Universidad Zaragoza
1542

Trabajo Fin de Grado

Mejora iterativa de un sistema de control de trayectorias para un dron

Iterative Improvement of a Trajectory Control System for a Drone

Autora

Raquel Terrer van Gool

Directores

Édgar Ramírez Laboreo

Luis Miguel Riazuelo Latas

Grado en Ingeniería de Tecnologías Industriales

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2024

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a mis tutores, Édgar Ramírez Laboreo y Luis Riazuelo Latas, todo el tiempo que han dedicado a guiarme y enseñarme a lo largo de todo el proceso. Gracias por vuestros consejos y por compartir vuestro conocimiento conmigo. Ha sido un placer trabajar con vosotros.

También quiero agradecer a toda mi familia por no dudar nunca de mí y por su apoyo incondicional. En especial, gracias, Mariela, por todo lo que aprendo de ti, y gracias mamá, gracias papá, porque sin duda, este logro es gracias a vosotros.

Mejora iterativa de un sistema de control de trayectorias para un dron

RESUMEN

El principal objetivo de este trabajo es mejorar las trayectorias de drones cuyas funciones requieren realizar trayectorias repetitivas en numerosas ocasiones como la inspección, la monitorización o vigilancia. En muchas situaciones, dichas trayectorias no se cumplen como es esperado y cometen ciertos errores por diversos motivos. La idea es explotar este comportamiento repetitivo para mejorar la trayectoria del dron, usando para ello datos provenientes de un sistema de captura de movimiento externo y un algoritmo de aprendizaje automático. El trabajo ha sido realizado en la nave del grupo de investigación Robótica, Visión por Computador e Inteligencia Artificial, donde se encuentran las herramientas necesarias para las actividades planteadas.

El trabajo se ha dividido en dos partes principales, una de simulación por ordenador y otra de experimentación real. Tras una primera familiarización con el dron y los sistemas a utilizar, se han procedido a simular en Matlab diferentes tipos de fallos en el vuelo de los drones. Se han planteado varias formulaciones del algoritmo automático de mejora para encontrar el que, una vez aplicado, reproduzca la trayectoria deseada con el menor error posible y que necesite de menos iteraciones para converger. En dichas pruebas, se ha observado un buen funcionamiento del control planteado y una aproximación importante a la trayectoria deseada. Una vez conseguida la mejora, con los resultados obtenidos y la mejor formulación posible, a través de herramientas como Matlab, Python y el sistema de captura de movimiento OptiTrack se ha realizado la validación experimental de los resultados en el dron Tello EDU. En la validación se ha podido implementar el control iterativo y la trayectoria del dron se ha ido aproximando a la deseada.

Finalmente, con el fin de automatizar la elección de los parámetros del algoritmo y acelerar la convergencia de los errores para conseguirla en menos iteraciones, se han realizado varias optimizaciones por simulación con funciones de coste diferentes. Se busca encontrar las ganancias que proporcionen la convergencia más rápida, sigan siendo estables y aproximen la trayectoria a la ideal lo máximo posible. Las mencionadas simulaciones se han realizado de nuevo en Matlab. Tras analizar los resultados, se puede ver cómo se obtiene una convergencia más rápida a costa de mayores oscilaciones en las primeras iteraciones del vuelo. Además, se plantean futuras líneas de investigación para mejorar la formulación planteada, optimizar el rendimiento del dron y explorar nuevas aplicaciones.

Índice

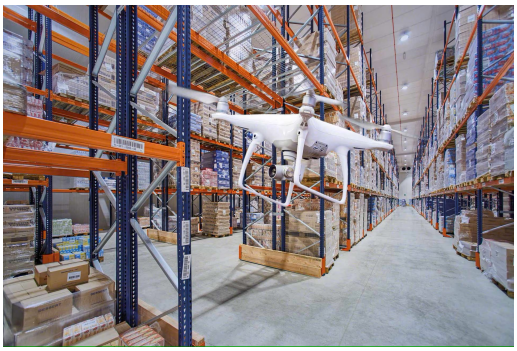
1. Introducción

1.1. Motivación

Las mejoras tecnológicas se introducen rápidamente en los diversos sectores industriales. Una de las que más impacto está teniendo debido a su gran versatilidad es el uso de drones. Los drones aportan soluciones a problemas que antes eran muy costosos, difíciles o peligrosos. Empresas como Amazon han apostado públicamente por esta tecnología en la entrega de paquetes, sin embargo, el uso de drones no se limita solo a esto. Cada vez son más las aplicaciones para las que se utilizan, desde monitorizar cultivos y aplicar fertilizantes y pesticidas en agricultura, a detectar focos en incendios o ayudar a extinguir fuegos forestales.

Existen otro tipo de funciones para las que los drones son muy útiles y permiten ahorrar recursos. Por ejemplo, en tareas de vigilancia [1], monitorización de instalaciones [2] o gestión de inventarios [3]. Cuando dichas aplicaciones son realizadas por drones, permiten mayor precisión con una mejor accesibilidad a lugares que de otra forma no serían accesibles. Estas actividades tienen en común que realizan trayectorias repetitivas en un espacio cerrado. Se va a buscar explotar esta característica de dichas tareas para mejorar la precisión de las trayectorias recorridas para drones más accesibles con errores en el sistema de control.

(a) Tareas de vigilancia



(b) Control de inventarios

Figura 1.1: Ejemplos de tareas repetitivas realizadas por drones.

1.2. Contexto y antecedentes

Previamente, en el grupo de investigación Robótica, Visión por Computador e Inteligencia Artificial, se ha realizado la puesta en marcha del sistema de captura de movimiento externo OptiTrack[4] y un comienzo de experimentación con los drones Tello EDU en el Trabajo de Fin de Grado: *Utilización de un sistema de captura de movimiento externo para el control del movimiento de equipos de drones* [5].

Siguiendo esta línea de trabajo, el presente proyecto busca explorar nuevas vías de investigación dentro del control de drones. Concretamente, este trabajo se centrará en el escenario donde no se puede obtener o no se tiene un modelo de comportamiento del dron y se debe buscar un modo de mejorar la trayectoria. Para ello se continuará con el estudio del dron Tello EDU, el cuál está enfocado a la educación y presenta numerosas facilidades para la investigación en diversos campos, entre ellos el control.

1.3. Objetivos y tareas

El objetivo del presente Trabajo de Fin de Grado es buscar la forma de mejorar la trayectoria de drones con limitaciones tecnológicas que repitan las mismas trayectorias numerosas veces. Para conseguir dicho objetivo se proponen las siguientes tareas:

1. **Investigación sobre trabajos previos existentes.** Estudio de la situación actual del sector y posibles soluciones al problema.
2. **Familiarización con el hardware y software a utilizar.** Comprensión y análisis de las diferentes opciones con las que enfocar el problema. Elección de la mejor opción para comunicarse con el dron y hacer el cálculo del control.
3. **Estudio del comportamiento del dron en la simulación.** En dicha tarea, se dispone de un simulador de drones que permite visualizar el comportamiento bajo diferentes situaciones. Por ejemplo, se pueden añadir perturbaciones y fallos al controlador interno del dron. Se explorará el comportamiento en diferentes trayectorias con el fin de, posteriormente, aplicar la mejor corrección de la trayectoria realizada.
4. **Desarrollo del algoritmo de aprendizaje automático para mejorar la trayectoria.** Se propone el desarrollo de un algoritmo que mejore automáticamente los comandos que se deben enviar para mejorar la precisión en el vuelo.
5. **Optimización de la elección de los parámetros del algoritmo de aprendizaje.** En la búsqueda de mejorar la eficiencia del algoritmo, se plantea la automatización de la elección de los parámetros del algoritmo según diferentes funciones de coste para un comportamiento óptimo.
6. **Validación por simulación y ensayos.** Finalmente, se programará todo de manera que se pueda implementar en una simulación y en pruebas reales.

1.4. Herramientas

Las herramientas que se van a utilizar a lo largo del trabajo son:

- **Matlab.** Será la principal herramienta utilizada. En ella se reciben los datos del sistema de movimiento externo a través de OptiTrackToolbox[6], se realizan las pruebas de simulación, la implementación del algoritmo de mejora de trayectoria y las pruebas reales para comprobar su funcionamiento.
- **Python.** Mediante Python se realizará la comunicación con el dron con el módulo djitellopy [7]. Dicho módulo establece la conexión UDP y utiliza capas superiores para asegurarse de que recibe la información. Contiene una función que permite mandarle al dron las velocidades en x, y, z y la velocidad angular en z (v_ϕ).
- **Optitrack.** Sistema de captura de movimiento externo que devuelve la posición y orientación de los cuerpos rígidos. Se cuenta con una zona de vuelo de 6x6x6 metros para realizar las pruebas.
- **Dron Tello EDU.** Sus limitadas prestaciones y diseño orientado a la educación hacen de él una elección perfecta para las pruebas de simulación reales.

1.5. Estructura de la memoria

A continuación, se detalla la estructura de la memoria:

- ***Capítulo 1. Introducción.*** En este capítulo se explican las motivaciones que llevan al desarrollo del trabajo, los objetivos, las tareas planteadas y las herramientas utilizadas.
- ***Capítulo 2. Descripción del sistema.*** Con el propósito de entender de la mejor forma posible la evolución del trabajo, en este capítulo se habla del modelo matemático. Se definen las condiciones de las simulaciones posteriores y se presentan las trayectorias generadas. También se explica con detalle el hardware empleado en la experimentación real.
- ***Capítulo 3. Iterative Learning Control.*** Es el cuerpo central del trabajo. Se introduce la idea, se exponen diferentes formulaciones y se simula en el ordenador para posteriormente validar los resultados mediante una experimentación real.
- ***Capítulo 4. Optimización.*** Busca mejorar la eficiencia del algoritmo mediante la elección automatizada de los parámetros del mismo. Se desarrollan diferentes funciones de coste y se simulan por ordenador para ver los resultados.
- ***Capítulo 5. Conclusiones.*** Contiene las conclusiones obtenidas en los apartados anteriores.

2. Descripción del sistema

En este capítulo se desarrolla el conjunto de ecuaciones y principios matemáticos que describen el comportamiento del sistema. También se explica la implementación del modelo matemático en el escenario planteado, así como las herramientas de simulación utilizadas, condiciones consideradas y trayectorias analizadas. Finalmente, se detallan los procedimientos experimentales, así como el hardware empleado para contrastar los resultados teóricos con datos empíricos.

2.1. Modelo matemático

En esta sección se planteará el problema a resolver de forma matemática y la teoría necesaria para entender el desarrollo del Trabajo de Fin de Grado. El problema presenta numerosas posiciones y orientaciones en diferentes sistemas y orígenes de referencia. Además, algunas referencias se van moviendo con el paso del tiempo.

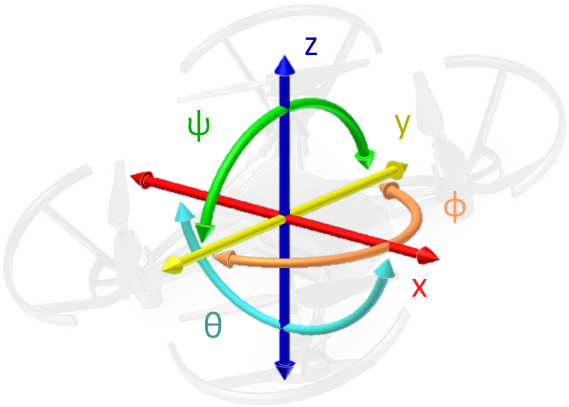


Figura 2.1: Ejes del dron.

Un dron es un sistema con 6 grados de libertad: 3 grados de posición y 3 de orientación. Las orientaciones se expresan mediante la formulación RPY (Roll, Pitch, Yaw) que es lo mismo que la formulación con los ángulos de Euler en orden ZYX (Figura 2.1). Es decir, ϕ representa la rotación en el eje z, después θ representa la rotación en el eje y y finalmente ψ en el eje x. Por lo tanto, la posición y orientación de un dron en el espacio viene dada por la

ecuación 2.1.

$$X = (x, y, z, \psi, \theta, \phi)^T \tag{2.1}$$

Los drones utilizan los giros en θ y en ψ para desplazarse de manera lineal en el plano xy. Por lo tanto, el único giro real que hace el dron para orientarse en el espacio es la rotación en ϕ. Es decir, se puede simplificar a la siguiente ecuación:

$$X = (x, y, z, \phi)^T \tag{2.2}$$

Es importante destacar que cuando se hable del sistema de referencia local del dron, se estará hablando de un sistema de referencia en movimiento, el cual cambia en cada instante con la posición del dron (Figura 2.2). Es por lo tanto de vital importancia entender el instante en el que se halla el dron para calcular la posición de un punto en el espacio respecto a él mismo.

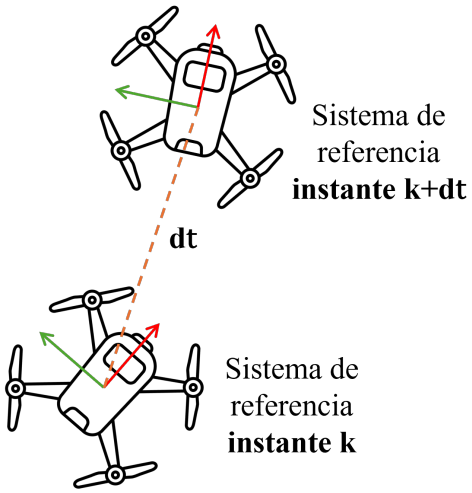


Figura 2.2: Sistema de referencia local del dron.

Para cambiar las posiciones a las diferentes referencias, se utilizarán las matrices de transformación homogéneas, las cuales expresan la posición y la orientación de los sólidos rígidos. Estas matrices permiten, mediante el producto entre ellas, realizar transformaciones geométricas que definen el estado de los sólidos rígidos en el espacio en diferentes referencias.

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Las componentes r_{ij} definen la rotación mientras que el vector t describe la traslación.

En resumen, como se puede observar en la figura 2.3, los sistemas de referencia a tener en cuenta son:

- **La referencia global.** Es una referencia absoluta, estática, no cambia a lo largo del tiempo.
- **La referencia del dron.** Es una referencia local, como se explica en la figura 2.2, cambia en cada instante de tiempo con la posición del dron.
- **La referencia deseada.** Es una referencia local, de manera similar a la referencia del dron, cambia en cada instante. Se mueve según la posición y orientación deseada en cada instante.

En este caso, el sistema de captura de movimiento externo proporciona la posición y orientación del dron en el sistema de referencia global. La referencia deseada es la trayectoria que se quiere conseguir con el dron. Con ello, ya se puede calcular la referencia deseada desde el punto de vista del dron, lo que equivale al error cometido durante la trayectoria.

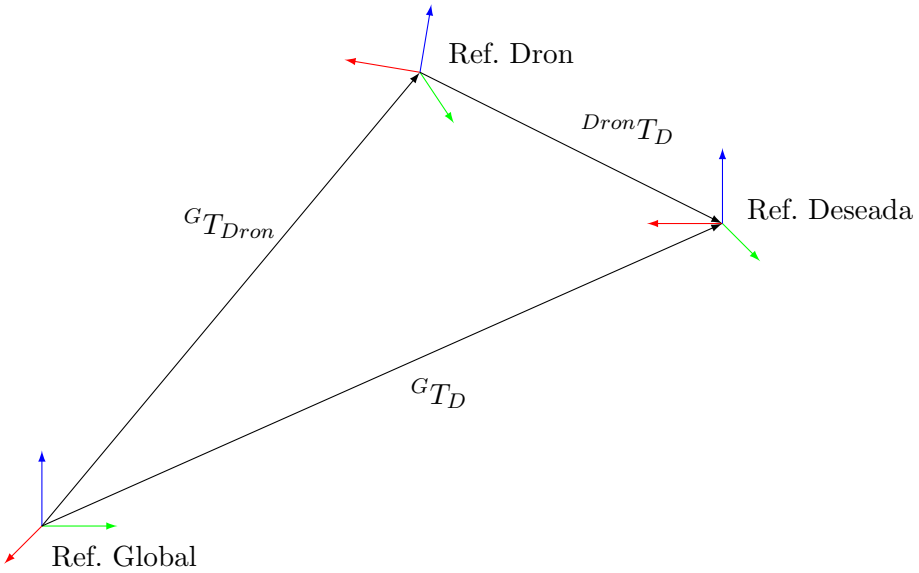


Figura 2.3: Sistemas de referencia del problema

En la figura 2.3, $^{G}T_{Dron}$ es la matriz homogénea de transformación del dron vista desde la referencia global, $^{G}T_D$ es la matriz homogénea de transformación de la posición deseada vista desde la referencia global y $^{Dron}T_D$ es la de la posición deseada vista desde la referencia del dron. En este caso y para el problema a resolver, esta transformación coincide con el error que comete el controlador del dron en cada instante. Para hallar la posición deseada desde el punto de vista del dron se calcula con la transformadas de la siguiente forma:

$$^{D}T_T = (^{G}T_D)^{-1} *^{G} T_T \tag{2.4}$$

2.2. Simulación teórica

Para encontrar la forma más eficaz y eficiente de mejorar la trayectoria del dron, se cuenta con un simulador de vuelo implementado en Matlab. El simulador es un modelo simplificado desarrollado por el grupo de investigación, por ahora solo disponible de forma interna. Dicho simulador es capaz de reproducir el vuelo ideal del dron y el vuelo real que realiza debido a fallos en su controlador. Se pueden aplicar errores en la ganancia y en las constantes temporales del controlador, además de añadir perturbaciones externas. Esto simula las posibles diferencias entre el valor solicitado y el real, además del tiempo que se tarda en alcanzar la respuesta (τ).

Algorithm 1 Funcionamiento del simulador

velocidades = [v_x, v_y v_z, v_ϕ] ▷ Matriz 4xN

% Definir condiciones de simulación
Perturbación externa = [x, y, z]
Ganancias del controlador = [k_x, k_y, k_z, k_ϕ]
Constantes temporales del controlador = [τ_x, τ_y, τ_z, τ_ϕ]
Posición inicial = [x_0, y_0, z_0, ϕ_0]

% Simulación
[posición_real, posición_deseada, t_ref] = simular_dron(condiciones_simulación, velocidades)

En el algoritmo 1 se pueden observar los parámetros que hay que introducir y las variables que nos devuelve. La generación de velocidades debe ser una matriz 4xN, donde N son los valores enviados al dron cada T periodo de tiempo y las cuatro filas representan las velocidades que pueden ser enviadas al dron: v_x, v_y, v_z y v_ϕ. Funciona con cualquier serie de valores que generen todo tipo de trayectorias. Tras la simulación, se obtienen los valores de posición reales y deseadas.

Los drones objeto de estudio no cuentan con sistema de posicionamiento, sino que deben ser controlados mediante una serie de velocidades en sistema de referencia local. Por lo tanto, para controlar el dron, en primer lugar, se genera un archivo con las velocidades que debe seguir el dron para cumplir con la trayectoria deseada. Sin embargo, el dron comete ciertos errores, lo que lleva a que el vuelo no sea ideal. En los siguientes apartados se explicará la generación de las trayectorias y las condiciones de simulación que se seguirán en posteriores simulaciones. En la figura 2.4, se puede ver un esquema del funcionamiento del sistema.

2.2.1. Condiciones de las simulaciones

Los drones objeto de estudio no cuentan con un sistema de retroalimentación que mejore la trayectoria mediante control en bucle cerrado, lo que provoca que los errores del controlador afecten directamente al recorrido. A fin de entender y mejorar la trayectoria se va a estudiar cómo afectan al vuelo del dron cada tipo de error del controlador y las perturbaciones.

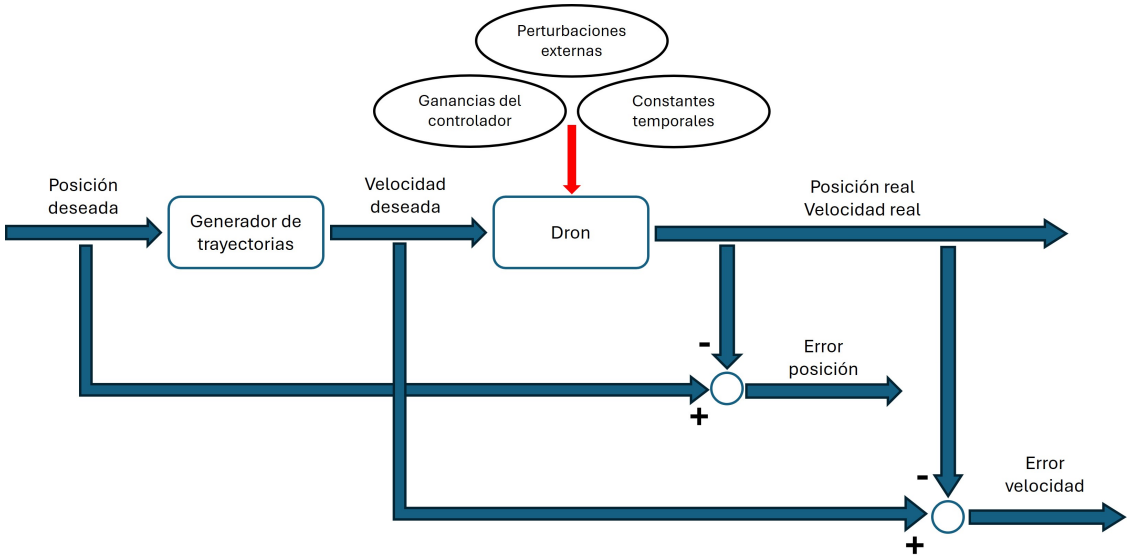


Figura 2.4: Esquema de funcionamiento del sistema.

2.2.1.1. Ganancias del controlador

Cuando existe error en las ganancias del controlador, el dron no alcanza las velocidades solicitadas porque el controlador está mal modelado o simplemente por una insuficiente calidad de los componentes del dron. Se puede ver el comportamiento del dron por fallos de ganancia del controlador en el anexo C.

2.2.1.2. Constantes temporales

El modelo del simulador considera que la velocidad del dron converge a la velocidad deseada con una dinámica de primer órden a través de la constante temporal. En el anexo C se corrobora que el comportamiento del dron sigue esta dinámica para la posterior experimentación.

Evaluar diferentes constantes temporales permite simular el error en la trayectoria que provocan los motores durante el tiempo que tardan en alcanzar el 63,2 % de la velocidad final real (no la solicitada). Idealmente, este tiempo es lo más pequeño posible. Se puede ver el comportamiento del dron ante fallos en la constante temporal del controlador en el anexo C.

2.2.1.3. Perturbaciones externas

En esta variable podrían incluirse distintos elementos como una corriente de aire constante u otro tipo de errores no previamente mencionados que cometa el dron. Los errores corregibles serán aquellos que se mantienen o cuya variación es muy leve a lo largo de las iteraciones.

En este tipo de errores se le incluirán los errores que no se puedan categorizar en los dos explicados previamente.

En la realidad, los errores por dichos motivos no son distinguibles a no ser que se cuente con el modelo del dron y datos reales de su propio vuelo. Sin embargo, para el desarrollo del trabajo, el poder aplicar los diferentes defectos de manera independiente ayuda a entender de qué forma se puede aplicar el control para que tenga el mejor rendimiento posible. Con el objetivo de simular un comportamiento real, se establecen los valores mostrados en la tabla 2.1 para las posteriores simulaciones, salvo que se indique lo contrario.

Perturbaciones externas		**Ganancia del controlador**		**Constante temporal**	
P_x	0	k_x	0,7	τ_x	0,01
P_y	0,5	k_y	1	τ_y	0,1
P_z	0	k_z	0,7	τ_z	0,1
		k_ϕ	0,9	τ_ϕ	0,01

Tabla 2.1: Condiciones de simulación.

2.2.2. Generación de trayectorias

Para poder simular diferentes escenarios, son necesarias la generación de diferentes trayectorias. La generación de trayectorias, conociendo las posiciones y orientaciones por las que debe pasar el dron, define una serie de valores de velocidad en cada periodo, T, en los diferentes ejes. Deben ser velocidades en referencia local, es decir, donde los ejes se mueven en cada instante con el dron. Cuanto más pequeño es el periodo, mayor capacidad de actuación sobre el sistema se tiene. Los códigos generan un archivo que en caso de ser comandado, reproduce la trayectoria deseada. Se plantean las siguientes opciones:

2.2.2.1. Trayectoria helicoidal

Se trata de una trayectoria que forma una hélice. Está compuesta por velocidades constantes en v_x, v_y y v_ϕ. Es la principal trayectoria estudiada ya que presenta cierta complejidad y al mismo tiempo permite un análisis claro e intuitivo. Como se puede ver en la figura 2.5a, tras la simulación con las condiciones detalladas en la tabla 2.1, el dron comete cierto error tratando de seguir la trayectoria deseada.

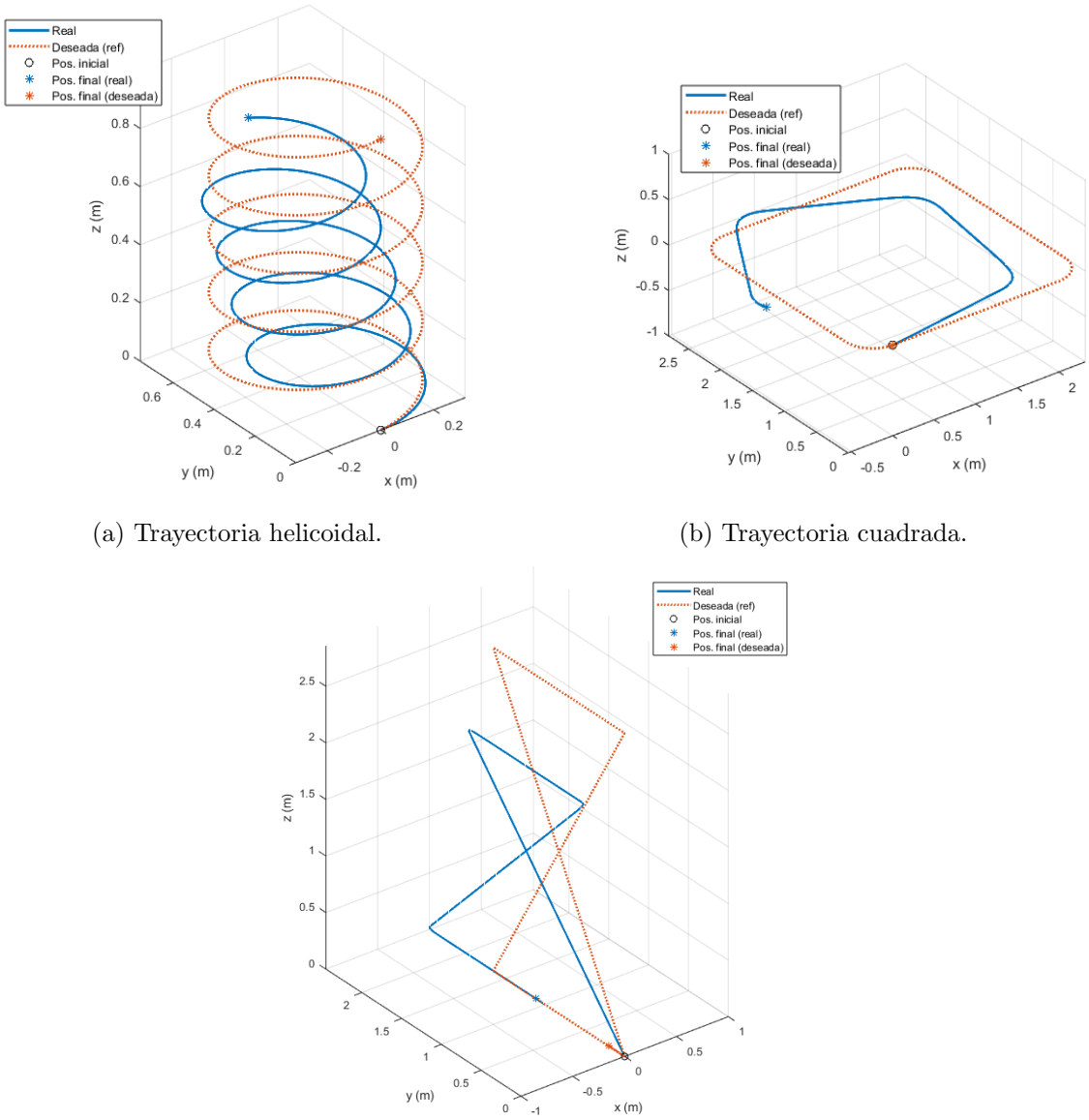
2.2.2.2. Trayectoria cuadrada

Dicho código busca realizar una trayectoria cuadrada con continuidad de velocidad en las esquinas. Cuenta con velocidad en x constante y en las esquinas aplica v_ϕ para girar. En la figura 2.5b se puede observar cómo el fallo que comete el controlador en las velocidades,

impide que el dron vuela a su lugar de inicio aunque intente trazar un cuadrado y volver al mismo punto.

2.2.2.3. Trayectoria reloj de arena

Sería la opción más próxima a una trayectoria mediante waypoints. No se le ha aplicado ningún tipo de continuidad en velocidades ya que se busca analizar el comportamiento de la mejora de trayectoria ante dicho escenario. Se puede observar la trayectoria deseada y la que sigue el dron en la figura 2.5c. En esta trayectoria no afecta la orientación del dron ya que se mantiene constante en todo el recorrido.



(a) Trayectoria helicoidal.

(b) Trayectoria cuadrada.

(c) Trayectoria reloj de arena.

Figura 2.5: Trayectorias planteadas.

2.3. Metodología práctica

En la sección a continuación, se explica el dron empleado para los experimentos reales, el sistema de captura de movimiento externo utilizado y se detalla el enfoque seguido para la implementación del código en la experimentación.

2.3.1. Hardware Empleado

2.3.1.1. Dron Tello EDU

El dron objeto de estudio en las validaciones será el Tello EDU [8], un dron desarrollado por la empresa Ryze Tech en colaboración con DJI, empresa líder en fabricación de drones, e Intel, líder en el diseño y fabricación de microprocesadores. Equipado con una cámara de 5 megapíxeles y preparado para aprender a programar, permite el envío y la recepción de información a través de una conexión Wi-Fi y protocolo UDP. También cuenta con un sistema de estabilización de vuelo basado en un barómetro y una Unidad de Medición Inercial (IMU), complementados con una cámara orientada hacia abajo, la cual utiliza visión computacional para reconocer patrones en el suelo.

Figura 2.6: Dron Tello EDU de Ryze Tech.
Imagen de `https://www.creativakids.com/tello_edu.php`

Para estudiar las limitaciones del dron a tener en cuenta para el desarrollo de un control que permita la mejora de su trayectoria, se han realizado diversas pruebas con él. Como resultado, se han obtenido las siguientes conclusiones:

- El dron Tello EDU no permite acceder al control de velocidad de los motores y su frecuencia de recepción y envío de velocidades es demasiado lenta para un control en bucle cerrado (10 Hz).

- El protocolo de comunicación UDP no garantiza la entrega de los paquetes de datos, por lo que puede haber pérdida ocasional de datos.
- No cuenta con un sistema de posicionamiento, por lo que no hay realimentación de la posición real.
- El sistema de estabilización depende mucho de las condiciones exteriores de luz y es sensible a las perturbaciones.

Para el envío de comandos al dron se han probado diferentes formas de comunicación a través de conexiones desde Matlab, Python o la propia aplicación de Tello [9] con el fin de encontrar la mejor opción con la que elaborar el control. En este punto se encuentran varios problemas con la comunicación con el dron:

- Utilizar el paquete de Ryze Tello Drones [10] de Matlab permite solamente el control a alto nivel a través de comandos muy genéricos, lo que restringe mucho las posibilidades. Para más información, véase el anexo A.
- La aplicación de Tello [9] no permite la programación y exige mandar comandos desde el propio dispositivo smartphone.
- La comunicación con el dron mediante una conexión UDP a través de Matlab pierde muchos paquetes de información, lo que no permitirá un control adecuado del dron. Para más información, veáse el anexo B.

La mejor opción para comunicarse con el dron por su fiabilidad y la variedad de opciones que presenta es a través del módulo de Python *DJITelloPy* [7]. Dicho módulo cuenta con numerosas funciones para programar el dron. Permite establecer una conexión con el dron y este se asegura de que el dron recibe los paquetes de información enviados. En el anexo E se explican más en profundidad las funciones utilizadas y la comunicación desarrollada con el dron para las simulaciones.

2.3.1.2. Sistema OptiTrack

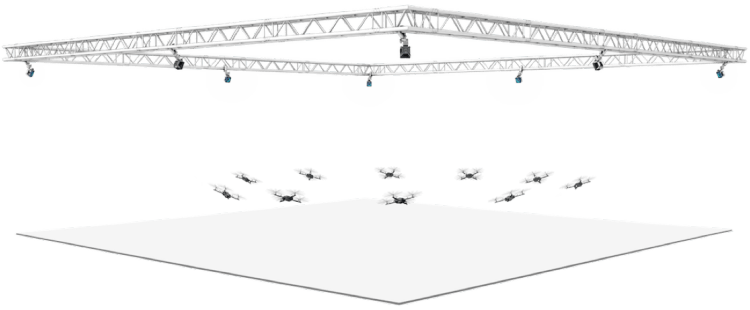
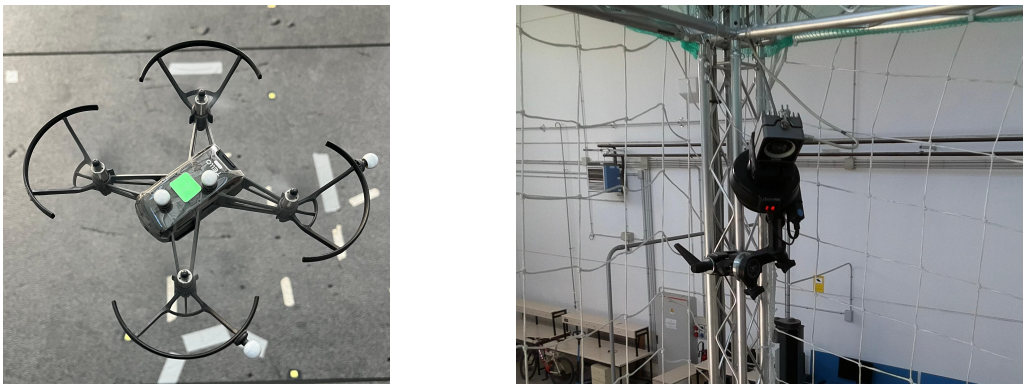


Figura 2.7: Arena Optitrack[4].

OptiTrack es un sistema de captura de movimiento externo de alta precisión. Se emplea en aplicaciones que requieren un seguimiento de posición y orientación de cuerpos rígidos en

el espacio. Proporciona soluciones en sectores como la robótica, la ciencia del movimiento, la realidad virtual o la animación [4]. Utiliza cámaras infrarrojas de alta precisión para detectar marcadores reflectantes en el espacio. Dichos marcadores son pequeñas esferas que reflejan la luz infrarroja emitida por las cámaras. Una vez capturan el marcador, triangulan la posición para determinar su posición y orientación. Se pueden observar dichos marcadores y las cámaras infrarrojas en la figura 2.8. La imagen de la cámara de captura de movimiento ha sido tomada con el dron Tello EDU. La zona de vuelo es un espacio de 6x6x6 metros.



(a) Marcadores del dron Tello EDU. (b) Cámara infrarroja.

Figura 2.8: Hardware del sistema de captura de movimiento Optitrack.

El software principal de OptiTrack es Motive, el cual es capaz de hacer streaming de datos y permite la integración con otras herramientas como Matlab. En la figura 2.9 se puede observar el software y la interfaz de la que dispone para mostrar el movimiento de los cuerpos rígidos. Reconoce los distintos cuerpos rígidos mediante unos marcadores colocados en diferentes posiciones. Cada distribución la asocia a un cuerpo rígido. Los datos recogidos servirán para desarrollar el control y la mejora de trayectorias del dron Tello EDU.

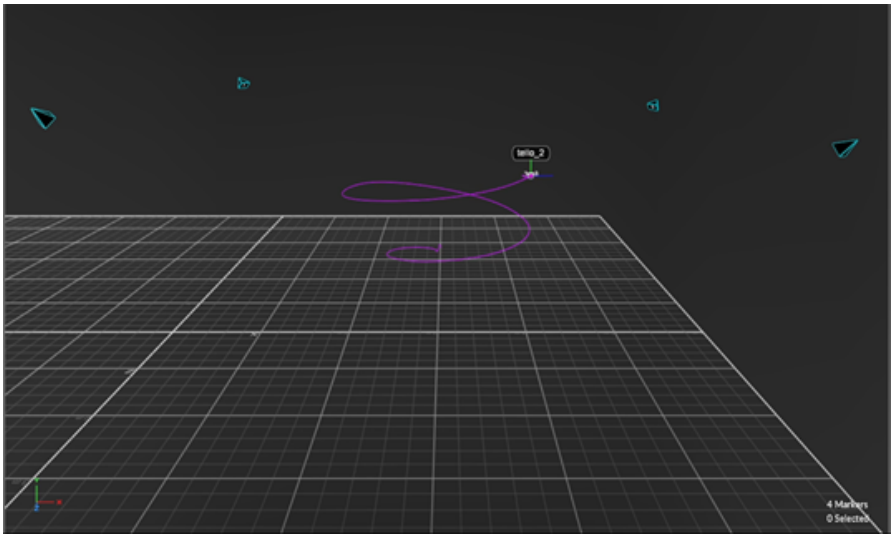


Figura 2.9: Captura de movimiento del dron durante un experimento mediante Motive.

2.3.2. Experimentos reales

La implementación de todos los sistemas y el software necesario para obtener las posiciones y orientaciones deseadas y reales no es trivial. La arquitectura del código desarrollado se puede ver en la figura 2.10 y la estrategia de implementación ha sido la siguiente:

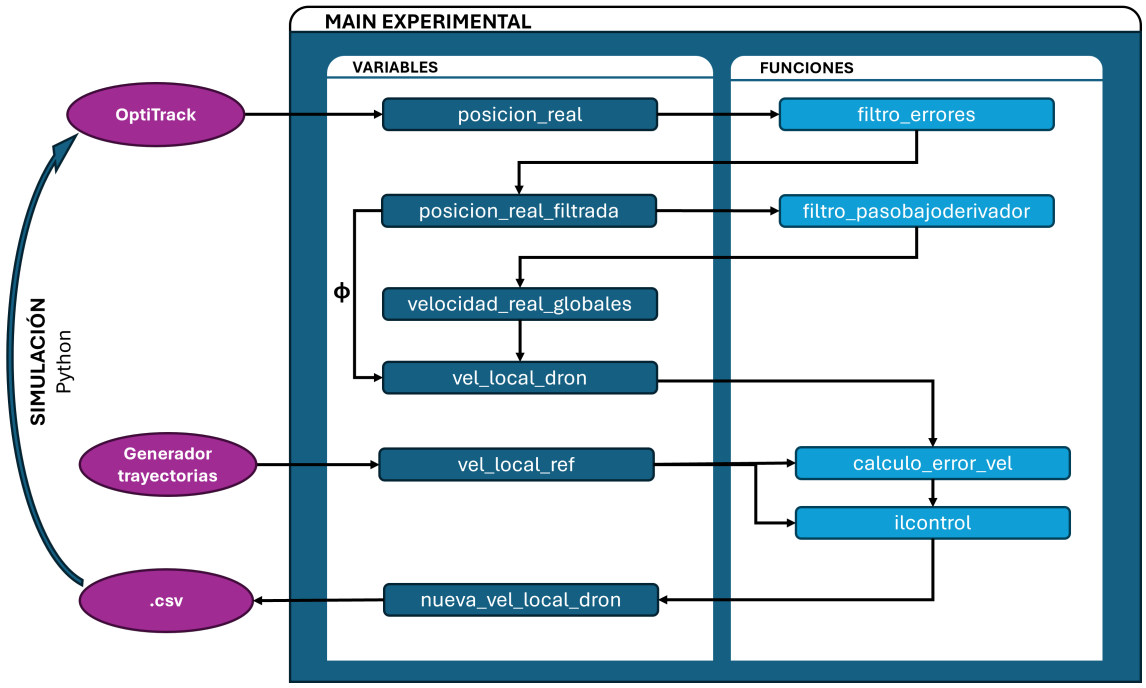


Figura 2.10: Arquitectura del código experimental.

1. **Generación del archivo inicial con la trayectoria deseada.** Se crea un archivo csv con el código generador de trayectorias mencionado en el apartado 2.2.2.

2. **Comunicación con el dron y envío del archivo con las velocidades mediante un script de Python.** Se pueden encontrar las funciones utilizadas para la comunicación con el dron en el anexo E.

3. **Capturar datos mediante el OptiTrack.** Una vez capturados, dichos datos requieren un filtrado ya que hay algunos repetidos e instantes de tiempo que fallan y no recogen datos. Se filtran todos los datos recogidos y se realiza una normalización. En la figura 2.11 se puede observar la zona de vuelo en la que se realizan los experimentos en la nave del grupo de investigación.

4. **Cambio de sistemas de referencia.** Una vez se tienen los datos de la trayectoria en sí, se realiza una transformación de los valores de posición y orientación para poder compararlos con la referencia. Esto se hace trasladando el primer valor de la trayectoria al origen de la referencia global y rotando para que coincidan en rotación el primer valor de la trayectoria deseada y la trayectoria real. De esta manera, se puede obtener

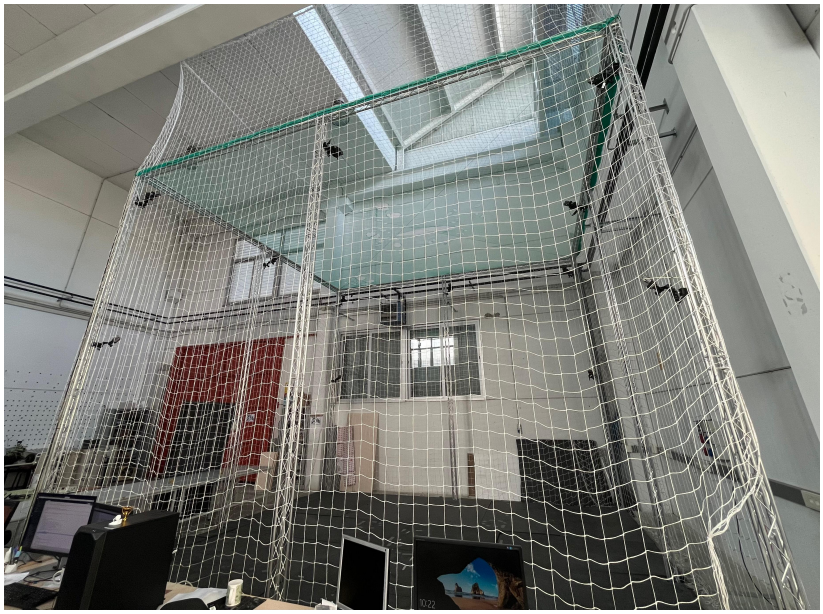


Figura 2.11: Zona de vuelo del OptiTrack.

el error de posición comparando los datos reales y la trayectoria deseada. Finalmente, se interpolan los valores para que tengan el periodo adecuado, el cual viene determinado por el dron, ya que si se le envían datos con una frecuencia superior a 12,5 Hz es altamente probable que pierda algún dato.

5. **Cálculo de la velocidad desarrollada por el dron.** Para el cálculo de la velocidad real desarrollada por el dron, es necesario filtrar la señal debido a que las medidas contienen ruido por naturaleza. Aprovechando dicha necesidad, se ha diseñado un filtro paso bajo con derivador para que la señal, que son las posiciones, salga filtrada y derivada. Con dicha velocidad y la orientación del dron se puede calcular la velocidad en referencia local. Se pueden encontrar más detalles sobre el filtrado de las señales en el anexo F. De esta manera, se obtiene el error de velocidad.

3. Iterative Learning Control

3.1. Introducción

Tal y como se ha comentado en capítulos anteriores, la finalidad de este trabajo es mejorar la trayectoria realizada por drones con limitaciones tecnológicas y que realizan una trayectoria repetitiva. Para ello se propone una técnica de control cuyo principal objetivo es mejorar la precisión de la trayectoria en cada repetición o iteración, utilizando los datos obtenidos en vuelos previos. Dicha técnica es conocida como Iterative Learning Control (ILC).

Los inicios del Iterative Learning Control se remontan a los 80, cuando se menciona por primera vez una técnica para mejorar el control de sistemas repetitivos, como brazos de robot, en un artículo titulado *Bettering operation of dynamic systems by learning: A new control theory for servomechanism or mechatronics systems* [11]. El concepto surge de la limitación de los controles hasta aquel entonces por desarrollar una forma óptima de control cuando había muchas incertidumbres o perturbaciones en el modelo.

En este caso, se plantea un escenario en el cual no se cuenta con el modelo de un dron cuyas especificaciones presentan ciertas limitaciones y no es capaz de seguir con éxito una trayectoria. El dron desempeña una tarea cuya trayectoria es repetitiva y precisa realizarla con la mayor precisión posible. Para ello, en este capítulo se presenta una posible solución aplicando la técnica de control de Iterative Learning Control.

3.2. Conceptos teóricos

3.2.1. Formulación

Existen diferentes tipos de formulación para el algoritmo ILC. Algunos, como la función de aprendizaje por inversión de la planta tienen una rápida convergencia y precisión, sin embargo, dependen de tener un modelo muy fiable del sistema, lo que se contrapone al escenario planteado. En otros, como el ILC con filtro Q (Q-ILC), el diseño de sus parámetros también dependen de la planta [12]. Para el escenario a analizar se utilizará la formulación de función de aprendizaje tipo PD. La cual se formula de la siguiente manera:

$$u_{j+1}(k) = u_j(k) + k_p e_j(k+1) + k_d(e_j(k+1) - e_j(k)) \tag{3.1}$$

Donde j representa el número de iteración y k el instante de tiempo dentro de la trayectoria, k_p es la ganancia del término proporcional y k_d del término derivativo. Algunos autores utilizan el error en el mismo instante, $e_j(k)$, en el término proporcional. Se ha decidido utilizar el error en el siguiente instante porque el error converge más rápidamente (Véase apartado 3.3.1.1: *Elección de* $e_j(k)$ *o* $e_j(k+1)$*).*

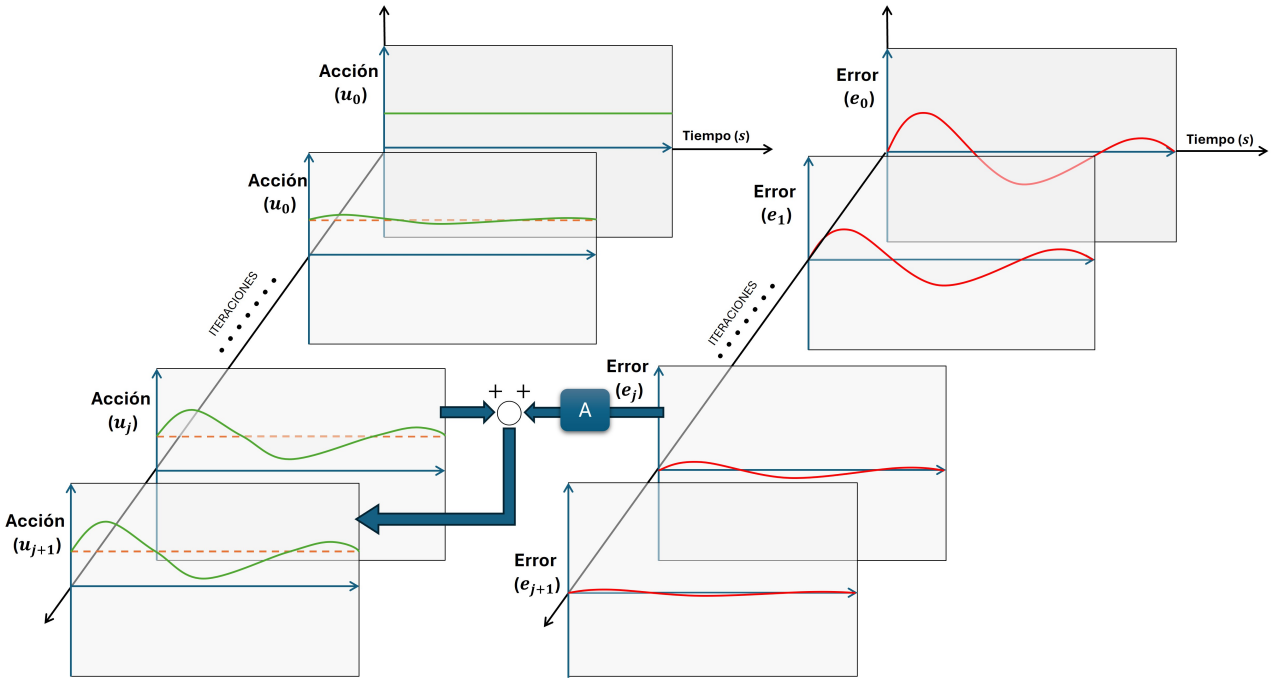


Figura 3.1: Diagrama de bloques del control.

La figura 3.1 muestra el funcionamiento del algoritmo de control. Utiliza los datos del error y la acción anterior para corregir la siguiente iteración e ir modificando y mejorando el error con el paso de las iteraciones.

3.3. Formulación aplicada: posibles alternativas

Aplicando la teoría al caso en cuestión, el algoritmo de Iterative Learning Control se basa en utilizar repeticiones anteriores de una misma trayectoria del dron y mediante el ajuste de una ganancia adecuada, corregir dichos errores en cada eje iterativamente, mejorando la precisión tras cada vuelo. Se espera conseguir el comportamiento explicado en la figura 3.2. Para aplicar el ILC a la mejora de trayectoria se suponen independientes los movimientos en dichos ejes.

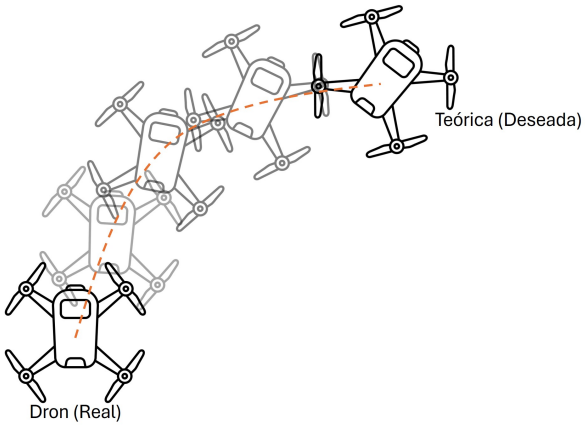


Figura 3.2: Resultado esperado con el paso de las iteraciones.

3.3.1. Formulación A: error en posición

En esta formulación, se calcula el error de posición en los tres ejes y el error de rotación en el eje z comparando la posición real del dron y la posición deseada (Figura 3.3). La acción son las velocidades enviadas al dron, mientras que el error está expresado en metros en el caso de la posición y en radianes en el caso de la orientación. La formulación completa quedaría entonces de la siguiente forma:

$$v_{j+1}^{n}(k) = v_{j}^{n}(k) + k_p e_{j}^{n}(k+1) + k_d(e_{j}^{n}(k+1) - e_{j}^{n}(k)) \tag{3.2}$$

Siendo n los cuatro grados de libertad del dron: x, y, z y ϕ. Además del error medido en posición para x, y y z y orientación para ϕ.

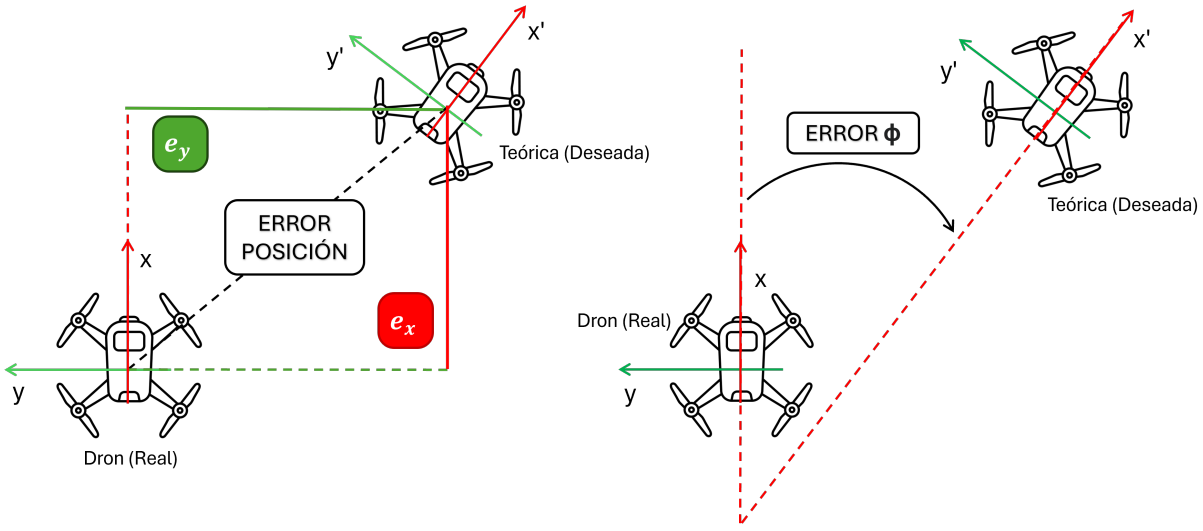


Figura 3.3: Errores utilizados por el Iterative Learning Control.

3.3.1.1. Elección de $e_j(k)$ o $e_j(k+1)$

En este apartado se explica la elección del mejor instante temporal, si $e_j(k)$ o $e_j(k+1)$, para el término proporcional en la formulación del algoritmo ILC. Para ello, se han realizado las mismas pruebas bajo las mismas condiciones de simulación teniendo en cuenta ambas formulaciones con el error siendo de posición. Se seguirá una trayectoria helicoidal durante 100 segundos con un periodo de 0,1 segundos. El resto de las condiciones de la simulación son las mostradas en las tablas 3.1 y 3.2.

Perturbaciones externas		**Ganancia del controlador**		**Constante temporal**	
P_x	0	k_x	0,9	τ_x	0,1
P_y	0,5	k_y	1	τ_y	0,01
P_z	0	k_z	0,8	τ_z	0,1
		k_ϕ	1	τ_ϕ	0,1

Tabla 3.1: Condiciones de simulación para la elección de la formulación.

Ganancias proporcionales		**Ganancias derivativas**	
k_{px}	0,001	k_{dx}	0
k_{py}	0,001	k_{dy}	0
k_{pz}	0,001	k_{dz}	0
$k_{p\phi}$	0,001	$k_{d\phi}$	0

Tabla 3.2: Condiciones de las ganancias del ILC para la simulación.

El error es la distancia entre el punto deseado y el punto real en el que ha estado el dron. Para comparar la efectividad de cada método se ha utilizado la raíz del error cuadrático medio (RMSE) en términos de distancia euclidiana. Dicha medida permite comparar la precisión del vuelo en todos los ejes a lo largo del tiempo. También se ha tenido en cuenta el RMSE de la orientación. Además, se ha calculado la desviación estándar (STD) de los errores instantáneos en las dos opciones para aportar algo más de información sobre la distribución de los errores. En las tablas 3.3 y 3.4 se puede ver un resumen de los resultados de posición comparando el error inicial sin aplicar ningún algoritmo y ambas formulaciones tras 200 iteraciones.

$e_{inicial}$ **(m)**		$e_j(k)$ **(m)**		$e_j(k+1)$ **(m)**	
RMSE	1,5524	RMSE	0,4767	RMSE	0,3594
STD	0,9178	STD	0,2228	STD	0,1251

Tabla 3.3: Errores en posición cometidos en las diferentes formulaciones comparado con el error en la trayectoria inicial (Iteración 0).

$e_{inicial}$ **(rad)**		$e_j(k)$ **(rad)**		$e_j(k+1)$ **(rad)**	
RMSE	$9{,}7719 * 10^{-4}$	RMSE	$8{,}4542 * 10^{-4}$	RMSE	$0{,}1748 * 10^{-4}$
STD	$0{,}9821 * 10^{-4}$	STD	$7{,}9149 * 10^{-4}$	STD	$0{,}1584 * 10^{-4}$

Tabla 3.4: Errores en orientación cometidos en las diferentes formulaciones comparado con el error en la trayectoria inicial (Iteración 0).

Es decir, en posición, la opción con $e_j(k)$ presenta una reducción del error del 69,29 % mientras que con $e_j(k+1)$ mejora en un 76,85 %. También es mejor la desviación media con el error en el siguiente instante. Lo cual se explica, ya que cuando se asigna una velocidad para corregir un error, la corrección no es instantánea sino que se afecta al instante siguiente. Cabe destacar que la mejora en orientación pasa de ser muy leve (13,48 %) a mejorar en un 98,21 %. Es decir, para la formulación del algoritmo se utilizará $e_j(k+1)$ en el término proporcional quedando la ecuación 3.1.

3.3.2. Formulación B: añadiendo instantes de tiempo

Como se ha explicado en el apartado 3.3.1.1 Elección de $e_j(k)$ o $e_j(k+1)$, utilizar el error de un instante de tiempo más avanzado mejora globalmente el rendimiento del control. Es por ello que se propone una formulación que implementa los tres instantes de tiempo siguientes. Cada instante puede tener una ganancia diferente, lo que permite dar más importancia al error que más interese a la hora de corregir la velocidad en la siguiente iteración.

$$v_{j+1}(k) = v_j(k) + k_{p1}e_j(k+1) + k_{p2}e_j(k+2) + k_{p3}e_j(k+3) \tag{3.3}$$

3.3.3. Formulación C: error en velocidades locales

Este tipo de formulación es similar al explicado en el apartado 3.3.1, sin embargo, utiliza el error de velocidad en referencia local en lugar del error de posición en cada eje. La formulación queda de forma similar:

$$v^n_{j+1}(k) = v^n_j(k) + k_p e^n_j(k+1) + k_d(e^n_j(k+1) - e^n_j(k)) \tag{3.4}$$

Siendo n los diferentes ejes ya comentados previamente. En este caso, los errores también serán medidos en las unidades del sistema internacional pero para velocidades (m/s) y velocidades angulares (rad/s).

3.4. Estabilidad

El resultado del control depende en gran medida de la elección adecuada de las ganancias del algoritmo. Se deben elegir las ganancias más grandes posibles sin que se desestabilice el sistema ni tenga demasiadas oscilaciones, pero lo suficientemente grandes para que converja lo antes posible. La elección de las ganancias dependerá de cómo sea el sistema y las perturbaciones que soporte.

Cuando un sistema es estable se cumple la ecuación 3.5. En el control presentado, no existen técnicas para asegurar la estabilidad como las que existen para los controladores clásicos. Por lo tanto para asegurar la estabilidad, la elección de ganancias deberá ser la adecuada acorde al sistema.

$$\lim_{j\to\infty} \|e_j\|_2 = 0 \tag{3.5}$$

Por lo tanto en la elección de las ganancias para la ecuación 3.1 se deberá tener en cuenta:

- Una k_p muy grande puede provocar inestabilidad, especialmente si el sistema tiene constantes temporales demasiado elevadas. Por otro lado, si es demasiado pequeña puede necesitar muchas iteraciones para converger.

- La k_d puede corregir oscilaciones causadas por la k_p. Sin embargo, el efecto de la k_d puede causar movimientos o acciones erráticas debido a la sensibilidad de esta al ruido, por ejemplo, introduciendo oscilaciones rápidas e indeseadas.

Es importante conocer que en la formulación del ILC, se suman velocidades con el producto entre las ganancias y el error, el cual es de posición. Por lo tanto, a la hora de elegir el valor de las ganancias hay que tener en cuenta que las variables implicadas en el controlador tienen magnitudes y órdenes de magnitud diferentes.

3.5. Simulación teórica

Se han realizado simulaciones por ordenador para analizar el comportamiento del supuesto planteado mediante las trayectorias expuestas en el apartado 2.2.2. Para ello, como punto de partida y teniendo en cuenta los criterios de estabilidad, se han utilizado las ganancias adecuadas a cada formulación. También se supone que el controlador en estas simulaciones no tiene ningún tipo de saturación.

3.5.1. Trayectoria helicoidal

Para la trayectoria helicoidal mostrada en la figura 2.5a los resultados obtenidos han sido los siguientes:

3.5.1.1. Formulación A: error en posición

Para la simulación de la primera trayectoria, con el fin de garantizar la estabilidad, se han elegido las ganancias mostradas en la tabla 3.5.

Ganancias proporcionales		**Ganancias derivativas**	
k_p	0,01	k_d	0,001

Tabla 3.5: Ganancias para la simulación por ordenador.

$e_{inicial}$ **(m)**		e_{50} **(m)**		e_{100} **(m)**	
RMSE	0,4553	RMSE	0,1697	RMSE	0,1082
STD	0,1886	STD	0,0596	STD	0,0547
$e_{inicial}$ **(rad)**		e_{50} **(rad)**		e_{100} **(rad)**	
RMSE	3,3959	RMSE	0,0838	RMSE	0,0487
STD	3,0389	STD	0,0673	STD	0,0433

Tabla 3.6: Errores cometidos en la trayectoria helicoidal implementando el ILC. Formulación A: error en posición.

Como se puede observar en la tabla 3.6, la mejora es notable. Respecto al error inicial tras 100 iteraciones se consigue una mejora del 76,28 % en posición y una mejora del 98,57 % en orientación. Cabe destacar que con 50 iteraciones ya se ha conseguido una mejora del 62,73 % y del 97,53 % respectivamente, por lo que si lo que se prioriza es la orientación, con 50 iteraciones ya se tendría una muy buena mejora. Se puede apreciar en la figura 3.4 que tras 100 iteraciones el algoritmo mejora especialmente la primera parte de la trayectoria.

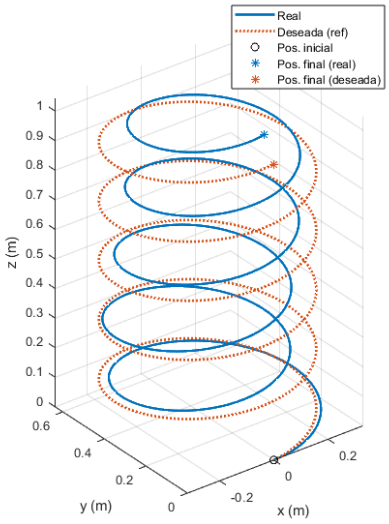


Figura 3.4: Trayectoria helicoidal tras 100 iteraciones. Formulación A: error en posición.

3.5.1.2. Formulación B: añadiendo instantes de tiempo

En esta formulación se ha elegido una k_p de 0,01 en todos los ejes para darle el mismo peso a los tres instantes temporales de los errores que se tienen en cuenta.

$e_{inicial}$ **(m)**		e_{50} **(m)**		e_{100} **(m)**	
RMSE	0,4553	RMSE	0,1028	RMSE	0,1370
STD	0,1886	STD	0,0443	STD	0,0490
$e_{inicial}$ **(rad)**		e_{50} **(rad)**		e_{100} **(rad)**	
RMSE	3,3959	RMSE	0,0188	RMSE	0,052
STD	3,0389	STD	0,0168	STD	0,0048

Tabla 3.7: Errores cometidos en la trayectoria helicoidal implementando el ILC. Formulación B: añadiendo instantes de tiempo.

En este caso, se consigue mejor aproximación a la trayectoria deseada con 50 iteraciones que con 100 (Tabla 3.7). Un motivo puede ser que como se le están añadiendo tres ganancias proporcionales, es probable que hayan sido demasiado elevadas y haya provocado que el sistema se vuelva inestable haciendo que a mayor número de iteraciones, el error no vaya disminuyendo. Aún así, se consigue una mejora del 69,91 % a las 100 iteraciones.

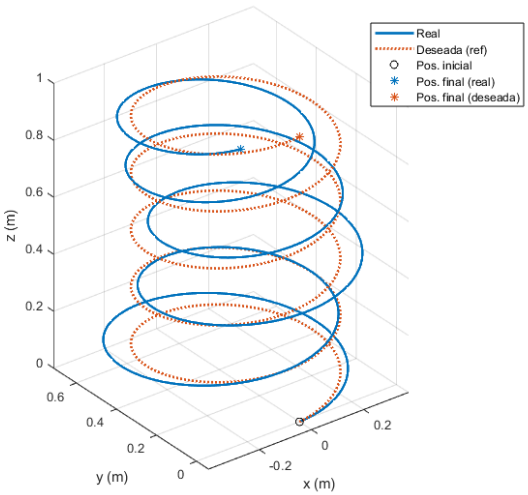


Figura 3.5: Trayectoria helicoidal tras 100 iteraciones. Formulación B: añadiendo instantes de tiempo.

3.5.1.3. Formulación C: error en velocidades locales

Para realizar la simulación en este apartado, se realizará con las ganancias mostradas en la tabla 3.8. En este caso, pueden ser levemente superiores ya que la corrección de la velocidad es el producto de la ganancia por el error de velocidad en el eje correspondiente, por lo que con las ganancias de dicha tabla, ya nos garantizamos que tenga un orden de magnitud menor,

como se ha explicado en el apartado de estabilidad.

Ganancias proporcionales		**Ganancias derivativas**	
k_p	0,1	k_d	0,01

Tabla 3.8: Ganancias para la simulación por ordenador.

$e_{inicial}$ **(m)**		e_{50} **(m)**		e_{100} **(m)**	
RMSE	0,4553	RMSE	0,0259	RMSE	0,0210
STD	0,1886	STD	0,0039	STD	0,0046
$e_{inicial}$ **(rad)**		e_{50} **(rad)**		e_{100} **(rad)**	
RMSE	3,3959	RMSE	0,0051	RMSE	0,039
STD	3,0389	STD	$9{,}2497 * 10^{-4}$	STD	$3{,}9849 * 10^{-4}$

Tabla 3.9: Errores cometidos en la trayectoria helicoidal implementando el ILC. Formulación C: error en velocidades locales.

Sin duda, es la formulación más efectiva consiguiendo a las 100 iteraciones una mejora del 95,39 %. También mejora la orientación hasta en un 98,85 %. Aunque hay que tener en cuenta que, en orientación, de la iteración número 50 a la 100 solo hay una mejora del 1.08 %, por lo que habría que valorar cuál es la precisión que se busca. Se pueden ver los errores en la tabla 3.9 y el resultado en la figura 3.6.

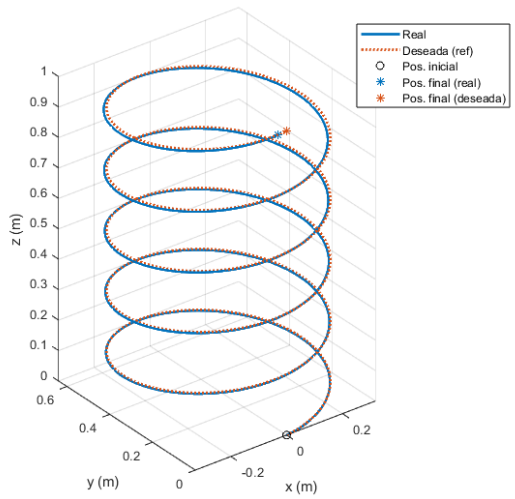


Figura 3.6: Trayectoria helicoidal tras 100 iteraciones. Formulación C: error en velocidades locales.

3.5.2. Trayectoria cuadrada

En vista de los resultados en el apartado anterior, se evalúa la trayectoria cuadrada (Figura 2.5b) mediante el método más preciso: Formulación C: error en velocidades locales. Para ello, se han utilizado las mismas ganancias que en la tabla 3.8. El resultado (Figura 3.7)

es una mejora del 80,11 % en posición y del 95,92 % en orientación. Sin embargo, la trayectoria no mejora en las siguientes 50 iteraciones. Se ha realizado un análisis más completo de esta situación en el apartado 3.5.4.

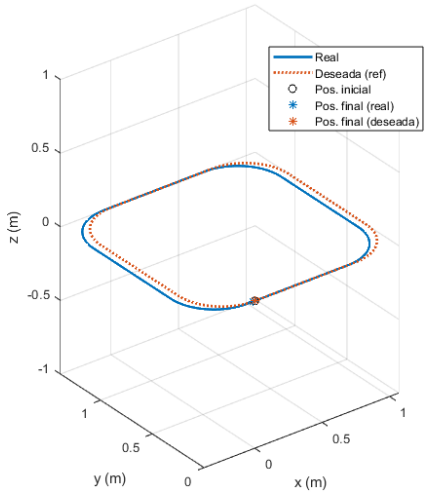


Figura 3.7: Trayectoria cuadrada tras 100 iteraciones. Formulación C: error en velocidades locales.

$e_{inicial}$ **(m)**		e_{50} **(m)**		e_{100} **(m)**	
RMSE	0,3147	RMSE	0,0626	RMSE	0,0637
STD	0,1461	STD	0,0310	STD	0,0324
$e_{inicial}$ **(rad)**		e_{50} **(rad)**		e_{100} **(rad)**	
RMSE	0,191	RMSE	0,0078	RMSE	0,0081
STD	0,1083	STD	0,0095	STD	0,0099

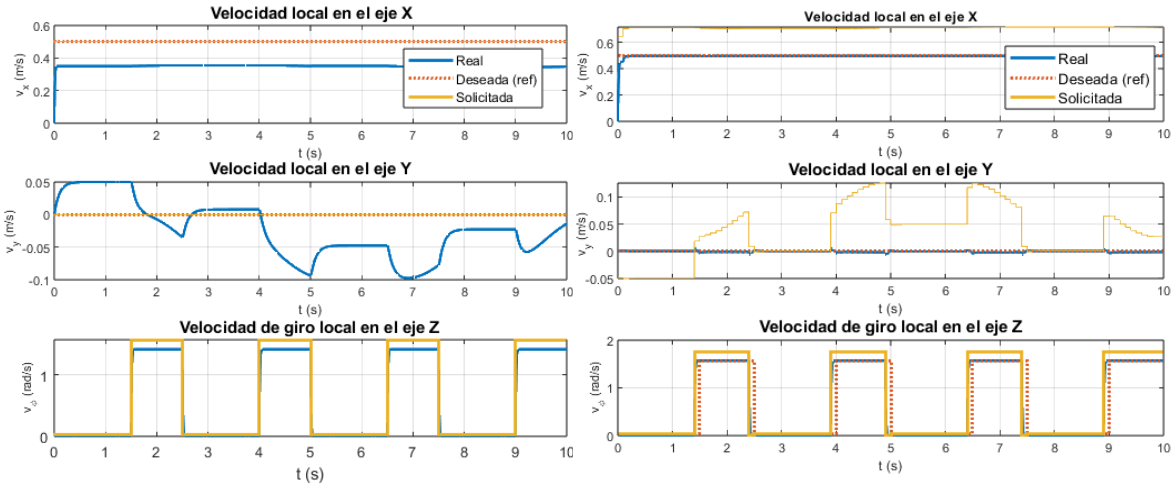
Tabla 3.10: Errores cometidos en la trayectoria cuadrada implementando el ILC. Formulación C: error en velocidades locales.

La figura 3.8 es muy ilustrativa del comportamiento del algoritmo. Donde mejor se aprecia es en la velocidad local en el eje y. La velocidad deseada en dicho eje es cero, sin embargo se puede ver que la velocidad real que lleva el dron (línea azul) debido al controlador es diferente. Para corregirlo, tras las 50 iteraciones, la velocidad solicitada pasa a corregir los errores (línea amarilla).

3.5.3. Trayectoria reloj de arena

$e_{inicial}$ **(m)**		e_{50} **(m)**		e_{100} **(m)**	
RMSE	0,8990	RMSE	0,0657	RMSE	0,0580
STD	0,3123	STD	0,0223	STD	0,0229

Tabla 3.11: Errores cometidos en la trayectoria reloj de arena implementando el ILC. Formulación C: error en velocidades locales.



(a) Trayectoria inicial (b) Trayectoria tras 50 iteraciones

Figura 3.8: Comparativa velocidades enviadas para la trayectoria cuadrada.

De la misma forma que la trayectoria cuadrada, se evalúa la trayectoria del reloj de arena (Figura 2.5c) con las mismas ganancias que en la tabla 3.8. En esta trayectoria no se tiene en cuenta la orientación porque no afecta a la misma. Tras 100 iteraciones se mejora la trayectoria en un 93,55 %, aunque de nuevo, con 50 iteraciones ya se ha producido una mejora del 92,69 %. La desviación media se mantiene sin grandes variaciones de la iteración 50 a la 100, habiendo mejorado en un 92,86 % desde la inicial. Los datos concretos se pueden encontrar en la tabla 3.11. Los resultados de las iteraciones se muestran en la siguiente figura:

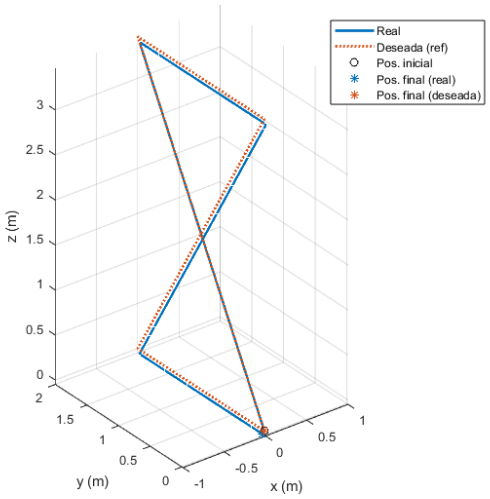
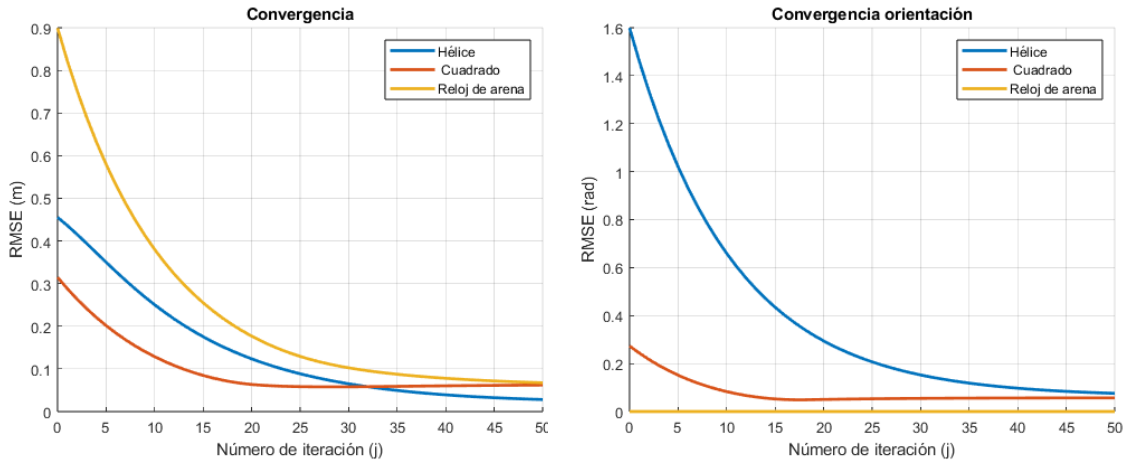


Figura 3.9: Trayectoria reloj de arena tras 100 iteraciones. Formulación C: error en velocidades locales.

3.5.4. Convergencia

En este apartado se expone la convergencia de la Formulación C: error en velocidades locales para las tres trayectorias planteadas. Es decir, cómo evoluciona el error cuadrático medio con el paso de las iteraciones, tanto de posición como de orientación.



(a) Posición (b) Orientación

Figura 3.10: Convergencia del error con el paso de las iteraciones.

Como se puede observar en la figura 3.10, tanto en la trayectoria del reloj de arena como en la hélice, el error tiende a cero conforme el número de iteraciones va aumentando. Sin embargo, en la trayectoria cuadrada, esto no es así y se puede observar como una vez llega a unas 20 iteraciones el error se mantiene casi constante. Esto podría deberse a la tipología de la trayectoria y a las limitaciones de los controladores para alcanzar el supuesto ideal. En cuanto al error de posición ocurre de manera similar, a partir de la iteración 20 deja de mejorar e incluso empeora levemente, aunque se estabiliza el error. En la trayectoria helicoidal continúa mejorando y el error en orientación tiende a cero. No se especifica el error de orientación de la trayectoria del reloj de arena ya que no implica ningún tipo de giro y es cero en todos los instantes temporales.

Las diferencias de rendimiento entre las trayectorias se debe a cómo están compuestas, mientras que en la hélice se le pide unas velocidades constantes, en las otras dos trayectorias hay un cambio de velocidades según el instante del vuelo. Esto provoca que el controlador tenga que cambiar la señal enviada e introduce el fallo de la constante temporal del controlador en cada cambio, que cuanto mayor es, más lento es el sistema y por lo tanto más difícil de controlar y seguir la trayectoria ideal.

3.6. Experimentación real

Se ha realizado la validación experimental del algoritmo de Iterative Learning Control presentado en este capítulo. Para ello se ha elegido la mejor formulación vista en las

simulaciones por ordenador, la que utiliza el error en velocidades en referencia local del dron. Se ha simulado la trayectoria helicoidal con $v_x = 0{,}3$ m/s, $v_z = 0{,}15$ m/s y $v_\phi = \frac{\pi}{4}$ rad/s. Se puede ver una imagen de la zona de vuelo y el dron durante la experimentación en la figura 3.11.

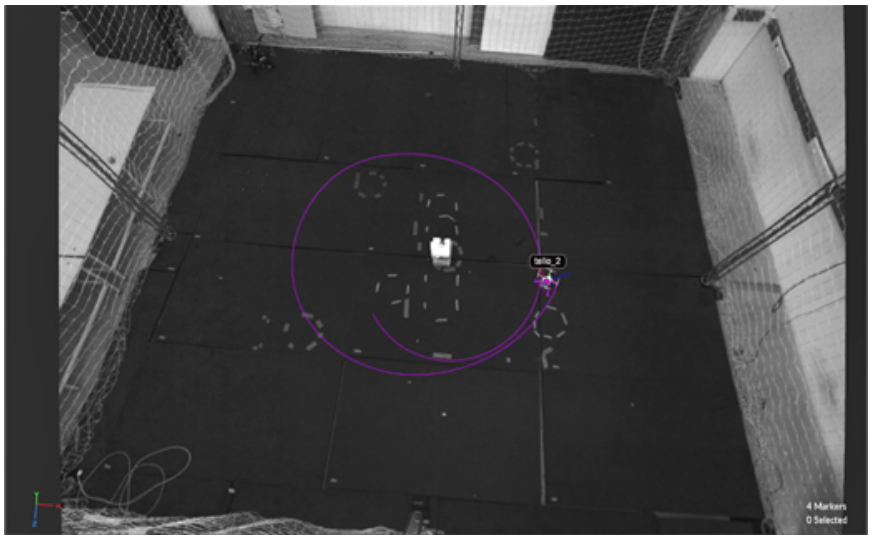


Figura 3.11: Zona de vuelo del OptiTrack durante la experimentación.

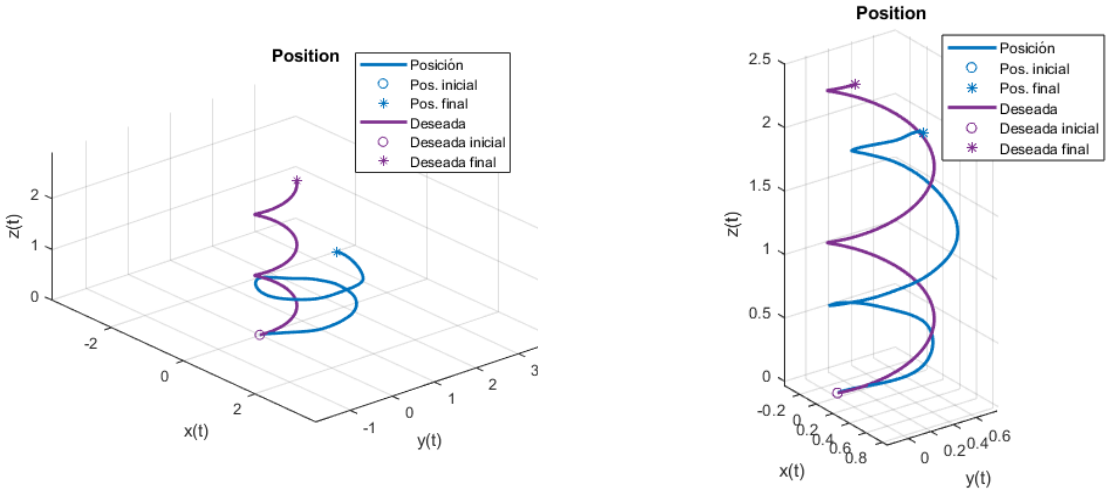
En primer lugar, con la estimación del controlador del dron calculado en el anexo C se ha simulado el supuesto para predecir el comportamiento del dron ante el control. Los resultados de la simulación se pueden observar en el anexo D. En dicha simulación y para la posterior experimentación, el algoritmo ILC se ha implementado utilizando las siguientes ganancias:

Ganancias proporcionales		**Ganancias derivativas**	
k_{px}	0,1	k_{dx}	0,01
k_{py}	0,1	k_{dy}	0,01
k_{pz}	0,2	k_{dz}	0,01
$k_{p\phi}$	0,2	$k_{d\phi}$	0,01

Tabla 3.12: Ganancias para la simulación de la estimación del dron Tello EDU.

Tras la simulación por ordenador para afianzar que la elección de las ganancias hace el sistema estable y es aplicable al caso del Tello EDU, se puede concluir que el algoritmo desarrollado corrige el error cometido por el dron Tello EDU con las ganancias establecidas. Tras 10 iteraciones, ya muestra una mejora significativa (mejorando en un 68,42 % en posición y 96,49 % en orientación) y tras 50, el RMSE es de 0,026 m y 0,0018 rad, lo que supone una mejora del 98.53 % y 99.94 % respectivamente (Tabla D.1). Por lo tanto, se procede a su implementación.

En la experimentación, cuando se le solicitan los valores de referencia, el dron no es capaz de replicar la trayectoria deseada (Figura 3.12, en la trayectoria inicial). En el anexo G se encuentra el error de posición separado en ejes. En la figura 3.13, se pueden observar las



(a) Trayectoria inicial (b) Trayectoria tras 20 iteraciones

Figura 3.12: Validación experimental. Trayectoria helicoidal.

velocidades que se le solicitan al dron en el primer vuelo, las cuales son iguales a las deseadas. Se puede observar que el controlador del dron tiene fallos de ganancia, especialmente en el eje z, y unas constantes temporales elevadas, esto quiere decir que cada vez que se le solicita una velocidad diferente, este tarda un tiempo importante en alcanzarla, por lo que si en menos de ese tiempo se le solicita una diferente, el dron aún no ha alcanzado el valor deseado y produce error en la trayectoria. Con el paso de las iteraciones, el algoritmo ha ido corrigiendo la velocidad enviada al dron para mejorar los errores que va cometiendo. Tras 20 iteraciones, el control ha sido capaz de mejorar la trayectoria a la mostrada en la figura 3.12 (b) solicitando las velocidades observadas en la figura 3.14. La corrección ha sido capaz de corregir con gran precisión la orientación del dron y ha conseguido buenas mejoras de posición.

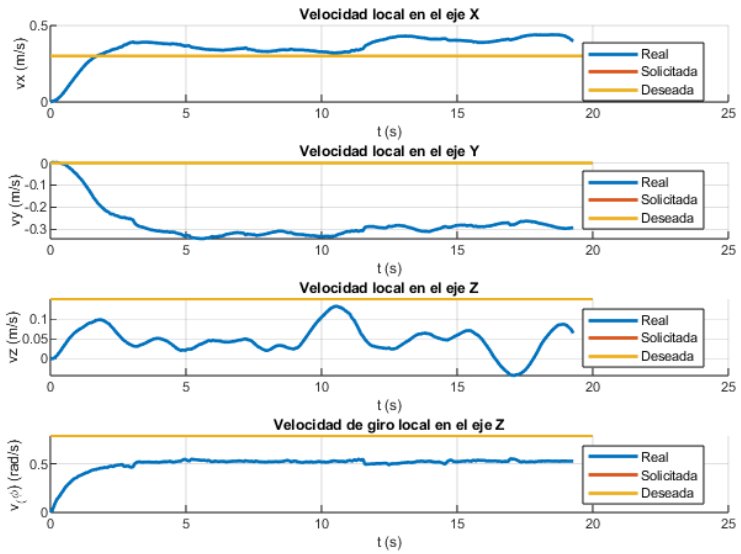


Figura 3.13: Velocidades en el vuelo inicial.

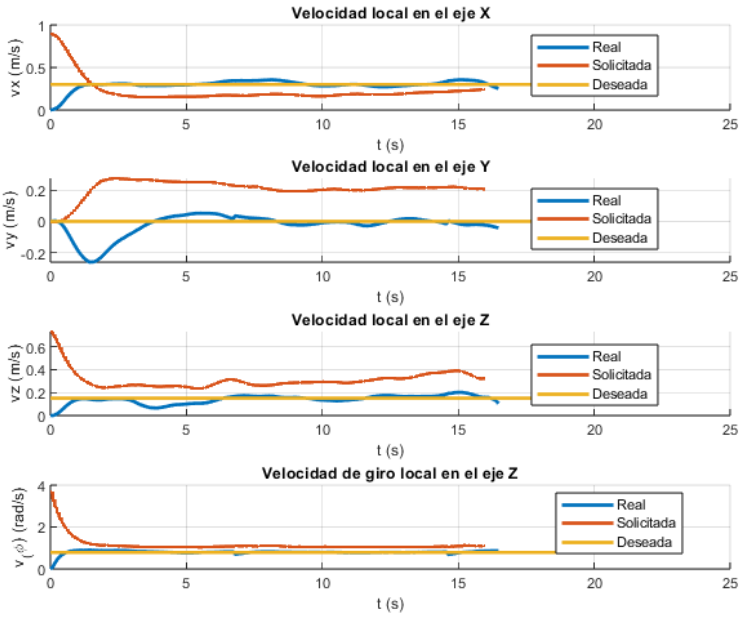
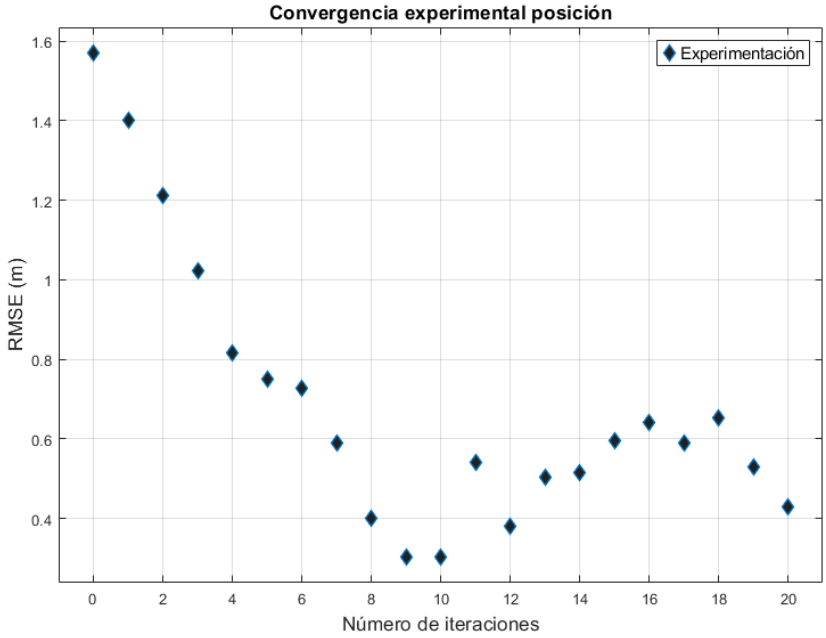


Figura 3.14: Velocidades tras realizar 20 iteraciones.

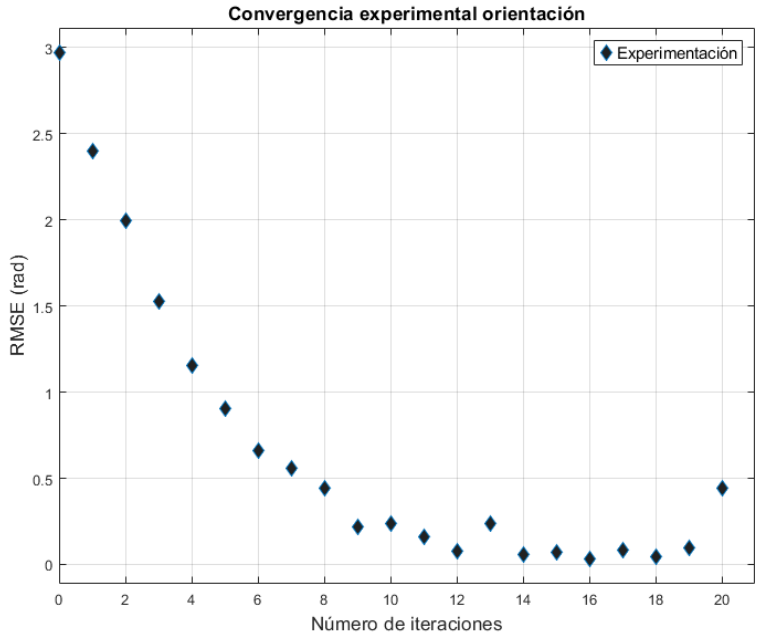
Con el fin de evaluar la efectividad del algoritmo se ha analizado la convergencia del error en las pruebas experimentales. Para ello, mediante los errores de posición y orientación obtenidos durante las simulaciones, se ha calculado la raíz del error cuadrático medio (RMSE) de cada iteración. Además, se han aproximado los resultados mediante una regresión. Se pueden observar los resultados en las gráficas de la figura 3.15. Además, en el anexo G se encuentran otras tablas de interés para entender cómo ha afectado el algoritmo con el paso de las iteraciones.

En resumen, tras la experimentación se pueden extraer las siguientes conclusiones:

- El algoritmo mejora en un $85,00\,\%$ la orientación en la trayectoria tras 20 iteraciones.
- El algoritmo mejora en un $72,66\,\%$ la posición en la trayectoria tras 20 iteraciones.
- El controlador en el eje z no consigue llevar una velocidad constante.
- El sistema introduce una velocidad en el eje y de origen desconocido que el algoritmo no es capaz de compensar hasta pasados 3 segundos, lo que introduce un error en el sistema que hace que el error converja a $0,5\ m$ aproximadamente.
- La constante temporal del eje x produce un error en el eje x que arrastra al resto de la trayectoria aunque eventualmente sea capaz de alcanzar la velocidad deseada.



(a) Error experimental en posición.



(b) Error experimental en orientación.

Figura 3.15: Convergencia del error experimental con el paso de las iteraciones ajustado mediante una regresión.

4. Optimización

4.1. Introducción

En los resultados del capítulo anterior, se ha observado cómo el algoritmo es capaz de mejorar la trayectoria del dron de manera significativa. En favor de la estabilidad del sistema, se han realizado las pruebas con una ganancia baja, lo que provoca una convergencia más lenta del error. Con el fin de mejorar este aspecto, se propone encontrar una forma de optimizar la elección de las ganancias para una convergencia más rápida. Con este fin, dentro de las opciones que ofrece Matlab, se han valorado diferentes funciones:

- ***fminunc.*** Optimización sin restricciones y recomendado para problemas de dimensión baja-media (Hasta 10 variables a optimizar).
- ***fmincon.*** Permite restricciones, también debe ser diferenciable, permite una dimensión de variables a optimizar mayor que *fminunc*, hasta unas 100.
- ***fminsearch.*** No permite restricciones y no necesita ser diferenciable. Para una dimensión entre 5 y 10 variables a optimizar.
- ***patternsearch.*** Admite restricciones, es un algoritmo de búsqueda directa y permite una dimensión de entre 100 y 1000 variables.

El supuesto que debemos optimizar no tiene restricciones y se debe optimizar la elección de 8 ganancias. Por ello, se ha decidido escoger *fminunc*, también por ser la más rápida dentro de las opciones. Sin embargo, el éxito de la optimización radica en elegir bien la función de coste que evaluará el algoritmo de optimización. En el siguiente apartado se encuentran las funciones desarrolladas.

4.2. Funciones de coste

Con el fin de definir el error en cada punto, se utiliza el error euclidiano (Ecuación 4.1) para el error de posición. Buscando evaluar cómo de óptima es una iteración, se hace la media de los errores euclidianos en cada instante de tiempo, quedando el error medio en la

enésima iteración (Ecuación 4.2). En orientación, se utiliza la raíz del error cuadrático medio (Ecuación 4.3).

$$e_k^{(n)} = \sqrt{\left(e_{x,k}^{(n)}\right)^2 + \left(e_{y,k}^{(n)}\right)^2 + \left(e_{z,k}^{(n)}\right)^2} \tag{4.1}$$

$$\bar{e}^{(n)} = \frac{1}{K}\sum_{k=1}^{K} e_k^{(n)} \tag{4.2}$$

$$\mathrm{RMSE}_{\phi}^{(n)} = \sqrt{\frac{1}{K}\sum_{k=1}^{K}\left(e_{\phi,k}^{(n)}\right)^2} \tag{4.3}$$

Donde k es el instante de tiempo, K el número total de instantes de tiempo en la iteración y n el número de iteración.

4.2.1. I. Error en la enésima iteración

La primera función de coste busca minimizar el error, tanto en posición como en orientación en la enésima iteración. Para ello, sustituyendo la ecuación 4.1 en la ecuación 4.2 y sumándole la ecuación 4.3 para incluir la orientación, queda la función mostrada en la ecuación 4.4. Se busca optimizar dicha función porque evalúa el error en la enésima iteración, lo que, encontrando los valores adecuados, busca el menor error posible tras el número de iteraciones deseado.

$$J_1^{(n)} = \frac{1}{K}\sum_{k=1}^{K}\sqrt{\left(e_{x,k}^{(n)}\right)^2 + \left(e_{y,k}^{(n)}\right)^2 + \left(e_{z,k}^{(n)}\right)^2} + \sqrt{\frac{1}{K}\sum_{k=1}^{K}\left(e_{\phi,k}^{(n)}\right)^2} \tag{4.4}$$

Vistos los resultados de la convergencia del capítulo anterior, se intenta minimizar el error para $n = 10$ iteraciones.

4.2.2. II. Suma ponderada del error en iteraciones concretas

La segunda función de coste almacena los errores cometidos en las iteraciones que sean de mayor interés y a través de unos parámetros de ponderación, se le otorga más o menos peso a unas u otras. En este caso, las iteraciones que se buscan minimizar son las correspondientes al primer, segundo, tercer cuartil (Q_1, Q_2, Q_3) y al último valor (N). Los parámetros asignarán más peso a las últimas iteraciones. Con ello, además de optimizar la elección de las

ganancias, se busca asegurar que el sistema no pierde estabilidad, haciendo que el error vaya disminuyendo con el paso de las iteraciones. Por lo tanto, la ecuación de la segunda función de coste es la siguiente:

$$J_2 = a\ J_1^{(Q_1)} + b\ J_1^{(Q_2)} + c\ J_1^{(Q_3)} + d\ J_1^{(N)} \tag{4.5}$$

Siendo $J_1^{(n)}$ la ecuación 4.4 aplicada a las iteraciones especificadas y N el número total de iteraciones. Los parámetros a, b, c y d son las ponderaciones utilizadas. Nuevamente, se simulará para $N = 10$. En los casos como este, donde los cuartiles tengan decimales, se utilizará la iteración más próxima. En este caso en concreto: $Q_1 = 3$, $Q_2 = 5$, $Q_3 = 8$.

4.2.3. III. Suma ponderada del error en ciertos instantes de tiempo de la enésima iteración

La tercera función de coste calcula los errores específicos de los instantes de tiempo claves en la trayectoria en la iteración n. En este caso, como se busca reducir el error globalmente, se escogen los valores de tiempo correspondientes, de nuevo, al primer, segundo, tercer cuartil y al último valor (q_1, q_2, q_3, K). Con esta función de coste se pretende que no haya grandes errores de posición en momentos críticos. Para definir esta función de coste se ha aplicado la ecuación 4.1 en los instantes de tiempo mencionados sumándole el error absoluto en orientación, quedando de tal manera:

$$J_3^{(n)} = a\ (e_{q_1}^{(n)} + |e_{\phi,q_1}^{(n)}|) + b\ (e_{q_2}^{(n)} + |e_{\phi,q_2}^{(n)}|) + c\ (e_{q_3}^{(n)} + |e_{\phi,q_3}^{(n)}|) + d\ (e_K^{(n)} + |e_{\phi,K}^{(n)}|) \tag{4.6}$$

Donde a, b, c y d son los parámetros de ponderación para darle mayor valor a un punto en concreto.

4.3. Simulación y resultados

Se han simulado las tres funciones de coste expuestas previamente bajo las mismas condiciones explicadas en el capítulo de Iterative Learning Control en la tabla 3.1 y los parámetros de las funciones de coste han sido los mostrados en la tabla 4.1 para primar los valores tanto de las últimas iteraciones en la función de coste II como de los últimos instantes de tiempo en la función de coste III. Además se ha puesto como condición que las ganancias proporcionales (k_p) deben ser positivas.

Parámetros	
a	0,10
b	0,15
c	0,20
d	0,55

Tabla 4.1: Parámetros de ponderación de las funciones de coste.

Tras la simulación de la función de coste I (4.2.1), las ganancias obtenidas son las mostradas en la tabla 4.2. Tras la simulación de la función II (4.2.2), se han obtenido las ganancias de la tabla 4.3 y las de la tabla 4.4 se han obtenido en la simulación de la III (4.2.3). Con dichas ganancias calculadas, se han realizado los vuelos con el simulador de Matlab. Se han plasmado los resultados del RSME obtenido en la simulación en la figura 4.1.

Por lo explicado en el capítulo anterior en el apartado de estabilidad, cabe destacar que las ganancias de la tabla 4.4, k_{px} y k_{pz} son elevadas. Especialmente dados los valores de la velocidad, que son del orden de $0,15 - 0,5\ m/s$, por lo que se necesitará un valor inicial del error pequeño (Del orden de $0,1\ m$) para que no se produzcan cambios muy bruscos de una iteración a otra. Es por ello que es posible que hagan alcanzar en menos iteraciones el valor deseado pero también pueden provocar que en el sistema haya muchas sobreoscilaciones en las primeras iteraciones o incluso que el sistema se vuelva inestable y el algoritmo no funcione correctamente.

Ganancias proporcionales		**Ganancias derivativas**	
k_{px}	0,5193	k_{dx}	$8,7466 * 10^{-4}$
k_{py}	0,4833	k_{dy}	0,0257
k_{pz}	0,5295	k_{dz}	0,0061
$k_{p\phi}$	1,3834	$k_{d\phi}$	$4,8375 * 10^{-15}$

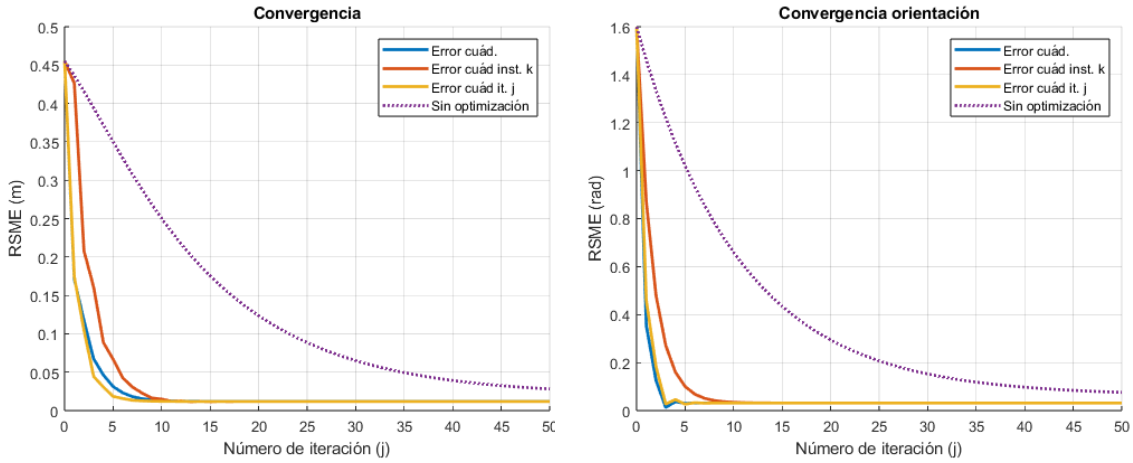
Tabla 4.2: Ganancias obtenidas en la simulación con la función de coste I.

Ganancias proporcionales		**Ganancias derivativas**	
k_{px}	0,5903	k_{dx}	$-2,4628 * 10^{-4}$
k_{py}	0,6196	k_{dy}	0,0491
k_{pz}	0,6864	k_{dz}	0,0092
$k_{p\phi}$	1,4578	$k_{d\phi}$	$2,7472 * 10^{-15}$

Tabla 4.3: Ganancias obtenidas en la simulación con la función de coste II.

Ganancias proporcionales		**Ganancias derivativas**	
k_{px}	2,2662	k_{dx}	$-3,0338 * 10^{-4}$
k_{py}	0,3640	k_{dy}	$-0,0015$
k_{pz}	2,4198	k_{dz}	0,0513
$k_{p\phi}$	0,5163	$k_{d\phi}$	$1,3626 * 10^{-5}$

Tabla 4.4: Ganancias obtenidas en la simulación con la función de coste III.



(a) Posición (b) Orientación

Figura 4.1: Convergencia del error con el paso de las iteraciones aplicando las ganancias obtenidas en la optimización.

En la figura 4.1 la línea azul son los resultados de la simulación con las ganancias obtenidas con la función de coste I, la amarilla con la función II y la línea roja con la III. Todas ellas están comparadas con la morada, que han sido los resultados obtenidos en el capítulo anterior.

Las ganancias calculadas hacen que la raíz del error cuadrático medio con el paso de las iteraciones evolucione de la forma mostrada en la figura 4.1. Como se puede observar, las tres opciones mejoran significativamente la convergencia del error a las 10 iteraciones. Entre ellas la que antes converge es la función de coste II. Suma ponderada del error en iteraciones concretas. En error permanente también es la que presenta el mejor valor $RSME^{II}_{perm} = 1{,}17652 * 10^{-2}\ m$ frente a $RSME^{II}_{perm} = 1{,}1812 * 10^{-2}\ m$ de la función de coste I y $RSME^{III}_{perm} = 1{,}18052 * 10^{-2}\ m$ de la III. En orientación la función I presenta la convergencia más rápida, pero en error permanente todos los métodos convergen para presentar un valor de $RSME^{ori}_{perm} = 3{,}10906 * 10^{-2}\ rad$, por lo que, teniendo en cuenta todos los datos, la mejor opción es la explicada en el apartado 4.2.2.

Para 10 iteraciones y el sistema presentado, las ganancias mostradas en la tabla 4.3 son las óptimas. Mediante el estudio de un dron real y su comportamiento, obteniendo un modelo aproximado, se podría predecir dicho sistema y ajustar las ganancias mediante simulación para conseguir una convergencia rápida. Cabe destacar que el hecho de aumentar las ganancias provoca en las primeras iteraciones unas sobreoscilaciones a tener en cuenta para los siguientes vuelos, ya que pueden darse unas condiciones en las que ese vuelo no se pueda permitir.

5. Conclusiones y trabajo futuro

5.1. Conclusiones

Este trabajo ha logrado demostrar que el método Iterative Learning Control (ILC) es efectivo para mejorar la precisión en la trayectoria de drones en tareas repetitivas, aprovechando datos de vuelos previos. En la fase de simulación, se utilizaron tres formulaciones del algoritmo para corregir los errores de trayectoria. Los resultados mostraron que, mientras que las formulaciones basadas en errores de posición lograban mejoras del 62.73 % y 69.91 % en la aproximación de la trayectoria tras 100 iteraciones, la formulación que empleaba el error en las velocidades consiguió una mejora superior al 95 % en tan solo 50 iteraciones. Además, todas las formulaciones mejoraron la orientación en más del 97 %, siendo la de error en velocidades la más eficiente en cuanto a rapidez y precisión.

A continuación, se pasó a la fase experimental, donde se realizaron vuelos con el algoritmo basado en el error de velocidades. Aunque se observó una mejora en la posición del 72.66 % y en la orientación del 80 % tras 20 iteraciones, la convergencia del error de posición resultó ser menos estable de lo esperado. Esto se atribuyó a factores propios del entorno real de vuelo, como la temperatura, el nivel de batería y la estabilización inicial, que afectan al rendimiento del dron. Este comportamiento resalta la necesidad de mejorar la robustez del controlador, lo que podría abordarse mediante técnicas complementarias en futuros estudios.

Finalmente, se optimizaron los parámetros del algoritmo mediante simulación en Matlab, evaluando tres funciones de coste diferentes. Todas las opciones mejoraron la velocidad de convergencia en más del 80 %, destacando la función de coste que minimizaba el error en las iteraciones $j = Q_1, Q_2, Q_3, N$, que presentó los mejores resultados en cuanto a rapidez de convergencia y un error en estado estacionario ligeramente más pequeño.

En conclusión, este trabajo ha demostrado que el método Iterative Learning Control puede ser eficaz para mejorar la precisión de vuelo de los drones en trayectorias repetitivas. A pesar de las dificultades encontradas en la implementación real, los resultados obtenidos son prometedores y ofrecen un gran potencial para aplicaciones en sectores como la vigilancia, el inventariado o la automatización industrial.

La figura 5.1 muestra la planificación que se ha llevado a cabo, la cual concuerda con la planteada al inicio del Trabajo de Fin de Grado.

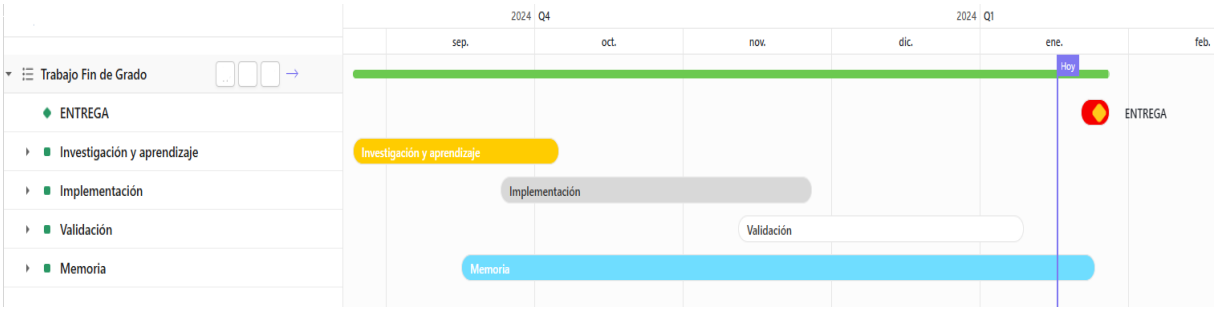


Figura 5.1: Diagrama de Gantt del Trabajo Fin de Grado.

5.2. Trabajo futuro

A pesar de que el trabajo ha cumplido con el propósito de mejorar las trayectorias de drones con tareas repetitivas, puede ser interesante continuar alguna línea de investigación de las explicadas a continuación.

- El simulador de vuelo ha permitido hacer una simulación previa para comprobar el comportamiento del algoritmo. Sin embargo, implementar más variables en el simulador de vuelo de los drones, como por ejemplo, ruido en las velocidades o saturación del controlador, puede representar vuelos más reales y ser utilizado para conseguir otro tipo de control.
- El dron Tello EDU cuenta con una cámara a la que se puede acceder para hacer reconocimiento de imágenes. Podría implementarse la mejora de trayectorias incluyendo el reconocimiento de imágenes.
- Mediante técnicas como una red neuronal, o de aprendizaje por refuerzo, podría buscarse un modelo aproximado del dron Tello EDU y, dada su poca capacidad de cómputo, utilizar el ILC para optimizar las ganancias y mejorar las trayectorias con la simulación del modelo y posterior valoración experimental

Bibliografía

[1] Seguridad, Acecho: *Sistemas de videovigilancia con drones*. https://www.acecho.es/sistemas-videovigilancia-drones/

[2] Prieto, A. ; Rodriguez, I. ; Rodas, J. ; Paiva, E. ; Gregor, R. ; Chaparro, E. ; Prieto-Araujo, E.: Image Processing Technique Applied to Electrical Substations Based on Drones With Thermal Vision for Predictive Maintenance. In: *2022 IEEE International Conference on Automation/XXV Congress of the Chilean Association of Automatic Control (ICA-ACCA)*, 2022, S. 1–6

[3] Mecalux: *Inventario con drones: ¿el futuro del control de stock?* https://www.mecalux.es/blog/inventario-con-drones

[4] OptiTrack: *OptiTrack*. https://optitrack.com/

[5] García, Patricia: Utilización de un sistema de captura de movimiento externo para el control del movimiento de equipos de drones. (2024), S. 1–61

[6] Kutzer, Michael: *OptiTrackToolbox*. https://es.mathworks.com/matlabcentral/fileexchange/55675-kutzer-optitracktoolbox

[7] damiafuentes, M4GNV5: *djitellopy 2.5.0*. https://pypi.org/project/djitellopy/#description

[8] Tech, Ryze: *Tello EDU*. https://www.ryzerobotics.com/tello-edu

[9] DJI: *Tello App*. https://www.dji.com/es/downloads/djiapp/tello

[10] Matlab: *MATLAB Support Package for Ryze Tello Drones*. https://es.mathworks.com/matlabcentral/fileexchange/74434-matlab-support-package-for-ryze-tello-drones?s_tid=FX_rc3_behav

[11] Arimoto, S. ; Kawamura, S. ; Miyazaki, F.: Bettering operation of dynamic systems by learning: A new control theory for servomechanism or mechatronics systems. In: *The 23rd IEEE Conference on Decision and Control*, 1984, S. 1064–1069

[12] Bristow, D.A. ; Tharayil, M. ; Alleyne, A.G.: A survey of iterative learning control. In: *IEEE Control Systems Magazine* 26 (2006), Nr. 3, S. 96–114. http://dx.doi.org/10.1109/MCS.2006.1636313. – DOI 10.1109/MCS.2006.1636313

[13] Buelta, Almudena ; Olivares, Alberto ; Staffetti, Ernesto ; Aftab, Waqas ; Mihaylova, Lyudmila: A Gaussian Process Iterative Learning Control for Aircraft Trajectory Tracking. In: *IEEE Transactions on Aerospace and Electronic Systems* 57 (2021), Nr. 6, S. 3962–3973. http://dx.doi.org/10.1109/TAES.2021.3098133. – DOI 10.1109/TAES.2021.3098133

[14] DJI: *Tello EDU SDK*. https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf

Lista de Figuras

Lista de Tablas

Anexos

A. Experimentación previa con el Dron Tello EDU

En este anexo se encuentra información sobre el vuelo del dron Tello EDU y las principales especificaciones técnicas. También se realizaron unas simples pruebas de vuelo para comprobar como era la recepción de información del dron a través del paquete MATLAB Support Package for Ryze Tello Drones [10].

TELLO EDU	**Especificaciones**
Peso	87 g
Velocidad máxima*	4 m/s
Tiempo de vuelo máx.	13 min

Tabla A.1: Principales especificaciones del dron Tello Edu

*Puede llegar a 8 m/s si se cambia la selección de vuelo predeterminado a vuelo FAST en la aplicación.

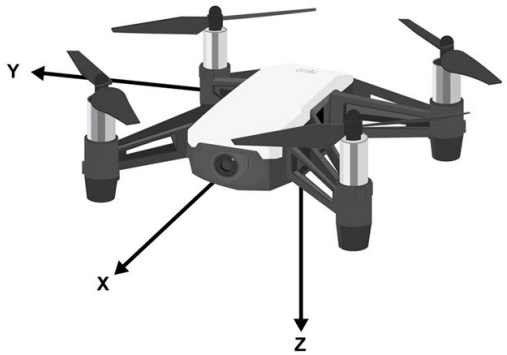


Figura A.1: Ejes dron Tello EDU de Ryze Tech en Matlab.
Imagen de `https://es.mathworks.com/help/matlab/supportpkg/move.html`

La velocidad del dron se expresa en m/s a lo largo de los ejes x, y y z. Dichos ejes están orientados respecto al marco inercial NED (Norte, Este, Abajo) como se puede ver en la figura A.1. El marco inercial es calculado al iniciar el dron, el cual se considera inicializado al establecerse una conexión exitosa entre el dron y Matlab mediante el comando *ryze()*, el cual crea un objeto en Matlab para interactuar con las funciones establecidas. Dicho paquete es muy útil para hacer vuelos simultáneos simples con numerosos drones.

Se realizaron unas pruebas para comprobar la fiabilidad del paquete y la calidad del dron para sacar datos internos desde su procesador. La trayectoria recorrida es una ida y vuelta sobre 1,5 metros, la velocidad solicitada fue de 0.5 m/s. Como se puede observar en las figuras A.2 y A.3 los controladores internos del dron presentan una constante de tiempo (τ) elevada, ello sumado a la baja precisión en velocidades que devuelve el dron, hace que esta medida no sea suficiente para un control efectivo.

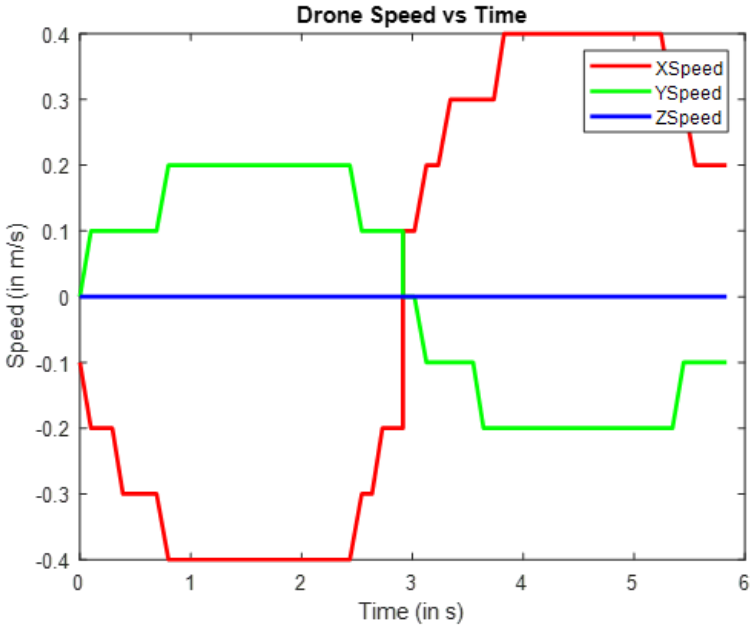


Figura A.2: Velocidad del dron en una trayectoria de ida y vuelta..

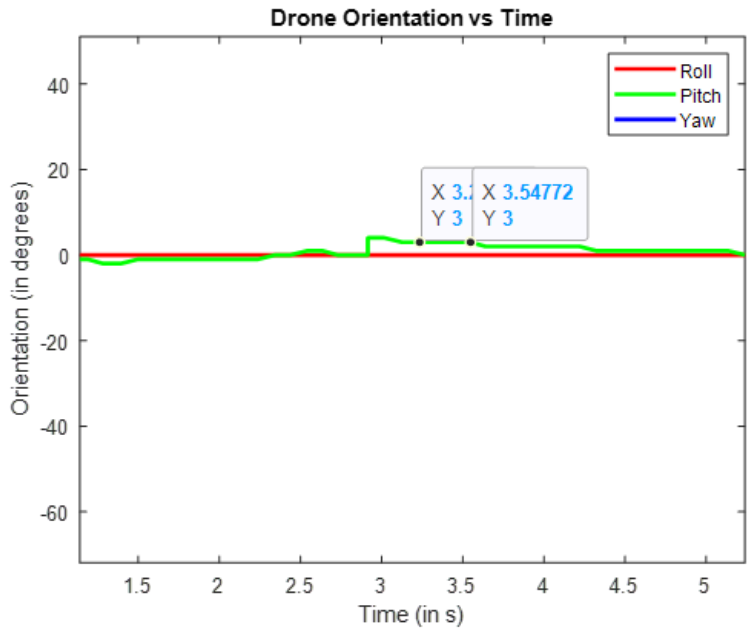


Figura A.3: Orientación del dron en una trayectoria de ida y vuelta.

Por tanto, las conclusiones extraídas de la familiarización con el dron son las siguientes:

- La frecuencia de salida de datos es insuficiente para poder realizar un control en bucle cerrado.
- El control de trayectorias del dron es demasiado a alto nivel. Solo permite precisar una velocidad para cada tramo de trayectoria. Mientras está ejecutando un comando, ignora cualquier otro que reciba hasta que acaba.
- Los controladores del dron presentan una constante de tiempo (τ) elevada.
- Se ha de trabajar en un control que envíe comandos al dron con una frecuencia de 10 Hz.
- Desde este punto se descarta la posibilidad de utilizar las velocidades internas del dron para el control. Se utiliza desde ahora el sistema de captura de movimiento externo Optitrack.

B. Comunicación con el dron Tello EDU mediante una conexión UDP en Matlab

En este anexo explica las conclusiones obtenidas tras desarrollar un código para establecer una conexión UDP con el dron para su control.

Una conexión User Data Protocol (UDP) es un tipo de comunicación que permite enviar datos de manera rápida sin establecer una conexión previa entre emisor y receptor. Es un protocolo en el que se prima la velocidad a la fiabilidad. Por esto mismo, no garantiza la entrega de paquetes, lo que lo convierte en un método poco efectivo, al menos sin una capa superior que implementen las funciones de verificación o control de errores.

Tras realizar las pruebas pertinentes se ha llegado a las siguientes conclusiones:

- La pérdida de información es demasiado grande, se pierden comandos enviados con frecuencia (Al menos un comando por vuelo) y recibidos (Una de cada dos veces no recibe información del vuelo). Lo que lo convierte en un método poco fiable para el control.
- Al igual que en el paquete de Matlab de los drones Tello[10], no permite el acceso a controlar la velocidad de manera directa.

C. Errores del controlador de vuelo del dron

En este anexo se analiza el funcionamiento del controlador interno del dron. Para ello, mediante las funciones explicadas en el anexo E y el código desarrollado en python para el envío de velocidades se le han solicitado al dron las velocidades máximas en cada grado de libertad. También se han aproximado las constantes temporales. Dichas constantes de definen como el tiempo que tarda el sistema en alcanzar el 63.2 % de su respuesta final. Tras la experimentación se han obtenido los siguientes resultados:

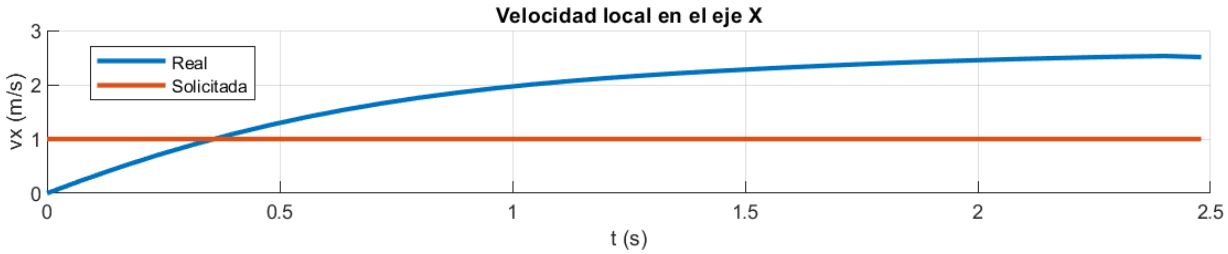


Figura C.1: Comportamiento del controlador en v_x.

Según la API (Application Programming Interface) del paquete de Python que contiene la función con la que se le solicitan velocidades al dron, deben ser valores entre -100 y 100 cm/s para la velocidad lineal y de entre -100 y 100 grados/s en el caso de la velocidad angular. La figura C.1 muestra como al solicitarle 100 cm/s en velocidad en el eje x, esta llega a alcanzar 240 cm/s. Lo mismo ocurre con la velocidad lineal en el eje y. La constante temporal en x e y es es $\tau = 0{,}6$.

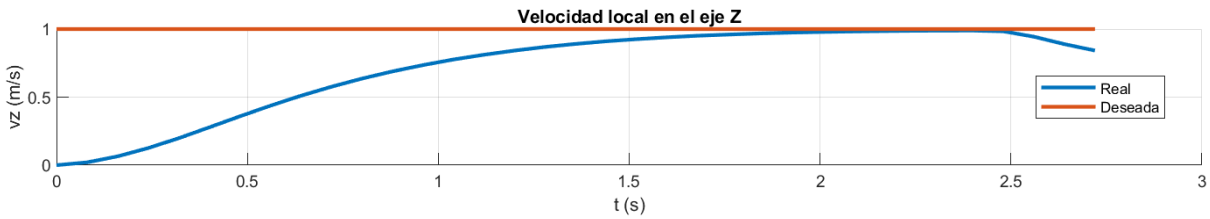


Figura C.2: Comportamiento del controlador en v_z.

En la velocidad en el eje z sí que se obtiene la velocidad esperada, sin embargo la constante temporal tiene un valor de $\tau_z = 0{,}8$, levemente peor que en los ejes x e y (Figura C.2). En

cuanto a la velocidad de giro, se alcanza en un 87 % la solicitada, con una $\tau_\phi = 1$.

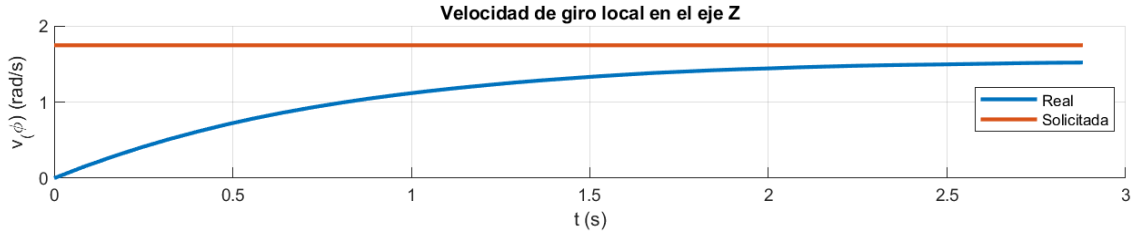


Figura C.3: Comportamiento del controlador en v_ϕ.

Cabe destacar que durante la experimentación se ha observado que si se le solicitan valores pequeños al controlador, este tiene dificultad para seguir las velocidades deseadas. Véase la velocidad local en el eje z de la figura 3.13. Esto significa que la ganancia del controlador no es constante y que varía dependiendo de la velocidad solicitada, así como la fiabilidad de su respuesta.

Con todos estos datos, se puede realizar una estimación del controlador real del dron, la cual será utilizada para predecir el comportamiento del algoritmo de control en el apartado de resultados.

Perturbaciones externas		**Ganancia del controlador**		**Constante temporal**	
P_x	0	k_x	2,4	τ_x	0,6
P_y	0	k_y	2,4	τ_y	0,6
P_z	0	k_z	0,33 − 1 [1]	τ_z	0,8
		k_ϕ	0,87	τ_ϕ	1

Tabla C.1: Estimación del controlador interno del dron Tello EDU

[1] A velocidades bajas, será más próximo a $k_y = 0{,}33$ mientras que a velocidades máximas alcanzará $k_y = 1$

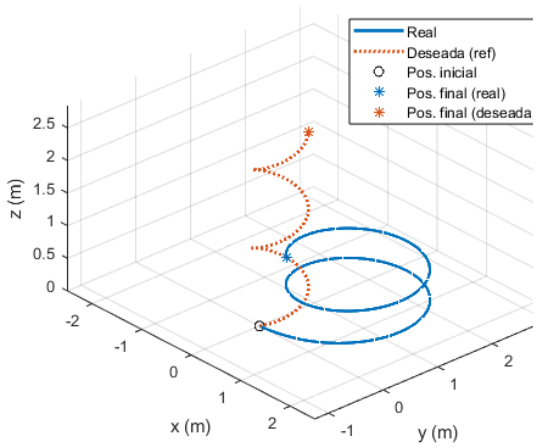
D. Simulación ILC con la estimación del controlador interno del dron Tello EDU.

En este anexo se explica la simulación realizada para predecir el comportamiento del dron ante el control. Para ello, se ha utilizado la estimación del controlador interno del dron calculada en el anexo C.

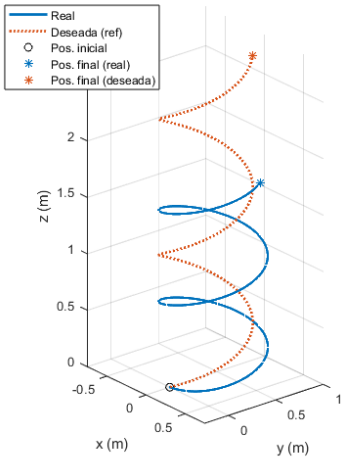
En dicha simulación se han utilizado las ganancias expresadas en la tabla 3.12. Se ha simulado la trayectoria helicoidal con $v_x = 0{,}3$, $v_z = 0{,}15$ y $v_\phi = \frac{\pi}{4}$. Los resultados de los errores cometidos en la simulación se pueden encontrar en la tabla D.1. Las evolución de las trayectorias se pueden observar en las figuras D.1. El comportamiento que se observa es el esperado en la validación experimental y por tanto, se espera una mejora de trayectoria.

$e_{inicial}$ **(m)**		e_{10} **(m)**		e_{50} **(m)**	
RMSE	1,7715	RMSE	0,5594	RMSE	0,0526
STD	0,7498	STD	0,2745	STD	0,0170
$e_{inicial}$ **(rad)**		e_{10} **(rad)**		e_{50} **(rad)**	
RMSE	3,005	RMSE	0,1055	RMSE	0,018
STD	1,9472	STD	0,0501	STD	$5{,}2002 * 10^{-4}$

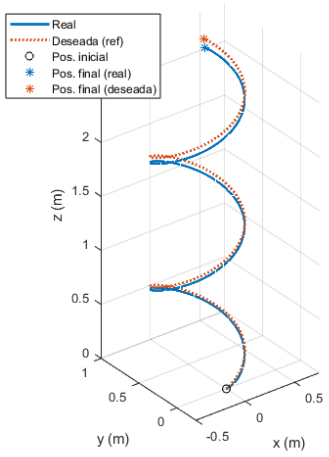
Tabla D.1: Errores en la simulación de la aproximación al dron Tello EDU.



(a) Trayectoria inicial

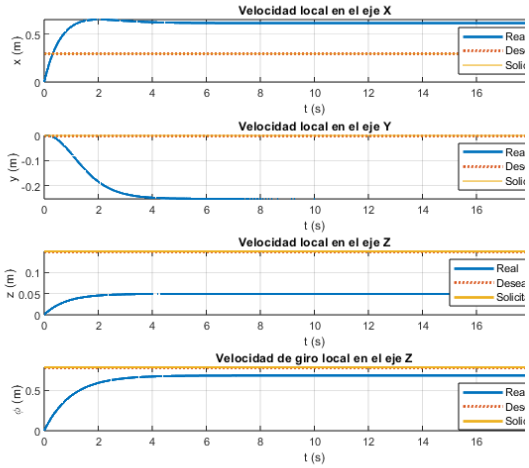


(b) Trayectoria tras 10 iteraciones

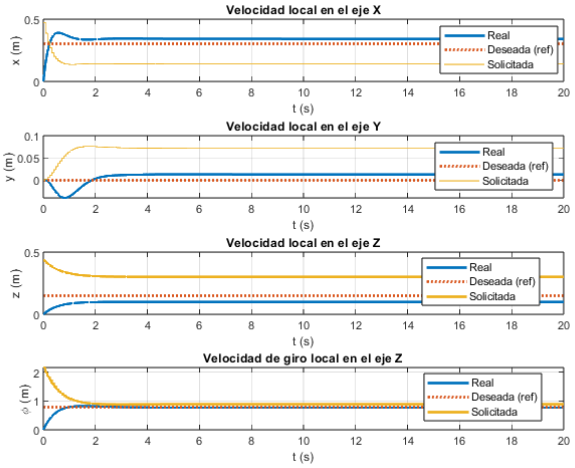


(c) Trayectoria tras 50 iteraciones

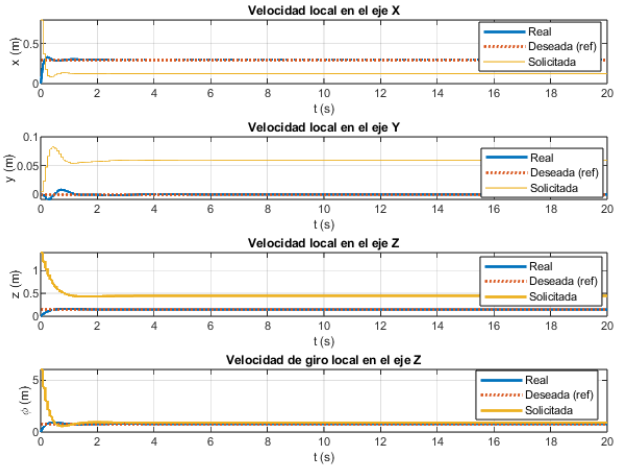
Figura D.1: Simulación dron Tello EDU. Trayectoria helicoidal.



(a) Velocidades iniciales.



(b) Velocidades tras 10 iteraciones.



(c) Velocidades tras 50 iteraciones.

E. Módulo Python DJITelloPY

El módulo de Python DJITelloPy[7] implementa las funciones especificadas en la SDK del Tello EDU[14] añadiendo capas superiores para asegurar que el envío y recepción de información se realiza con éxito. A continuación, se explican las funciones principales con las que se ha realizado el código python para enviar velocidades al dron.

- **connect()** Intenta establecer conexión con el dron, si no recibe respuesta tras 20 intentos comunica un fallo. Se establece con éxito cuando recibe un paquete con un 'ok' del dron.
- **takeoff()** Despega y estabiliza el dron, tras enviar el comando espera 20 segundos antes de continuar enviando otros comandos, ya el dron no procesa más comandos hasta que se ha estabilizado con éxito.
- **send_rc_control(vy,vx,vz,vphi)** Se solicitan valores de velocidad entre -100 y 100 cm/s en cada eje.
- **land()** Aterriza automáticamente. Este comando también se ejecuta automáticamente si a los 15 segundos el dron no ha recibido ningún tipo de comandos ni se le ha solicitado información.

De todas ellas, la función más importante es mediante la que se le mandan las velocidades al dron, send_rc_control(v_y,v_x,v_z,v_ϕ). Por defecto, los valores de la función, son enviados cada TIME_BTW_RC_CONTROL_COMMANDS = 0,001 segundos (1000 Hz). Sin embargo, el fabricante especifica que aunque el dron es capaz de recibir datos con una frecuencia de 30 HZ, es recomendable para un control más estable, utilizar una frecuencia de entre 10 y 20 Hz[8]. Si le mandamos al dron comandos con demasiada frecuencia ignorará dichos comandos hasta que sea capaz de procesar el anterior. En consecuencia, se ha escogido una frecuencia de 12,5 Hz para garantizar el correcto procesamiento de la información enviada a través de la función.

Sin embargo, la función cuenta con una diferencia respecto a la referencia del sistema global. Como se puede observar en la figura E.1, la componente angular es contraria a la

habitual, por lo tanto se le aplicará una corrección a la hora de mandar las velocidades al dron.

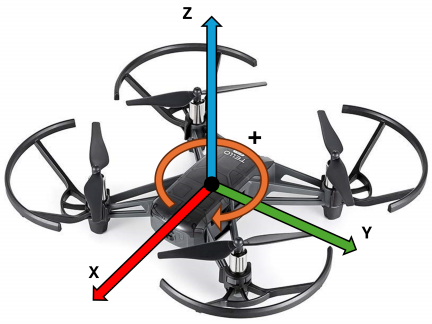


Figura E.1: Sistema de referencia del dron con el paquete DJITelloPY.

F. Filtrado de señales

En este anexo se explica el filtro que se ha utilizado para filtrar las señales recibidas por el OptiTrack. Para la realización del algoritmo de mejora y para conocer la posición y la velocidad en cada momento del dron Tello EDU, era necesario filtrar la señal recibida. Dicha señal eran valores de orientación y posición del dron en cada instante.

El primer filtrado de los datos ha consistido en eliminar datos repetidos o instantes de tiempo vacíos que proporcionaba el OptiTrack. Una vez se tenían filtrados, aprovechando la importancia de la velocidad del dron en el diseño del control, se ha implementado un filtro paso bajo con derivador con el fin de eliminar el ruido y obtener la señal derivada, es decir, la velocidad (Ecuación F.1.)

$$G(s) = \frac{s}{1+\tau s} = \frac{V(s)}{P(s)} \tag{F.1}$$

Para la discretización de la función de transferencia mediante Tustin (Ecuación F.2), el valor del periodo es $T = 0{,}08s$ (12,5 Hz) y como mímimo $\tau = 5T$, ya que de otra forma se podría generar aliasing y errores en la discretización. Por lo tanto, se considerará $\tau = 0{,}5$. No obstante, se ha creado una función en Matlab donde dados estos dos parámetros, se discretiza la planta y se filtran los datos en función de dichos parámetros.

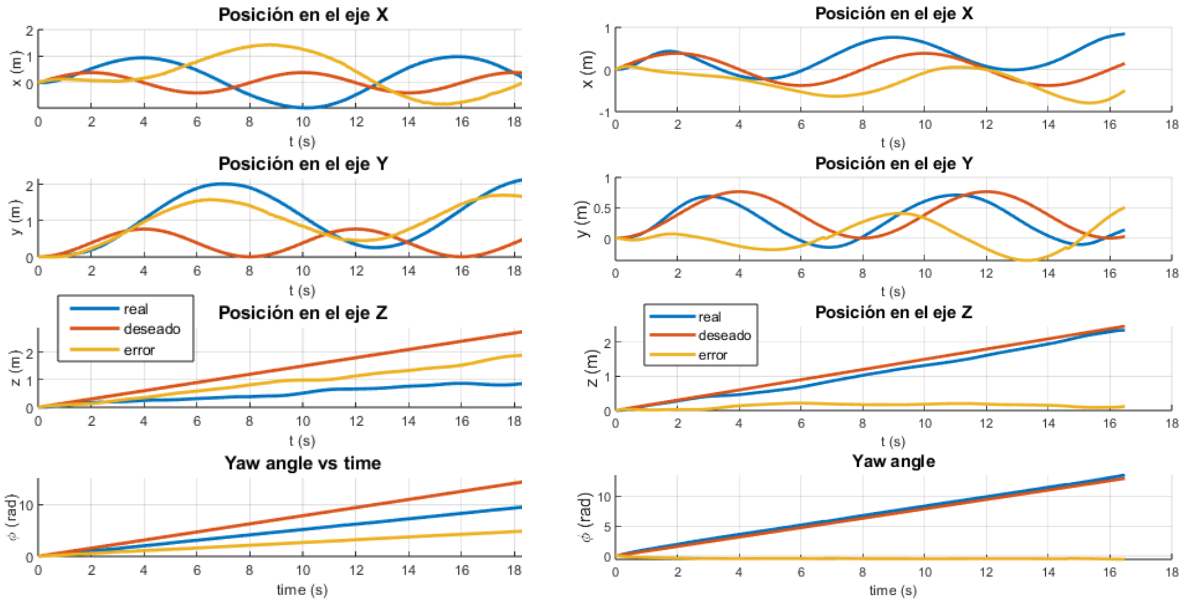
$$G(z) = \frac{az-b}{cz-d} \tag{F.2}$$

Donde para el caso mencionado, $a = b = 1,852$, $c = 1$ y $d = 0{,}8519$. Es estable si $|z| < 1$. Con los parámetros especificados, $|z| = 0{,}8519$, por lo tanto, es estable. Finalmente, pasando al dominio de tiempo obtenemos:

$$v_k = \frac{1}{c}(d * v_{k-1} + a * p_k - b * p_{k-1}) \tag{F.3}$$

G. Otras gráficas relevantes

En este anexo se pueden ver gráficas complementarias para comprender de la mejor forma posible el comportamiento del Iterative Learning Control y como se ha logrado pasar de la trayectoria inicial con mayores errores, a la trayectoria final reduciendo el error notablemente (En un 72,66 % en posición y en un 85,00 % en orientación). En complemento a las imágenes mostradas en la figura 3.12, en la figura G.1 se pueden observar tanto las posiciones como el error cometido en la trayectoria inicial y en la recorrida tras 20 iteraciones.



(a) Posiciones iniciales (Iteración 0) (b) Posiciones tras 20 iteraciones

Figura G.1: Posición en la validación experimental. Trayectoria helicoidal.

Se puede ver como la trayectoria se aproxima más a la ideal especialmente en el eje z y en la orientación.

Este comportamiento se ha logrado de la siguiente forma: a continuación se muestran las gráficas de las velocidades enviadas al dron, las velocidades reales y las deseadas. El algoritmo calcula el error entre la velocidad real y la deseada, utilizándolo para mejorar en los siguientes vuelos. Por ejemplo, en la velocidad de giro en el eje z, el controlador se no ha alcanzado la

velocidad deseada (Figura G.2), por lo que en la siguiente iteración solicitará mayor velocidad con el fin de alcanzar la ideal. El objetivo es que al cabo de varias iteraciones la velocidad acabe siendo la deseada y de dicha forma, mejorar la trayectoria.

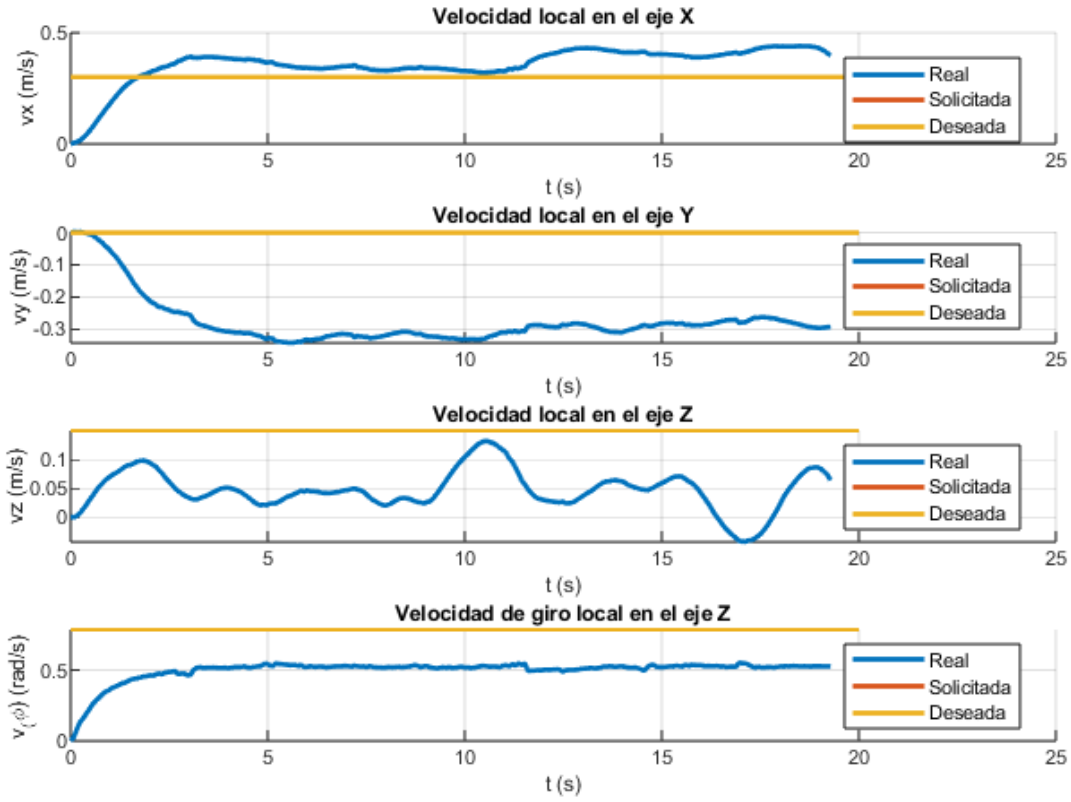


Figura G.2: Velocidad solicitada, real y deseada. Iteración 0.

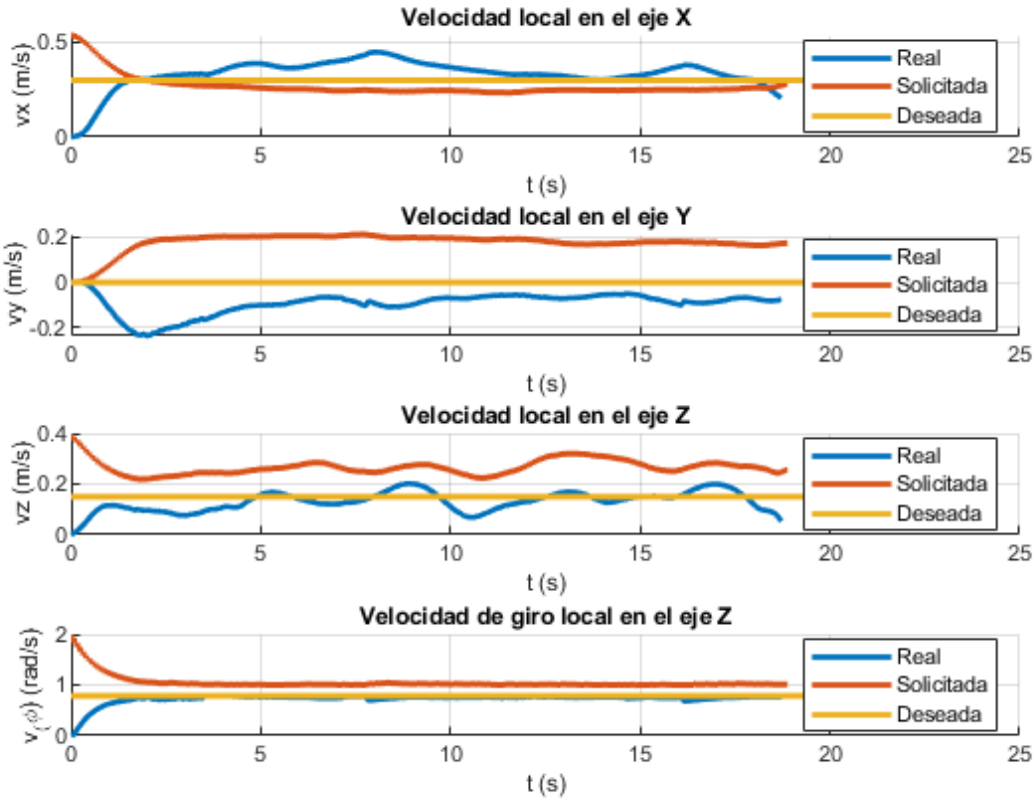


Figura G.3: Velocidad solicitada, real y deseada. Iteración 10.