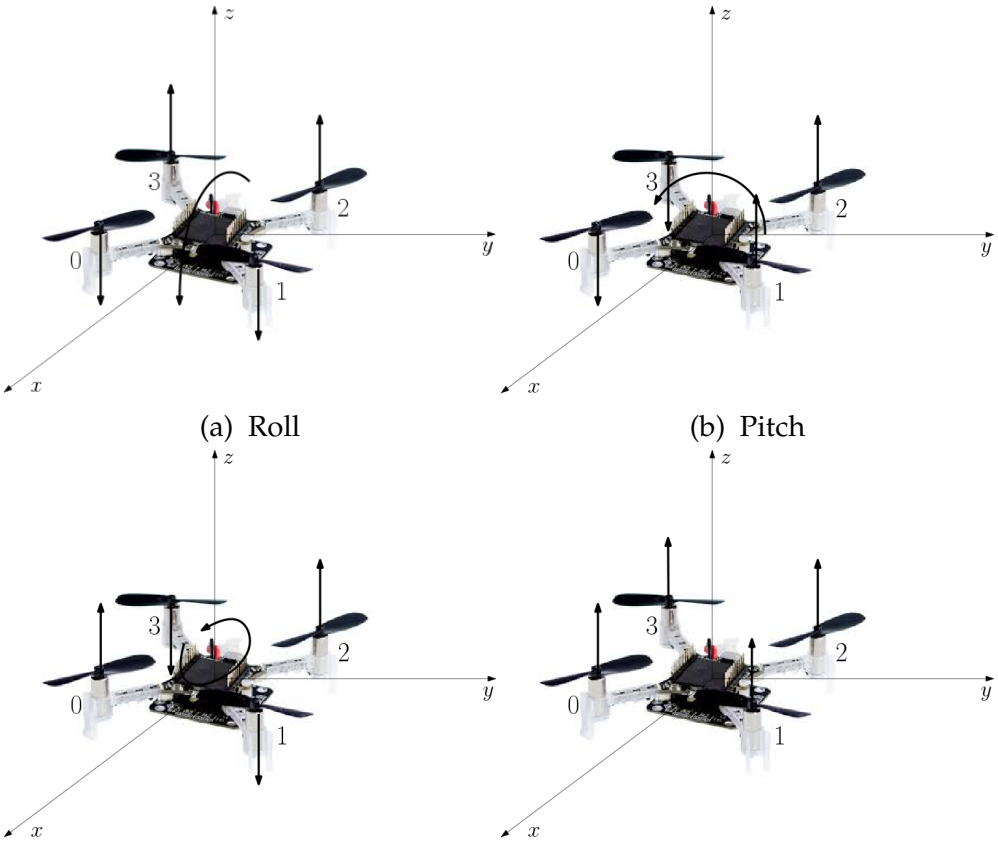


Figure 3.2: PID schema

The whole regulation works at lower rate than the stabilizer, which works at 1000 Hertz, as it would use unnecessary computational time otherwise. The rate is then of 500 Hertz. On the other hand, the desired orientation estimation takes a significant time to compute so its frequency is lower, 100 Hertz.

The algorithm takes the position and yaw desired as an input and transform it into the PWM that is directly applied on the four rotors of the Crazyflie, the process is shown in Figure 3.2. Firstly, having the desired position, the desired drone linear velocity is calculated with the *Position PID* in the inertial frame. Then, the roll, pitch and thrust are calculated in *Linear velocity PID*. This roll and pitch, and the yaw provided at the beginning, are compared to the drone roll, pitch, yaw with another PID, which gives the desired angular velocities. A final *Angula velocity PID* gives the roll, pitch, yaw rates that, with the thrust given in the linear velocity PID, are the input of the *PWM transformation*. Inside this box is described which value of PWM need each motor to achieve the point.



(a) Roll (b) Pitch

(c) Yaw (d) Thrust

Figure 3.3: How PWM is applied

The roll, pitch yaw rates and the thrust are distributed in the way is shown in equation 3.1, and will move the Crazyflie as shown in Figure

3.3. The thrust is applied to the four motors with the same value, due it is the responsible to maintain the drone in a certain altitude in the z axis. The roll rate positive value in applied to the motors 2 and 3, and the roll rate value but negative is applied in the motors 0 and 1, so it allows the drone to move in the x axis. The same happens with the pitch rate, in motors 1 and 2 the value is positive and in 0 and 3 is negative, being able to move in the y axis. The yaw rate, rotation around the z axis, is obtained applying the positive value to the motors 0 and 2 and the negative value on the other two motors.

$$\begin{aligned} PWM_{motor_0} &= Thrust - Roll\ rate + Pitch\ rate + Yaw\ rate \\ PWM_{motor_1} &= Thrust - Roll\ rate - Pitch\ rate - Yaw\ rate \\ PWM_{motor_2} &= Thrust + Roll\ rate - Pitch\ rate + Yaw\ rate \\ PWM_{motor_3} &= Thrust + Roll\ rate + Pitch\ rate - Yaw\ rate \end{aligned} \tag{3.1}$$

Also, Bitcraze distributes an open-source Python library that allows users to develop their own client applications [38]. The library provides simple calls to the USB radio that can be sent to the Crazyflie. It also allows the user to define callbacks for received data from the quadrotor.

3.3 Dynamic model

We propose the mathematical model of the Crazyflie 2.0 in this section. The dynamical model is mathematical description of nature movement. It is a key part when studying robots, because it describes the robot behavior according to its inputs. Thanks to these equations it is possible to define and predict the positions reached by the quadrotor investigating just the four motor speeds.

3.3.1 Preliminar notions

It is necessary to describe the reference coordinates we will use in the following equations. As it is described in Bitcraze documentation we will follow the convention set to the coordinate system [35] (Fig. 3.4). First of all, the 'Body Frame' which is non inertial and is fixed to each

quadrotor. The origin of the body frame matches with the quadcopter center of gravity and it is a ENU frame (east, north, up), meaning a positive altitude upwards. Secondly the 'World Frame', which is inertial and commune for all the Crazyflies.

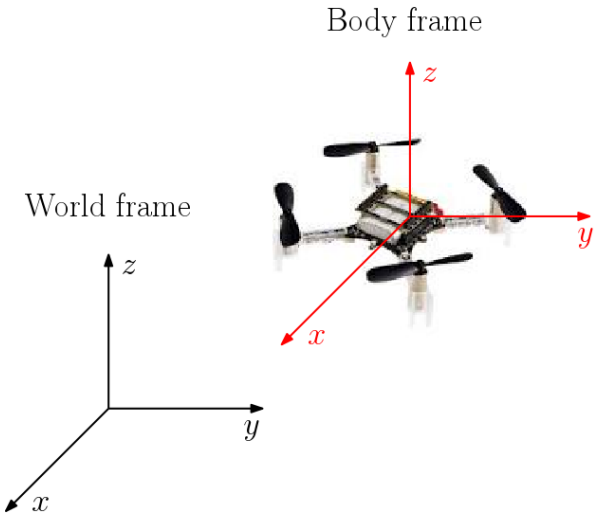


Figure 3.4: Coordinate systems

Six degrees of freedom are required in describing the space motion of a rigid body aircraft. Three of them are barycenter movements and the other three are movements around the barycenter; three translation withing the three axis and three rotation, roll, pitch and yaw.

As we expose in the last section, the velocity and position of the quadrotor can be controlled changing the values of motors speed. It happens because opposite propellers in the same diameter rotate in the same sense and, also, in the contrary sense of the propellers in the other diameter and depending on the relative speed between them the quadrotor performs different movements. Thrust when the four motor rotate at the same speed (Fig. 3.3d), pitching momentum (Fig. 3.3b) and rolling momentum (Fig. 3.3a) caused by the difference of four rotors speed, as explained before, and yawing momentum (Fig. 3.3c) caused by the unbalanced of the four rotors rotations [40].

In the following explanations we will use some variables in equations that are defined in the Table 3.2.

A super-index indicates in which frame is the vector expressed, be-

Vector	State	Description
p_{CG}	X	X position of the gravity center
	Y	Y position of the gravity center
	Z	Z position of the gravity center
Φ	ϕ	Roll angle
	θ	Pith angle
	ψ	Yaw angle
V_{CG}	u	X linear velocity of the gravity center
	v	Y linear velocity of the gravity center
	ω	X linear velocity of the gravity center
w	R	Roll angular velocity
	P	Pitch angular velocity
	Y	Yaw angular velocity
F_i	Force generated by propeller i	
T_i	Upward thrust force generated by propeller i	
M_i	Momentum generated by the propeller i	
h	Angular momentum around center of gravity	
P_i	Position of i motor in the body frame	
τ_i	Induced momentum by the i-th motor	

Table 3.2

ing, for example, p^o_{CG} the position of the gravity center expressed in the inertial frame and ω^b the angular velocity expressed in the body frame.

3.3.2 Dynamic Equation

The quadcopter dynamic equations proposed here take into account certain physical properties that are good approximations that simplify greatly the study and comprehension of this type of vehicles. Here are the hypothesis:

1. The quadcopter is a rigid body that cannot be deformed, thus it is possible to use the well-known dynamic equations of a rigid body.

2. The quadcopter is symmetrical in its geometry, mass and propulsion system.
3. The mass is constant, its derivative is 0.

Rotation Matrix

The mechanical classic laws of motion are valid in inertial systems and the robots dynamic equations are usually expressed in the body frame. So, to be able to operate with these equation we need to switch frames, on other words, it is necessary to define a rigid transformation matrix.

The rotation matrix R_o^b, from the inertial frame *i* to the body frame *b*, correspond to a rotation in each axis as we can see in the equation.

$$R_o^b = R_z R_y R_x \tag{3.2}$$

Ones this matrices are calculated, the resulting transformation matrix is

$$R_o^b = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\theta\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\theta\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\theta\cos\psi & \cos\phi\cos\theta \end{bmatrix} \tag{3.3}$$

If we need the rotation matrix from the body frame to the inertial frame we just need to do the inversion.

$$(R_o^b)^{-1} = (R_o^b)^T = R_b^o \tag{3.4}$$

Forces equations

According to Newton's Second Law, the expression for the sum of forces in the inertial frame is [41]

$$\sum F^o = m\dot{V}_{CG}^o \tag{3.5}$$

being the velocity derivative, using the Coriolis equation, the following expression.

$$\dot{V}_{CG}^o = \dot{V}_{CG}^b + \omega \times V_{CG} \tag{3.6}$$

The force is defined as

$$\sum F = m(\dot{V}_{CG}^{b} + \omega \times V_{CG}) \tag{3.7}$$

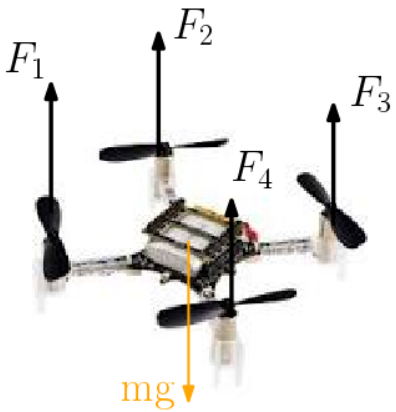


Figure 3.5

Each propeller of the quadcopter creates an aerodynamic force as shown in Figure 3.5 that acts upward in the body frame

In a situation where the quadcopter is parallel to the ground, meaning its roll and pitch angles are zero, the aerodynamic forces created by the propellers will search to counteract the effect of the gravity and then make the quadcopter move upwards, downwards or stay in a hover position.This qualitative analysis can be translated in the following equation.

$$\begin{bmatrix} 0 \\ 0 \\ F_z \end{bmatrix} - R_o^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = m \left(\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} R \\ P \\ Y \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \right) \tag{3.8}$$

Isolating the $\dot{V}_{CG}$ we get the equation of the linear velocity of the quadcopter center of gravity in its body frame.

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ F_z/m \end{bmatrix} - R_o^b \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} - \begin{bmatrix} R \\ P \\ Y \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{3.9}$$

For get the position of the quadcopter center of gravity in the inertial frame we apply the following expression.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = R_b^o \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{3.10}$$

To calculate the expression 3.9 and 3.10 we need to specify how is the aerodynamic force generated by each propeller.

$$F_i^b = \begin{bmatrix} 0 \\ 0 \\ T_i \end{bmatrix} \tag{3.11}$$

where the trust generated by a propeller is a function of the angular speed.

$$T_i = C_T \omega^2 \tag{3.12}$$

As we have four propellers the equation is the following.

$$\sum F_i^b = \begin{bmatrix} 0 \\ 0 \\ C_T(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \end{bmatrix} \tag{3.13}$$

Momentum equations

Following the theorem of angular momentum [41].

$$\sum M^o = \dot{h}^o \tag{3.14}$$

being the angular momentum around the center of gravity expression, using the Coriolis equation, the following equation.

$$\dot{h}^o = \dot{h}^b + \omega \times hh = J\omega \tag{3.15}$$

So, if we substitute on equation 3.15 in equation 3.14 we obtain the expression 3.16.

$$\sum M^b = J^b \dot{\omega} + \omega \times J\omega \tag{3.16}$$

where the J defines the quadrotor inertia matrix. As the Crazyflie is symmetrical around all its axes, the inertia matrix has all crossed terms equal to zero.

$$J = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{3.17}$$

Isolating the $\dot{\omega}$ we get the equation of angular velocity.

$$\begin{bmatrix} \dot{R} \\ \dot{P} \\ \dot{Y} \end{bmatrix} = J^{-1} \left(\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} - \begin{bmatrix} R \\ P \\ Y \end{bmatrix} - \begin{bmatrix} R \\ P \\ Y \end{bmatrix} \times J \begin{bmatrix} R \\ P \\ Y \end{bmatrix} \right) \tag{3.18}$$

Regarding the relation between ω and the $\dot{\Phi}$ we get the attitude equation.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} R \\ P \\ Y \end{bmatrix} \quad for\ \theta \neq \frac{\pi}{2} \tag{3.19}$$

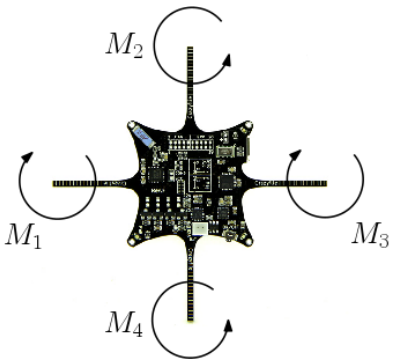


Figure 3.6

The momentum created by the propellers is

$$M^b = \sum P_i^b \times F_i^b + \sum \tau_i \tag{3.20}$$

If d denotes the distance from the center of gravity to the center of

each motor, the position of each motor is

$$P_1 = \begin{bmatrix} d/\sqrt{2} \\ -d/\sqrt{2} \\ 0 \end{bmatrix}, \quad P_2 = \begin{bmatrix} -d/\sqrt{2} \\ -d/\sqrt{2} \\ 0 \end{bmatrix}, \quad P_3 = \begin{bmatrix} -d/\sqrt{2} \\ d/\sqrt{2} \\ 0 \end{bmatrix}, \quad P_4 = \begin{bmatrix} d/\sqrt{2} \\ d/\sqrt{2} \\ 0 \end{bmatrix}, \tag{3.21}$$

Calculating the induced moments that only act in the z axis as shown in the Figure 3.6, we get the following τ expression.

$$\sum \tau = \begin{bmatrix} 0 \\ 0 \\ C_D(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2) \end{bmatrix} \tag{3.22}$$

Finally the total momentum is

$$\sum M_i^b = \begin{bmatrix} dC_D/\sqrt{2}(-\omega_1^2 - \omega_2^2 + \omega_3^2 + \omega_4^2) \\ dC_D/\sqrt{2}(-\omega_1^2 + \omega_2^2 + \omega_3^2 - \omega_4^2) \\ C_D/(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2) \end{bmatrix} \tag{3.23}$$

Chapter 4

Control algorithm

In this section we are going to elaborate the algorithm used to achieve the desired swarm behavior. First of all we are going to define all the parameters needed to develop this algorithm, some of them have already appeared in Chapter 2.

Symbol	Description
x_d	Leader desired point
$G(x)$	Collision avoidance graph
$\mathscr{E}(x)$	Collision avoidance graph edges
$G_o(x)$	Connectivity maintenance graph
$\mathscr{E}_o(x)$	Connectivity maintenance graph edges
d_{conn}	Sense radio
d_{coll}	Colliding distance

Table 4.1

As we said in Chapter 2 the Leader-Follower behavior we want to achieve is based on guarantee connectivity maintenance between robots in the initial graph and collision avoidance during the hole performance.

The control must be decentralized so the decision-making process must be distributed across the population of Crazyflies, each one control himself independently. The leader is just the agent that knows the desired point, but he does not command any order to the rest of

agents. Due to that, the idea is to develop a algorithm which will be run in every Crazyflie independently, and the only information they will share is their position with the agents that are in their sense radius. The code will be written in Python and the communication will be done with ROS, which will be shown in Chapter 5.

To guarantee connectivity maintenance we define the function

$$\beta_{conn}(\eta) : R_{\geq 0} \rightarrow \left[0, \overline{\beta}_{conn}\right]$$

$$\beta_{conn}(\eta) = \begin{cases} 0 & \eta < 0 \\ \vartheta_{conn}(\eta) & 0 \leq \eta < d_{conn}^2 \\ \overline{\beta}_{conn} & d_{conn}^2 < \eta \end{cases} \tag{4.1}$$

where $\overline{\beta}_{conn}$ is a positive constant and $\vartheta_{conn}(\eta)$ is a polynomial. As defined before, d_{conn} is the sense radius of the quadcopters.

This function depends on a η, which we define as

$$\eta = d_{conn}^2 - \|p_1 - p_2\|^2 \tag{4.2}$$

where p_i, as defined in Table 3.2, is the position of the agent i.

The polynomial $\vartheta_{conn}(\eta)$ guarantees that $\beta_{conn}(\eta)$ is twice continuous differentiable for all m in the set $\mathscr{M}_o$ and it must fulfill that $\vartheta_{conn}(0) = 0$ and $\vartheta_{conn}(d_{conn}^2) = \overline{\beta}_{conn}$.

As we can read in the function, when the distance between drones is higher than the sense radius, the robots in the edge are disconnected, η is negative and $\beta_{conn}(\eta)$ value is 0, so $\beta_{conn}(\eta)$ has no effect on the control. In the case agents distance is grater than 0, $\beta_{conn}(\eta)$ is defined by $\vartheta_{conn}(\eta)$. On the other hand, if the distance is less than 0, $\beta_{conn}(\eta)$ becomes a constant because is not $\beta_{conn}(\eta)$ responsibility to deal with collisions.

The further the agents are, the bigger force we want them to apply to remain inside the sense radius. Also, we want smooth robot translations, avoiding quick movements that could make the system unstable or oscillatory. Thats why we search for a function that smoothly goes from 0 to $\overline{\beta}_{conn}$, so we use a high order polynomial to achieve so

$$\vartheta_{conn}(\eta) = a\eta^5 + b\eta^4 + c\eta^3 \qquad a, b, c = constant \tag{4.3}$$

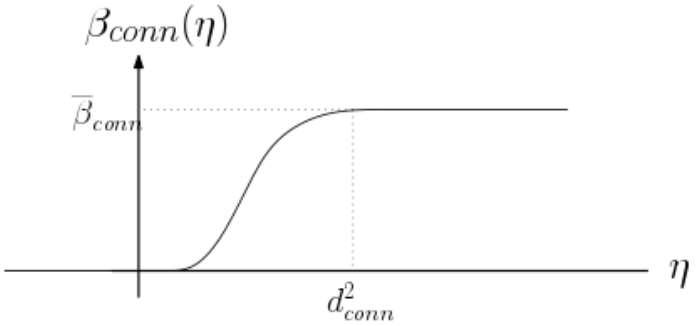


Figure 4.1

We can see the final shape of $\beta_{conn}(\eta)$ function in the Figur 4.1.

Furthermore, to implement collision avoidance we apply a similar algorithm as in the connectivity maintenance. First of all we define the function

$\beta_{coll}(\iota) : R_{\geq 0} \rightarrow [0, \overline{\beta}_{coll}]$

$$\beta_{coll}(\iota) = \begin{cases} 0 & \iota < 0 \\ \vartheta_{coll}(\iota) & 0 \leq \iota < d_{coll} \\ \overline{\beta}_{coll} & d_{coll} < \iota \end{cases} \tag{4.4}$$

where $\overline{\beta}_{coll}$ is a positive constant and $\vartheta_{coll}(\iota)$ is a polynomial. Being d_{coll} the following expression

$$d_{coll} = d^2_{conn} - (r_{m_1} + r_{m_2})^2 \tag{4.5}$$

This function depends on a ι, which we define as

$$\iota = \|p_1 - p_2\|^2 - (r_{m_1} + r_{m_2})^2 \tag{4.6}$$

The $\vartheta_{coll}(\iota)$ guarantees that $\beta_{coll}(\iota)$ is twice continuous differentiable for all m in the set $\mathscr{M}$. $\vartheta_{coll}(\iota)$ must fulfill that $\vartheta_{coll}(0) = 0$ and $\vartheta_{coll}(d_{coll}) = \overline{\beta}_{coll}$.

In this case, ι is negative and $\beta_{coll}(\iota)$ becomes 0 if the distance between drones is smaller that the minimum distance for not colliding, that is actually the sum of both quadrotor radius. When agents distance is positive but smaller than sense radius, $\beta_{coll}(\iota)$ is defined by

$\vartheta_{coll}(\iota)$. If the distance is grater than the sense radius $\beta_{coll}(\iota)$ becomes a constant because is not $\beta_{coll}(\iota)$ responsibility to deal with quadrotor disconnection.

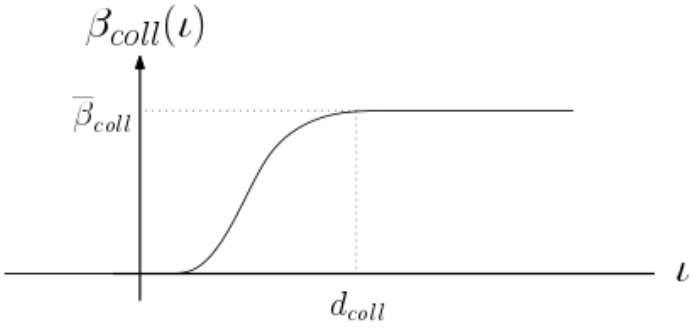


Figure 4.2

The closer robots are, the bigger force we want to apply on them to avoid the collision. As said before, we are looking for smooth robot translations, avoiding quick movements that could make the system unstable or oscillatory. That is why we search for a function that smoothly goes from 0 to $\overline{\beta}_{coll}$, so we use a high order polynomial.

$$\vartheta_{coll}(\iota) = a'\iota^5 + b'\iota^4 + c'\iota^3 \qquad a', b', c' = constant \tag{4.7}$$

We can see the final shape of $\beta_{coll}(\iota)$ function in the Figure 4.2.

Finally, we define

$$\beta'_{conn} = \frac{\partial}{\partial \eta}\left(\frac{1}{\beta_{conn}(\eta)}\right) \ \forall l \in \mathscr{M}_o \tag{4.8}$$

$$\beta'_{coll} = \frac{\partial}{\partial \iota}\left(\frac{1}{\beta_{coll}(\iota)}\right) \ \forall m \in \mathscr{M} \tag{4.9}$$

which diverge to infinity in a connectivity break or in a collision of robots l_1, l_2 and m_1, m_2 , respectively.

At this moment, the decentralized control expression has the following expression.

$$u = \sum_{l \in \mathscr{M}_o} \alpha_{i,l}\ \beta'_{conn} \frac{\partial \eta}{\partial x_{l_1}} + \sum_{m \in \mathscr{M}} \alpha_{i,m}\ \beta'_{coll} \frac{\partial \iota}{\partial x_{m_1}} \tag{4.10}$$

where $\alpha_{i,l}$ and $\alpha_{i,m}$ take the values -1, 1 or 0 in these three different cases

$$\alpha_{i,l} = \begin{cases} -1 & i = l_1 \\ 1 & i = l_2 \\ 0 & i \neq l_1, l_2 \end{cases} \tag{4.11}$$

$$\alpha_{i,m} = \begin{cases} -1 & i = m_1 \\ 1 & i = m_2 \\ 0 & i \neq m_1, m_2 \end{cases} \tag{4.12}$$

The terms to implement connectivity maintenance and collision avoidance are not enough to perform a stable quadrotor behavior, so we add a dissipative term, ξ. This term is, basically, a PI in the velocity of each quadrotor in order to have a smoother and non oscillating result.

$$\xi = k_{p_i} e_{v_i} + k_{v_i} \int_{t=0}^{t} v_i \, dt \tag{4.13}$$

where $e_{v_i} = v_i - v_{desired_i}$. For the followers, the desired velocity desired is obtained with the following expression.

$$v_{desired_i} = k_i \left(\sum_{l \in \mathscr{M}_o} \alpha_{i,l} \, \beta'_{conn} \frac{\partial \eta}{\partial x_{l_1}} + \sum_{m \in \mathscr{M}} \alpha_{i,m} \, \beta'_{coll} \frac{\partial \iota}{\partial x_{m_1}} \right) \tag{4.14}$$

However, for the leader the velocity desired needs a PI in the position.

$$v_{desired_1} = k_p e_p - k_i \int_{t=0}^{t} x_1 \, dt + k_1 \left(\sum_{l \in \mathscr{M}_o} \alpha_{i,l} \, \beta'_{conn} \frac{\partial \eta}{\partial x_{l_1}} + \sum_{m \in \mathscr{M}} \alpha_{i,m} \, \beta'_{coll} \frac{\partial \iota}{\partial x_{m_1}} \right) \tag{4.15}$$

Finally, to guarantee the leader tracking the desired position we need to add a new term, $\tilde{e}$.

$$\tilde{e} = e_{pos} + k_{pos} \int_{t=0}^{t} x_1 \, dt \tag{4.16}$$

$$e_{pos} = x_1 - x_{desired} \tag{4.17}$$

So, finally the control algorithm is define in the following exporession

$$u = \sum_{l \in \mathscr{M}_o} \alpha_{i,l}\, \beta'_{conn} \frac{\partial \eta}{\partial x_{l_1}} + \sum_{m \in \mathscr{M}} \alpha_{i,m}\, \beta'_{coll} \frac{\partial \iota}{\partial x_{m_1}} - \xi + \lambda \tilde{e} \tag{4.18}$$

where $\lambda = 1$ if $i = 1$ and $\lambda = 0$ if $i = 2, 3, \ldots$.

Chapter 5

Simulation

In this chapter we describe how we simulate the multi-agent control algorithm. The control simulation of a robot is vital in research. With it we can emulate the behavior of the real quadrotors in the computer and we can check how they will behave in real life, being able to adjust parameters and improve the performance to achieve the final objective.

First of all we need to describe the simulation environment proposed.

5.1 Simulator

I worked with a C++ simulator based on ROS communication. There are several computer platforms done to develop a quadrotor simulation, but non of them were explicitly design for Crazyflie 2.0. Therefore, I decided to use directly C++ Files and visualize the performance with R-VIZ which is a 3D visualizer for displaying live representations and state information from ROS on a virtual model of the robot.

C++ files, on other words, the proper simulator, emulate the inside of the Crazyflie. As we define in previous chapters, it consists of four PID that process the input data before sending it to the motors. As it

has to behave as the Crazyflie does, we need to implement its dynamical equations, explained in the second Chapter, to obtain a realistic response of the quadrotors.

In our case, we simulate a swarm of Crazyflies, so we are going to run the simulator code as many time as quadrotors are in the swarm.

5.1.1 PIDs implemented

Described on Chapter 3, the Crazyflie has four cascading PID previous applying any force to the rotors. They have a structure showed in Figure 5.1.

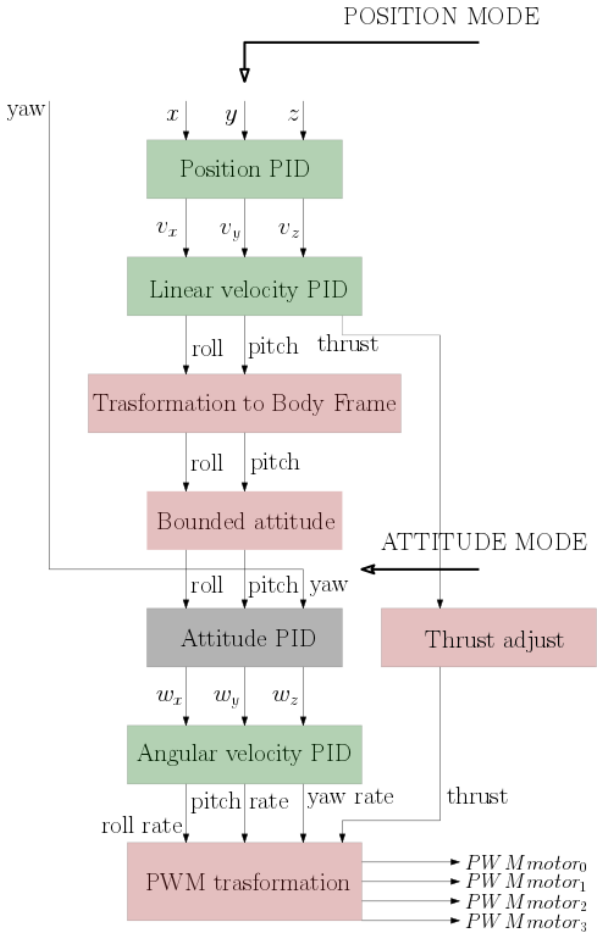


Figure 5.1

Regarding to this figure we can understand how PIDs are implemented in the simulator. Crazyflie can be controlled sending the position we want to achieve or sending the desired attitude, with the *POSTION/ATTITUDE MODE*. In the first option data pass through every single PID and, in the second option, data go directly to the attitude desired', skipping the first two PIDs that are no needed. To use the *POSTION MODE* we need to provide the position values in the three axis and the yaw angle we want to keep. In case of sending the attitude desired we have to define roll, pitch, yaw rate and the thrust desired. We can see, in the schema (Fig. 5.1), how the info go into the process depending on the mode chosen.

Some further clarification are needed in the PIDs implementation. First of all, inside the *Bounded attitude* box we constrain the attitude values due to Crazyflie hardware. It is needed because, even thought, we want a big attitude degree it can no be achieved physically by the drone, so we use the maximum possible degree instead. Besides, the *Transformation to Body Frame* box is needed because when we provide the desired position it is expressed in the world frame, and the Crazyflie works with body frame. Finally, inside the *Thrust adjust* we replicate the alteration firmware make to the thrust value. It consist of adding a value that approximatively correspond to $mass \times gravity$ to ensure a really stable hover. On the other hand, in *PWM transformation* we implement the distribution of PWM in the different rotors as we explained on Chapter 3 (Fig. 3.3).

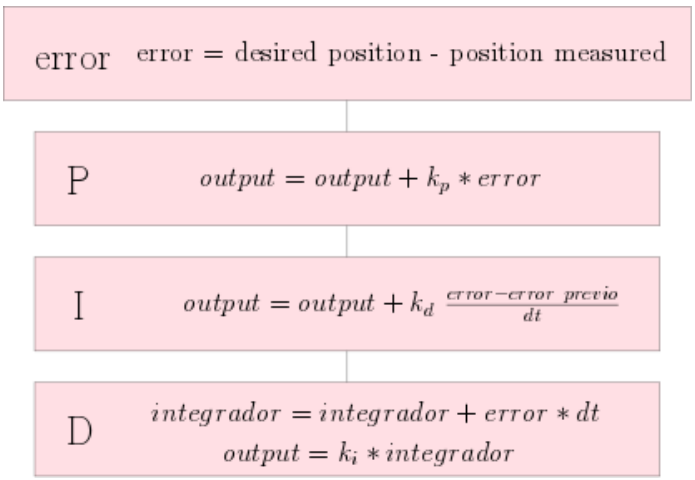


Figure 5.2

Inside every PID we have the structure that is explained in the flowchart of Figure 5.2. All the gains used are the ones used in the Crazyflie firmware.

5.1.2 Dynamic equation implemented

The dynamic equation explained in Chapter 3 must be implemented in the C++ code in order to provide the response of the real Crazyflie.

First of all, the output after PIDs inside Crazyflie is a PWM in each motor so, we have to convert this PWM to the input required by dynamic equations, which are forces and momentums. To do that, firstly we convert the PWM in revolution per minute with the expression 5.1.

$$RPM = 0.2685 \times PWM + 4070.3 \tag{5.1}$$

Knowing the angular velocity of each rotor we can calculate forces and momentums with equations 3.13 and 3.23.

In the *Dynamical Model* node shown in Figure 5.3 are implemented all the equations in Chapter 3 and also is fixed the ground constraint.

5.2 ROS

Now that we have the Control file and the Simulator we can run the swarm simulation. As explained before, the control must be decentralized, so every Crazyflie will run the control code independently, what means that we will run the control and the simulator file as many times as Crazyflie we want to test. Using a launch file we are able to run every code in a coordinate way.

It is not difficult to imagine the amount of information that has to be sent from one place to another. Not only to enable the communication between control and simulator, but also to communicate between Crazyflies when needed, allowing getting feedback information and

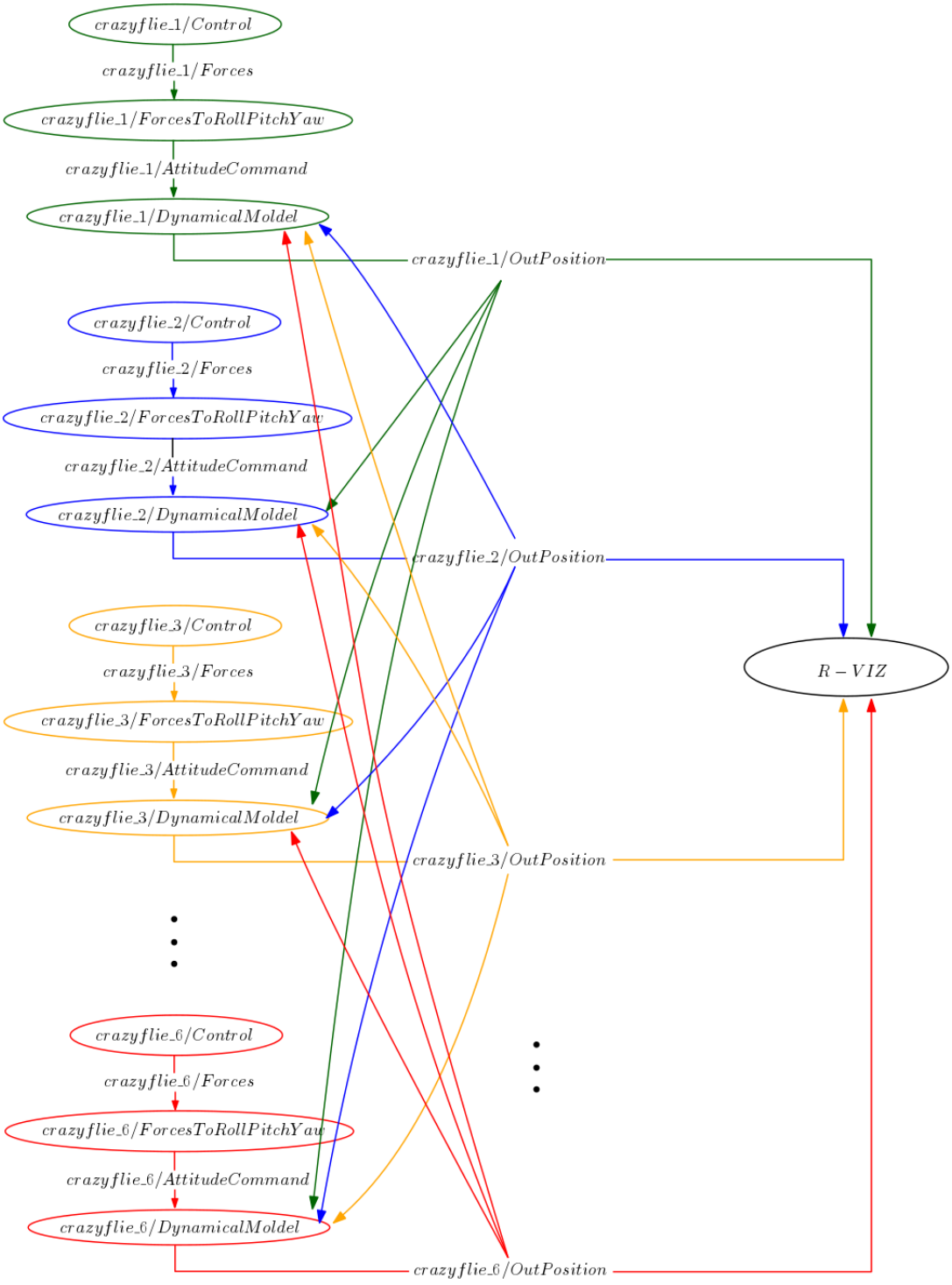

crazyflie_1/Control
crazyflie_1/Forces
crazyflie_1/ForcesToRollPitchYaw
crazyflie_1/AttitudeCommand
crazyflie_1/DynamicalMoldel
crazyflie_1/OutPosition
crazyflie_2/Control
crazyflie_2/Forces
crazyflie_2/ForcesToRollPitchYaw
crazyflie_2/AttitudeCommand
crazyflie_2/DynamicalMoldel
crazyflie_2/OutPosition
crazyflie_3/Control
crazyflie_3/Forces
crazyflie_3/ForcesToRollPitchYaw
crazyflie_3/AttitudeCommand
crazyflie_3/DynamicalMoldel
crazyflie_3/OutPosition
R − VIZ
crazyflie_6/Control
crazyflie_6/Forces
crazyflie_6/ForcesToRollPitchYaw
crazyflie_6/AttitudeCommand
crazyflie_6/DynamicalMoldel
crazyflie_6/OutPosition

Figure 5.3

sending new states to a software capable of doing the visualization. All this flux of information requires a big organization of the data that is managed by ROS.

We have chosen ROS because its information handling is simple and efficient and, also, because there are Crazyflie libraries explicitly done to work with ROS, so it make the communication faster and more optimized in comparison with other communication methods.

ROS manages the information simplifying the problem to nodes and topics. Nodes are the agents that need to be communicated, and topics are the information that need to be sent from one node to another. A node can be a subscriber and/or publisher of topics, so it can receive data from the topic or publish data on the topic. Each topic need a data type that must be described in the code, so the nodes know what can the publish or get from them.

In our experiment we have the schema showed in the Figure 5.3. Nodes are represented by circles, arrows indicate the data flux between nodes and topics and, finally, topics are located in the middle of this arrows. Each node and topic has its own name to *call* it. Further more, nodes and topics of a specific node, for example *crazyflie_1/Control* and *crazyflie_1/Forces*, are preceded with the name of the Crazyflie they are refered to, this make it easier to find the node or topic wanted.

For each Crazyflie, in this case six, we have a *Control* node, a *ForcesToRollPitchYaw* node and a *DynamicalMoldel* node. The *ForcesToRollPitchYaw* node translate the force output from the control node to an attitude, which is the actual input of the simulator node. Inside *DynamicalMoldel*, the simulator node, we have the PIDs and the dynamical equations explained in Section 5.1.

Besides, for each Crazyflie we have four topics. In topic *crazyflie_i/Forces* the control node publish the force desired, node *crazyflie_i/ForcesToRollPitchYaw* is subscribed to this topic, so it can get the data and translate it to attitude. On the other hand, *crazyflie_i/ForcesToRollPitchYaw* node publish the attitude information in topic *crazyflie_i/AttitudeCommand*, where the simulator node is subscribed.

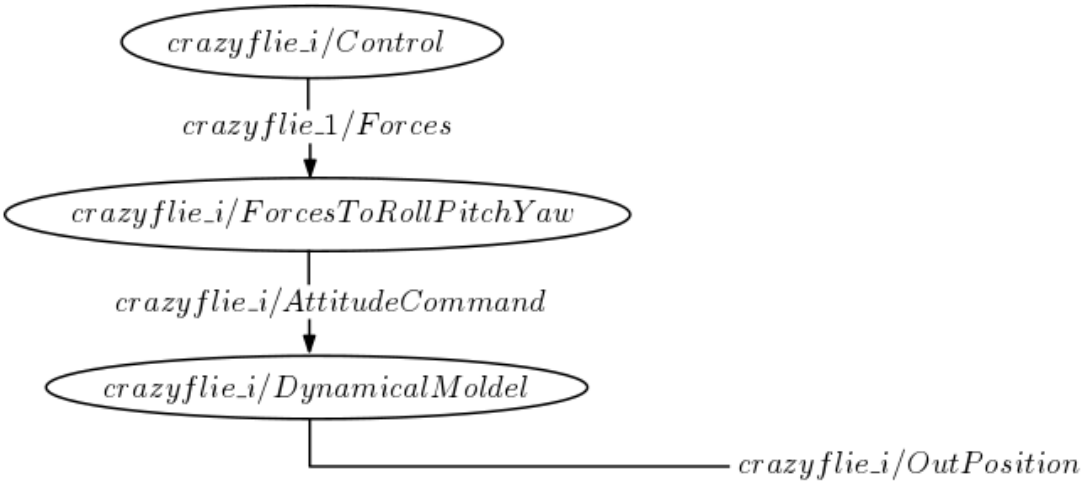


Figure 5.4

To allow the communication between Crazyflies we create the topic *crazyflie_i/OutPosition* in which they publish their position and the rest of Crazyflies can access to it. For instance, they are subscribed to the position of every single Crazyflie, however, they do not use this information unless they are inside their sense radius.

Finally the *R-VIZ* is the visualizer node. It is also subscribed to *crazyflie_i/OutPosition* to know the position of each Crazyflie at every moment and being able to print it.

5.3 Experiment

In our experiment we will use a swarm of six Crazyflies positioned in the way is shown in Figure 5.5.

We are going to do two experiments with this initial pose, each one with a different connectivity maintenance graph and same collision avoidance graph as it can not be changed because is every Crazyflie linked with the rest of them.

So, for the first one, *Experiment A*, we define the collision avoidance graph (Fig. 5.6a), and the connectivity maintenance graph (Fig. 5.6b).

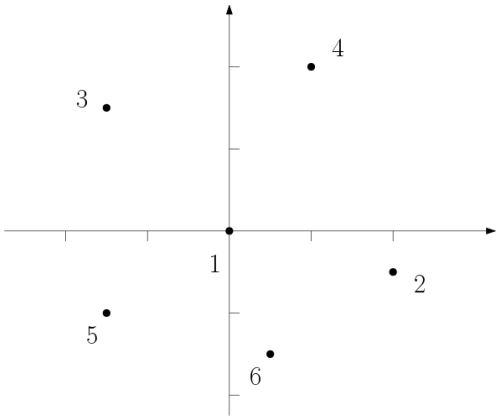
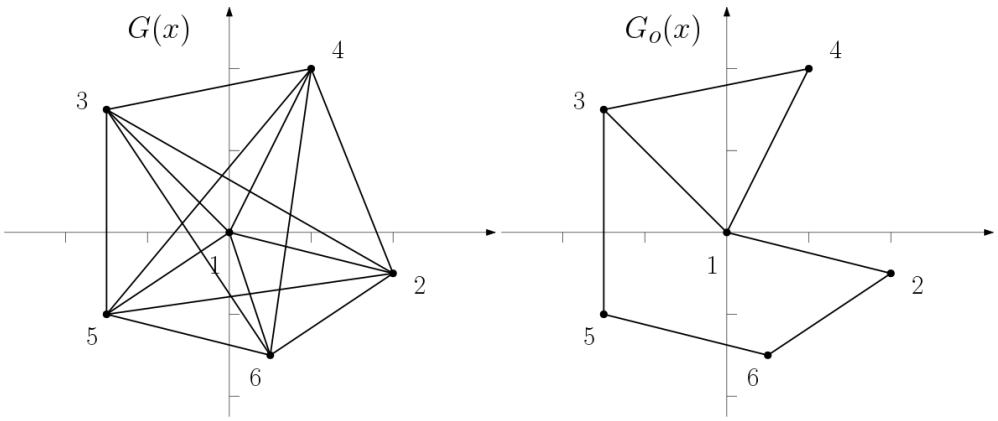


Figure 5.5: Initial pose



(a) Collision avoidance graph (b) Connectivity maintenance graph

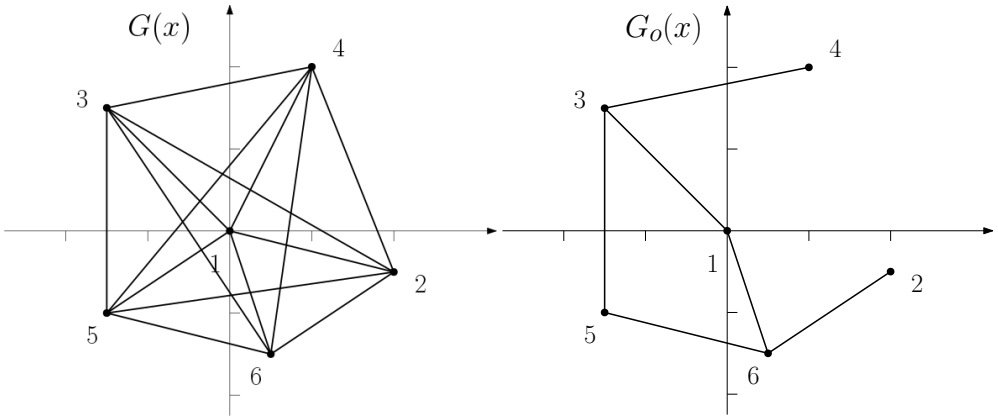
Figure 5.6: Experiment A

For *Experiment B* we define the graphs in Figure 5.7.

We will also test two different trajectories to see how it behaves in several situations. Both trajectories are defined by seven points:

I (0, 0, 0.2), (0, 0, 2), (0, 0, 5), (4, 5, 3), (-2, 4, 2), (3, -2, 3), (1, -1, 2)

II (0, 0, 0.2), (0, 0, 2), (-3, -3, 2), (-1, 4, 4), (4, -2, 1), (3, 3, 3), (1, 1, 1)

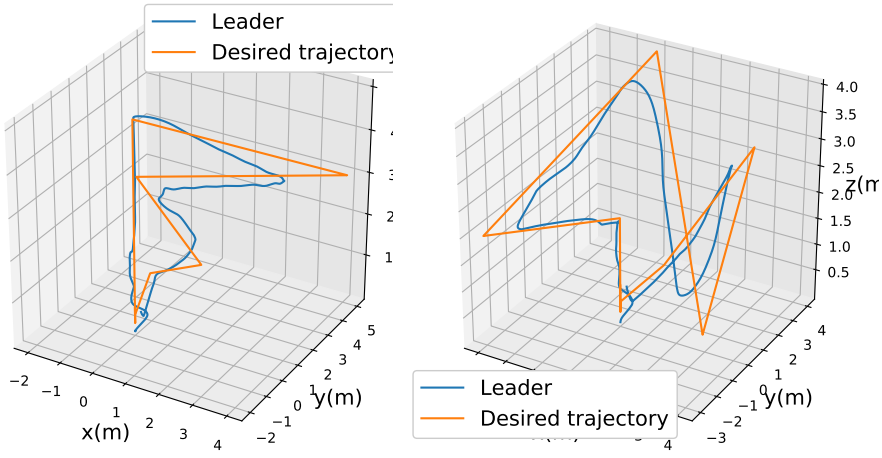


(a) Collision avoidance graph (b) Connectivity maintenance graph

Figure 5.7: Experiment B

5.4 Results

In this Chapter we will show the simulation behavior of the swarm in the experiments previously defined.



(a) Traj.I (b) Traj.II

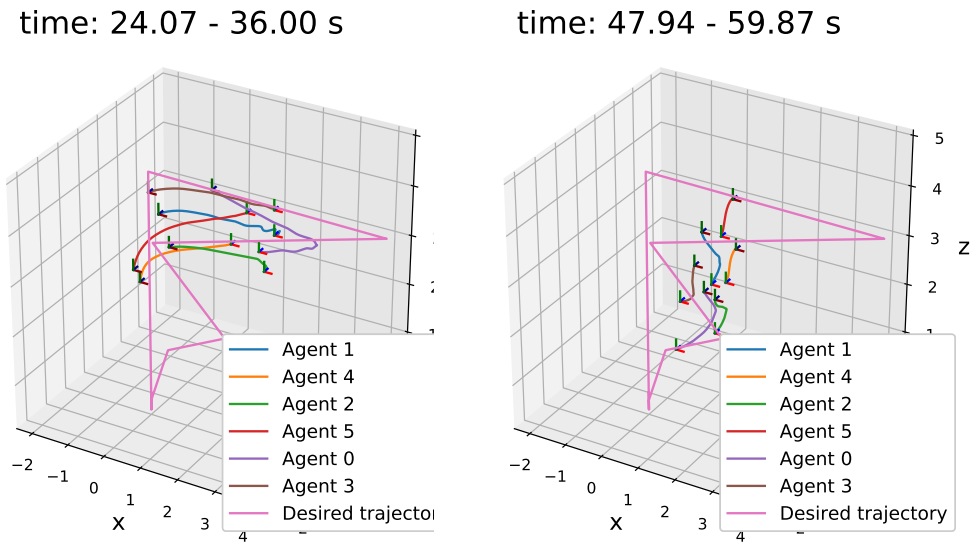
Figure 5.8: Lider 3D route in *Experiment A*

We are going no analyze the leader tracking position and the follower routes during the performance, taking in account the distances between agents in the swarm. Further more, we are going to ana-

lyze the output control force paying especial attention in the collision avoidance and connectivity maintenance terms impact.

For *Experiment A* the leader route of each trajectory is shown in the Figure 5.8. As we can appreciate, trajectory I ans II are tracked smoothly and without oscillation. However, the desired trajectory is not perfectly followed because the leader has the effect of the rest agents in the swarm. It means that the leader has also to remain the connectivity and avoid the collisions with other Crazyflies so he is not able to to the perfect route. This behavior can be changed tunning the parameters on term responsible for achieve the desired trajectory, but we have decided that is better to maintain the same control in every agent so it is more generic although we might have this tracking error.

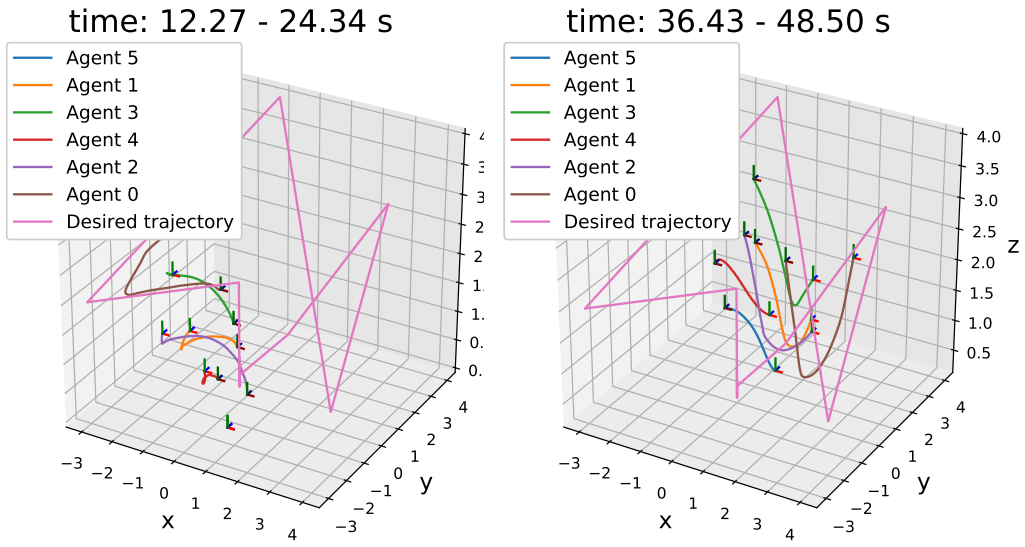
To bring an idea of how the system evolves in time we have printed the swarm behavior in some time spaces (Fig. 5.9 and 5.10).



(a) (b)

Figure 5.9: Swarm behavior in *Experiment A* Traj. I

In order to clarify agents conduct in the experiments, we need to analyze the distances between robots in every moment. Distances allow us to check if the swarm behavior is working properly, complying with the connectivity maintenance and the collision avoidance. In Figure 5.11 and 5.12 we plot the distances between every Crazyflie. The



(a) (b)

Figure 5.10: Swarm behavior in *Experiment A* Traj. II

dashed line at the button indicates minimum distance between agents. In fact, maximum colliding distance is higher than 4 meters, reason why it does not appear in the graph and is never exceeded.

As we can see, distances between Crazyflies are within the limits in every moment.

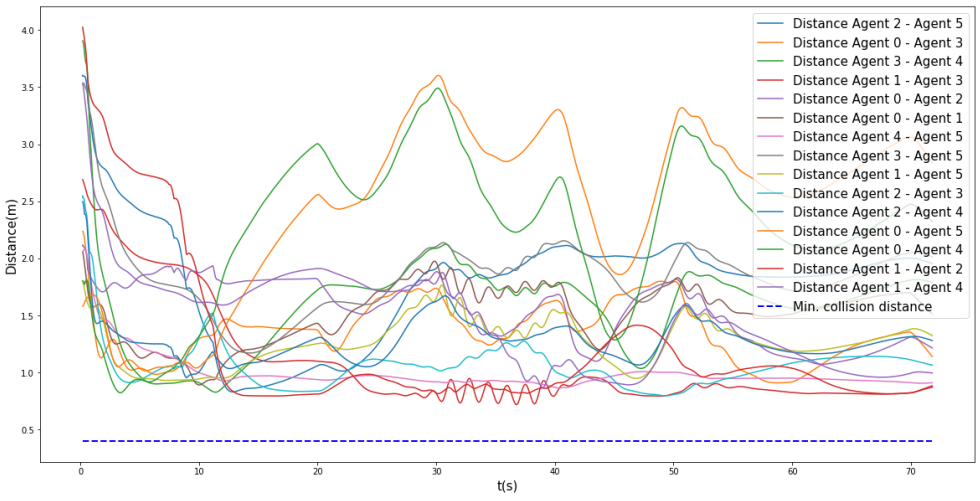


Figure 5.11: Swarm distances in *Experiment A* Traj. I

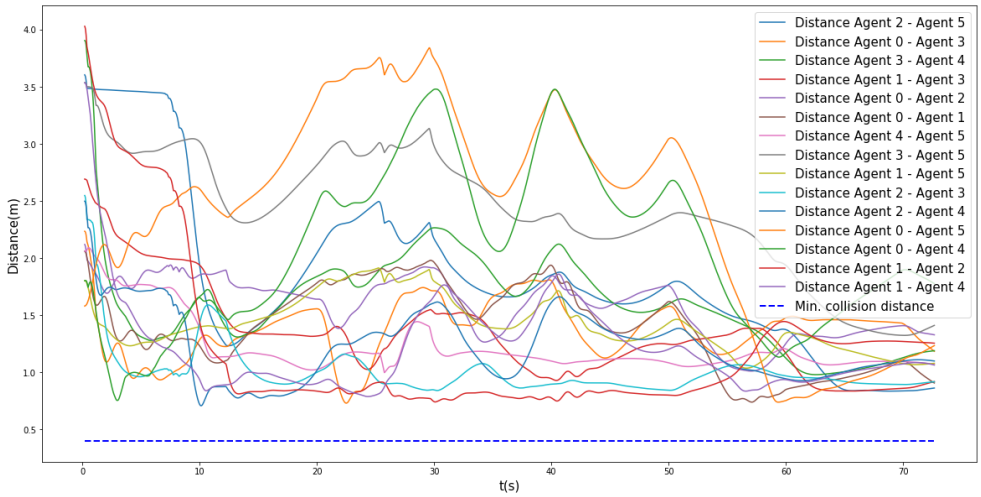


Figure 5.12: Swarm distances in *Experiment A*Traj. II

Afterwards, we need to analyze how is the control output signal given to every Crazyflie to achieve the goals. Without loose of generalization, we pick two random Crazyflies, 0 and 3, to expose the signals involved. For agent 1 we print the collision and connectivity terms (Fig. 5.13) and the output force (Fig. 5.14).

In agent 1, the collision avoidance is almost constant in the three axis when following trajectory I. This means that the agent have not had any problematic approximation with another agent in the performance. On the other hand, we can see that the connectivity maintenance term clearly changes its value, it means that exit some situations in which the connected agents are trying to move away and the term is trying to counteract the effect. Regarding force graph, even though

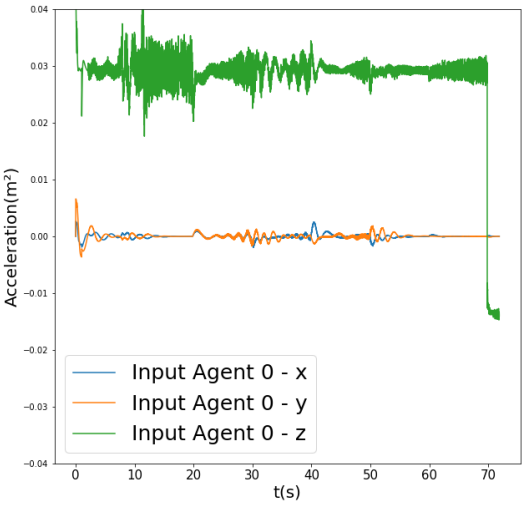


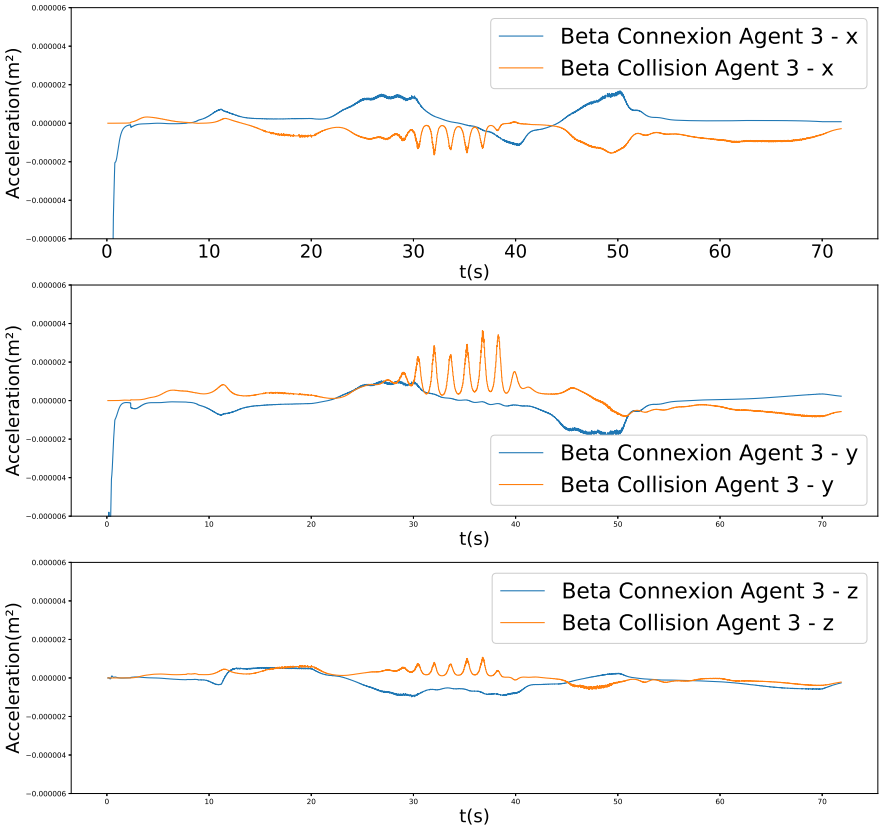
Figure 5.14: Force in *Exp. A* Traj. I for agent 1.

Figure 5.13: Coll. and conn. term in *Exp. A* Traj. I for agent 1.

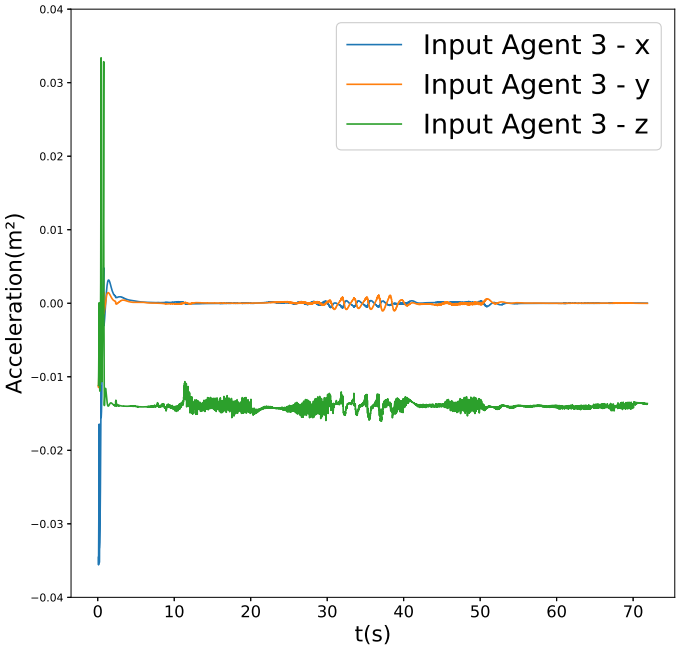
its value is fairly constant in the three axis, it has little oscillation due to connectivity term.

Analyzing Crazyflie number 3 we have Figure 5.15a for collision and connectivity terms and Figure 5.15b for the output force. In this agent, we find much more stable connectivity term, and some oscillations in collision term. The fast changes in values indicate that there is a problematic moment that is solved in a rapid and efficient way, being able to maintain a stable control force even thought they are trying to resolve the problem.

Force value in axis z is some units higher and, in average, less stable than its values in axis x and y. That is due to in axis z the control has to counter the gravity effect and drone's dynamics are much more complex than in other axis.



(a) Coll. and conn. term in *Exp. A* Traj. I for agent 3.

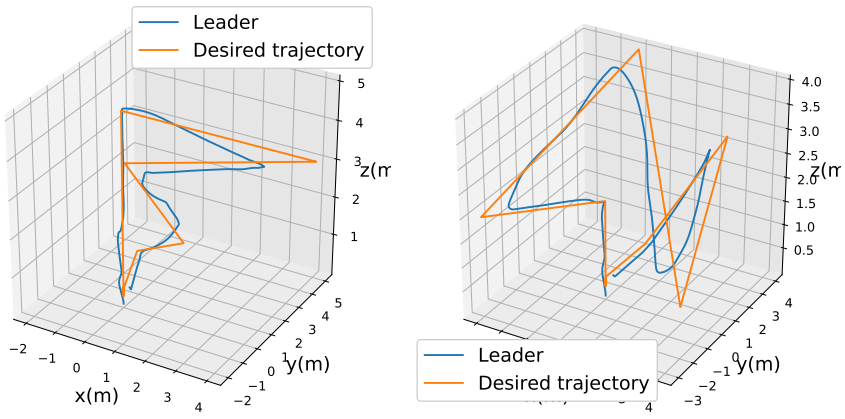


(b) Force in *Exp. A* Traj. I for agent 3.

Figure 5.15

Analyzing *Experiment B* we can compare the results when we change the configuration of connectivity maintenance graph.

For *Experiment B* the leader trajectories are exposed in Figure 5.16. As we can see, there is not significant difference in comparison with *Experiment A*.



(a) Leader in Traj.I (b) Leader in Traj.II

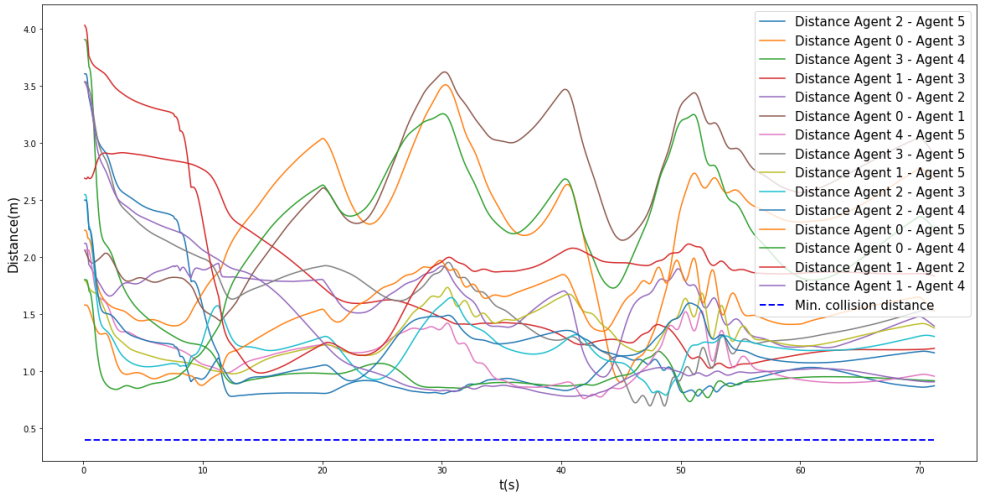
Figure 5.16: Leader route in *Experiment B*

Distances between agents appear in Figure 5.17 and demonstrate that also this experiment accomplish with the distance restrictions.

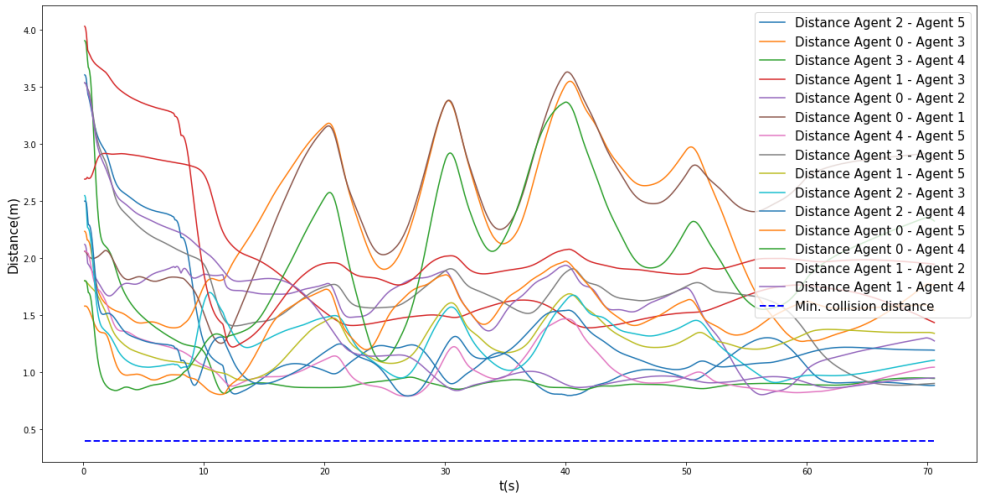
Now, we will plot the collision avoidance and connectivity maintenance terms graphs, as well as the force output, for Crazyflie 0 (Fig. 5.18) and 3 (Fig. 5.19).

For both agent we can not appreciate huge differences in comparison with *Experiment A*, so we can conclude that algorithm is robust when changing the connectivity maintenance graph and the trajectory chosen.

Finally, regarding to the output force we can observe how constant is its value for every agent. In aircraft control this linearity is highly searched because the inertia of aerial robots is fairly wide and using fluctuation values on the forces applied make a really unstable or even impossible flight.



(a) Swarm distance in Traj. I



(b) Swarm distance in Traj. II

Figure 5.17: Swarm distances

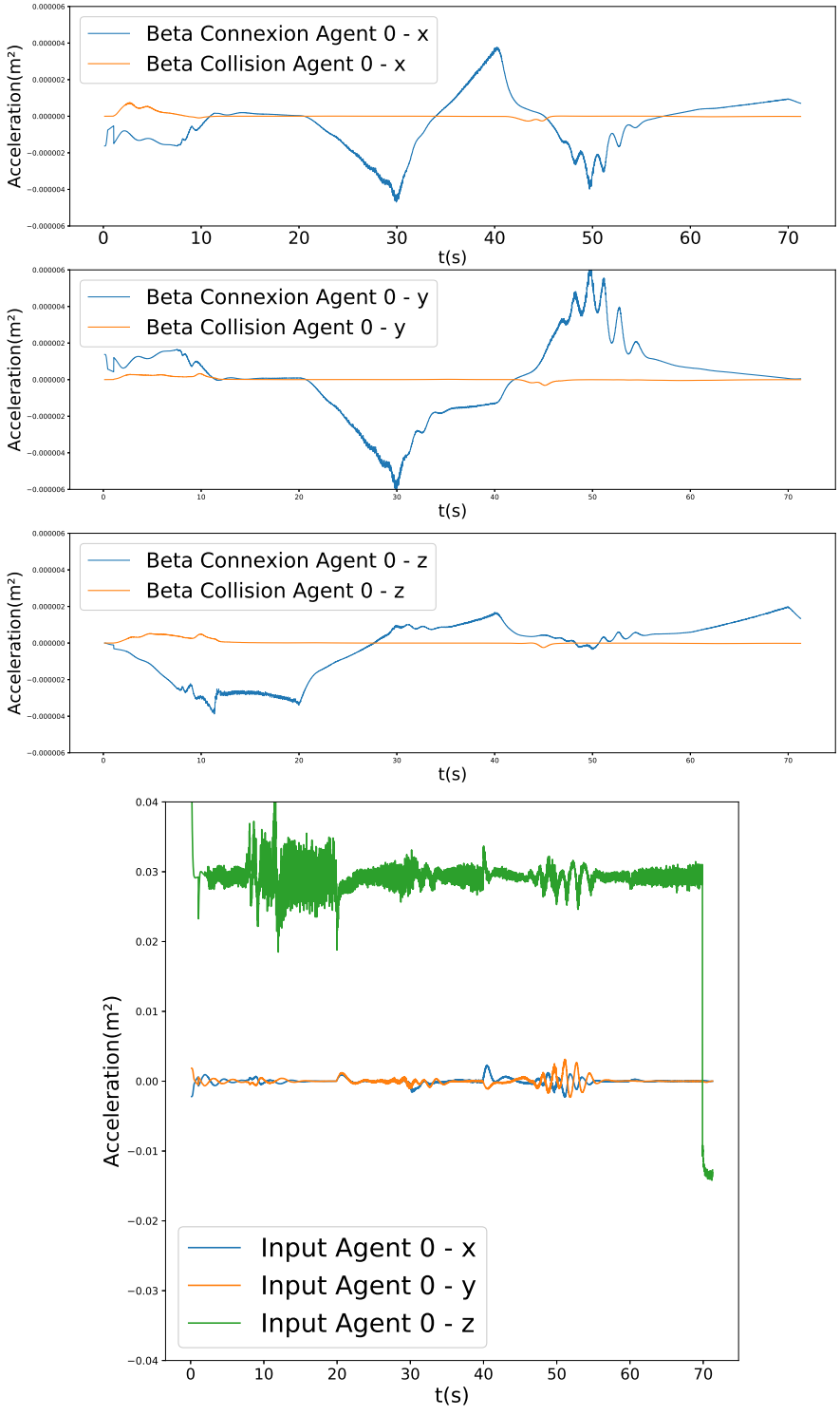


Figure 5.18: Agent 1, Traj. I

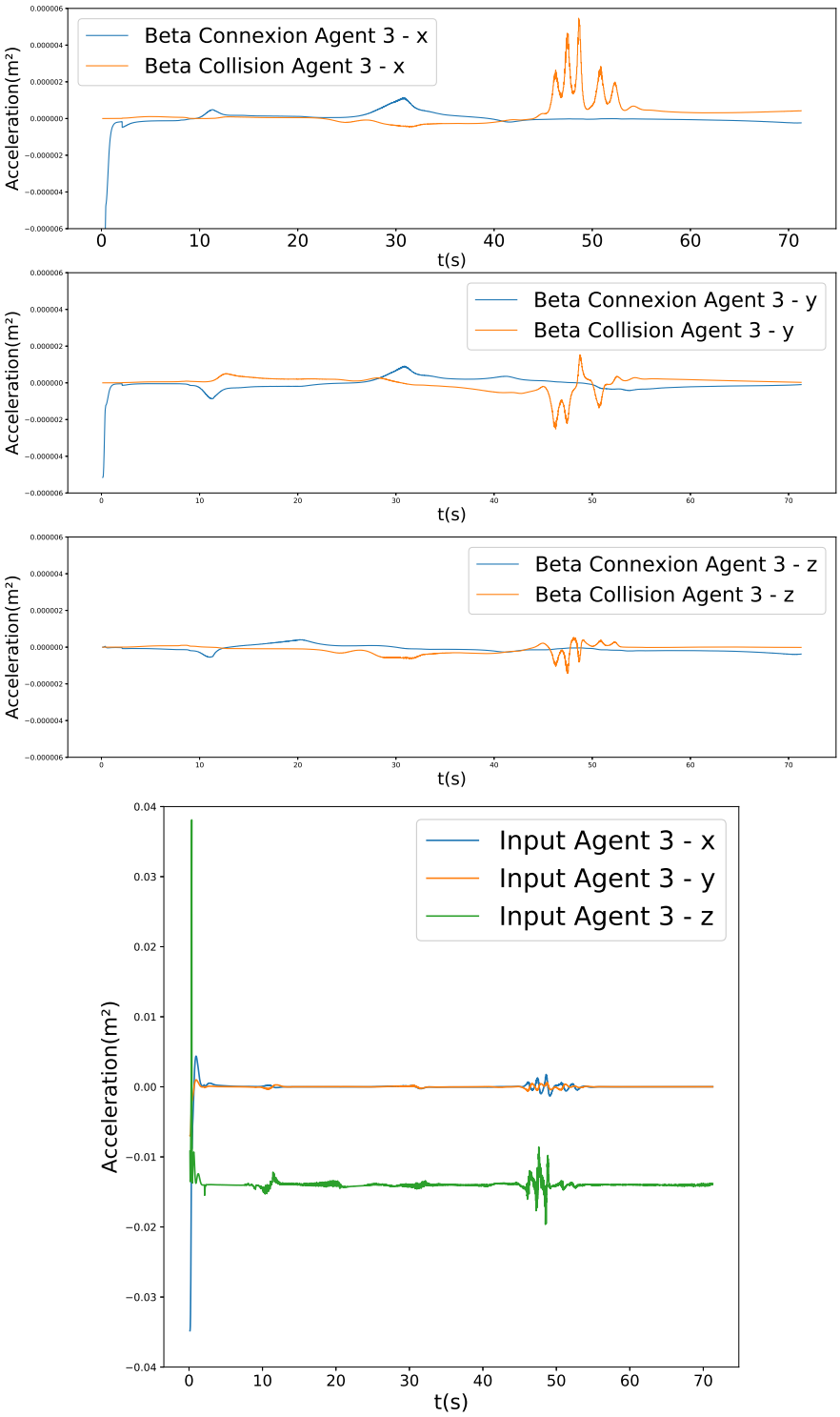


Figure 5.19: Agent 1, Traj. I

Chapter 6

Conclusion and future developments

6.1 Conclusion

We studied the Leader-Follower problem on a drone multi-agent network where direct communication among agents is only available when they are closer enough to sense each other.

We started presenting the problem formulation and its constraints. Specifically, which information is given to the leader and which is its duty and how followers have to behave in order to not collide and to remain together as a swarm.

Afterwards, we introduced the device we will use to make our experiments and its dynamics. Then we propose the designed algorithm to achieve the problem. It is based on graphs maintenance for collision avoidance and connexion between agents in order to keep the swarm together in a decentralized manner.

Finally, we described how the multi-agent system is implemented in ROS and we showed the results of the numerical simulations that we carried out, highlighting distances between robots in the swarm and output control force used. As a conclusion we can say that the control is enough stable and robust when changing configurations or

trajectories. After all, the goal have been successfully achieved.

6.2 Future work

After finishing this thesis we can think about future work that can be done in order go further with the work exposed in this report or even open new related research branches to study.

In the first instance, the main future work is to test this algorithm with real Crazyflies to be able to apply the control in actual coordination tasks. All the code and ROS communication done in this thesis is focused on allow an easy implementation of the control on the real Crazyflies.

Regarding the control algorithm, future works can go in the direction of improving the velocity of the swarm movements. Besides, in order to make it more robust, adding terms to deal with noise and disturbances.

Bibliography

[1] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, Jan 2007.

[2] Ziyang Meng, Wei Ren, and Zheng You. Decentralised cooperative attitude tracking using modified rodriguez parameters based on relative attitude information. *International Journal of Control*, 83(12):2427–2439, 2010.

[3] R Smith and F Hadaegh. Distributed estimation, communication and control for deep space formations. *IET Control Theory and Applications*, 1(2):445–451, 2007.

[4] W. Ren. Distributed cooperative attitude synchronization and tracking for multiple rigid bodies. *IEEE Transactions on Control Systems Technology*, 18(2):383–392, March 2010.

[5] N. Chopra, D.M. Stipanovic, and M.W. Spong. On synchronization and collision avoidance for mechanical systems. pages 3713–3718. IEEE, 2008.

[6] V. Gazi. Swarm aggregations using artificial potentials and sliding-mode control. *Robotics, IEEE Transactions on*, 21(6):1208–1214, 2005.

[7] Hajar Atrianfar and Mohammad Haeri. Flocking of multi-agent dynamic systems with virtual leader having the reduced number of informed agents. *Transactions of the Institute of Measurement and Control*, 35(8):1104–1115, 2013.

[8] Amir Ajorlou, A Momeni, and A G Aghdam. A class of bounded distributed control strategies for connectivity preservation in multi-agent systems. *Automatic Control, IEEE Transactions on*, 55(12):2828–2833, 2010.

[9] Alex Owen-Hill. Difference between robotics and artificial intelligence, July 2017.

[10] Leslie Anne Ballard, S Sabanovic, Jasleen Kaur, and Stasa Milojevic. Junior history. *Robotics and Automation Magazine, IEEE*, 19:114–119, 09 2012.

[11] Eric Roberts. Robotics: A brief history. *Standford University*, 2016.

[12] Stanford's robotic history. *Standford Magazine*, 2014.

[13] Scara. *Robot Hall of Fame*, 2006.

[14] H. Moravec. Fully interconnecting multiple computers with pipelined sorting nets. *IEEE Transactions on Computers*, 28:795–798, 10 1979.

[15] T. Kanade and D. Schmitz. Development of cmu direct-drive arm ii. In *1985 American Control Conference*, pages 703–711, June 1985.

[16] Matthew P. Golombek. The mars pathfinder mission. *Journal of Geophysical Research: Planets*, 102(E2):3953–3965, 1997.

[17] Moseley Dickinson, Jenkins. Roomba pac-man: Teaching autonomous robotics through embodied gaming. 2017.

[18] Lee Park, Kim and Oh. Mechanical design of humanoid robot platform khr-3 (kaist humanoid robot - 3: Hubo). 2005.

[19] K. Watanabe and Y. Yoneda. The world's smallest biped humanoid robot "i-sobot". In *2009 IEEE Workshop on Advanced Robotics and its Social Impacts*, pages 51–53, Nov 2009.

[20] M. A. Diftler, J. S. Mehling, M. E. Abdallah, N. A. Radford, L. B. Bridgwater, A. M. Sanders, R. S. Askew, D. M. Linn, J. D. Yamokoski, F. A. Permenter, B. K. Hargrave, R. Platt, R. T. Savely, and R. O. Ambrose. Robonaut 2 - the first humanoid robot in space. In *2011 IEEE International Conference on Robotics and Automation*, pages 2178–2183, May 2011.

[21] Boston Dynamics. Atlas. *www.bostondynamics.com*.

[22] Boston Dynamics. Pluto plus. *www.idrobotica.com*.

[23] www.bostondynamics.com.

[24] The future of drones. *California University*.

[25] Sally French. Lily drone review. 2018.

[26] Hexh20 pro v2 - review. 2017.

[27] Jeremiah Karpowicz. Airobotics defines how the "drone in a box" model is working for mining, construction and industrial applications. 2018.

[28] Lte drone control used in world-first delivery trial. *www.Rakuten.today.com*, 2017.

[29] PopUp. *www.italdesign.it, www.airbus.com*, 2017.

[30] Keller Rinaudo. How we are using drones to deliver blood and save lifes. *TED talks*, 2017.

[31] Sebastien de Halleux. How a fleet of wind-powered drones is changing our understanding of the ocean. *TED talks*, 2017.

[32] Chien Chern Cheah, Saing Paul Hou, and Jean Jacques E. Slotine. Region-based shape control for a swarm of robots. *Automatica*, 45(10):2406–2411, 2009.

[33] Christos K. Verginis and Dimos V. Dimarogonas. Adaptive leader-follower coordination of lagrangian multi-robot systems under transient constraints.

[34] IvenSense. *MPU 9250 Product Specification Revision 1.0.*

[35] *www.bitcraze.io*.

[36] Benoit Landry. Planning and control for quadrotor flight through cluttered environments. *B.S., Massachusetts Institute of Technology*, June 2015.

[37] Giri Prashanth Subramanian. Nonlinear control strategies for quadrotors and cubesats. *University of Illinois at Urbana-Champaign*, 2015.

[38] *www.github.com/bitcraze*.

[39] M. Fabrice Poirier. Control of the quadcopter crazyflie. *The University of Manchester*, 09 2018.

[40] Francesco Sabatino. Quadrotor control: modeling, nonlinear control design, and simulation. *The University of Manchester*, 06 2015.

[41] Carlos Luis. Design of a trajectory tracking controller for a nanoquadcopter. *Polytechnique Montreal*, 2016.