Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Control predictivo de drone comercial

Autor: Ana Sánchez Amores
Tutor: José María Maestre Torreblanca

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Control predictivo de drone comercial

Autor:
Ana Sánchez Amores

Tutor:
José María Maestre Torreblanca
Profesor titular

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Trabajo Fin de Grado: Control predictivo de drone comercial

Autor: Ana Sánchez Amores

Tutor: José María Maestre Torreblanca

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

Agradecimientos

A mi madre por su apoyo incondicional.

A mi padre por confiar en mí y despertar mi interés hacia la ingeniería.

A mi hermana por ser un ejemplo de esfuerzo y perseverancia.

A mis abuelos por el orgullo que siempre me han transmitido.

A mis amigos y compañeros por acompañarme en este camino.

A mis profesores de la escuela por enseñarme todo lo aprendido, en particular a mi tutor José María Maestre; por el entusiasmo mostrado hacia el tema de este proyecto.

Ana Sánchez Amores

Grado en Ingeniería de Tecnologías Industriales,

Sevilla, 2018

Resumen

En este proyecto se pretende emplear una librería desarrollada por el *Massachusetts Institue of Technology (MIT)* junto con la empresa *Parrot,* para generar un controlador predictivo basado en modelo (*MPC* por sus siglas en inglés) en el entorno de *Matlab/Simulink* que controle el minidrone *Parrot Rolling Spider*. Se quiere comparar el comportamiento de esta estrategia de control con el que presenta el sistema bajo un controlador por asignación de polos. Este último ha sido elegido como controlador óptimo por el alumno Emilio Marín Fernández en su trabajo fin de grado.

En primer lugar, se presenta la instalación del software necesario para la puesta en marcha de esta librería; indicando todos los pasos y comandos necesarios para conseguir la primera prueba en tiempo real, en la que volará el drone.

A continuación, se expone la base teórica del control predictivo basado en modelo, para posteriormente particularizar la estrategia para el problema que se presenta, sintonizando los parámetros necesarios para el control del drone. Se comparará el comportamiento del sistema con el MPC desarrollado con el controlador por asignación de polos.

Se propone una alternativa para tratar de solventar las limitaciones encontradas al tratar de implementar el controlador generado; explicando los conceptos teóricos de la estrategia explícita del control predictivo y buscando herramientas para conseguir este controlador.

Por último, se presentan una serie de ideas para intentar conseguir la implementación de este controlador en el minidrone, que se saldrían del entorno de Matlab/Simulink; en el que está desarrollado el toolbox.

Abstract

This project intends to use a library developed by the Massachusetts Institute of Technology (MIT) together with the Parrot company, to generate a model-based predictive controller (MPC) in the Matlab / Simulink environment that controls the Parrot Rolling Spider minidrone. We want to compare the behavior of this control strategy with that of the system under a controller by pole assignment. The last one has been chosen as the optimal controller by the student Emilio Marín Fernández in his final degree project.

In the first place, the installation of the necessary software for the start-up of this library is presented; indicating all the steps and commands necessary to get the first test in real time, in which the drone will fly.

Next, the theoretical base of the model-based predictive control is exposed, to later customize the strategy for the problem that is presented, tuning the necessary parameters for the control of the drone. The behavior of the system with the MPC developed with the controller will be compared with the pole assignment one.

An alternative is proposed to try to solve the limitations found when trying to implement the generated controller; explaining the theoretical concepts of the explicit model-based predictive control strategy and looking for tools to achieve this controller.

Finally, we present a series of ideas to try to get the implementation of this controller in the minidrone, that would leave the Matlab / Simulink environment; in which the toolbox is developed.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

sen	Función seno
cos	Función coseno
tan	Función tangente
A^T	Matriz traspuesta
A^{-1}	Matriz inversa
R	Matriz de rotación
$\dot{x}$	
$\leq$	Menor o igual
$\geq$	Mayor o igual
ϕ	Ángulo de cabeceo o pitch, giro alrededor del eje X
θ	Ángulo de alabeo o roll, giro alrededor del eje Y
ψ	Ángulo de guiñada o yaw
	Controlador MPC
A, B	Matrices lineales del sistema continuo
Ts	Tiempo de muestreo
Ad, Bd	Matrices lineales del sitema discreto
D	Matriz del sistema que relaciona el estado con las perturbaciones
W	Matriz perturbación
u	Acción de control; entrada al sistema
V	Función de costes
Q	Matriz de pesos del estado; pondera el seguimiento de referencias
R	Matriz de pesos del control; pondera la acción de control
ρ	Ajuste entre matrices de ponderación
α_i	Valor de ponderación del estado *i*
β_i	Valor de ponderación de la acción de control o entrada *i*
x_i max	Valor máximo del estado *i*
u_i max	Valor máximo de la acción de control o entrada *i*

1 INTRODUCCIÓN

Los científicos estudian el mundo tal como es;
los ingenieros crean el mundo que nunca ha sido.

Theodore von Karman

1.1 Motivación

Durante la última década el sector de los vehículos aéreos no tripulados, UAVs (del inglés *Unmanned Aereal Vehicle*), ha ampliado su campo de aplicación inicial para destinarse a un uso civil y comercial. Estos vehículos fueron pensados para situaciones bélicas y aplicaciones científicas o militares, ayudando tanto en la tarea de reconocimiento como en la de ataque.

A dia de hoy, la tecnología empleada en estos se ha desarrollado hasta tal punto que los UAVs colaboran con el ser humano en misiones como mapeo 3D para generar mapas de terrenos, control de calidad del aire, desastres naturales como la detección de incendios, vigilancia, búsqueda y rescate, inspección de estructuras como por ejemplo la detección de averías en tuberías... Se introducen importantes ventajas al no necesitar una persona que pilote el vehículo desde su interior, como por ejemplo poder reducir las dimensiones de este para dotarlo de mayor maniobrabilidad y accesibilidad a lugares de difícil o peligroso acceso para el ser humano, o la reducción del coste de explotación del vuelo frente a los tripulados. La posibilidad de despegue y aterrizaje vertical también supone una ventaja y hasta una necesidad para alguna de estas aplicaciones.

El vuelo del vehículo puede ser remoto, necesitando una persona que lo pilote desde una estación de control, o autónomo, lo que amplia las posibles tareas a realizar. Resultará de gran interés para este trabajo la última opción, ya que el control remoto se realiza mediante radio frecuencia y por lo tanto dependerá del alcance de esta como de la vista del piloto. El control autónomo podrá realizarse mediante una trayectoria preprogramada o mediante arquitecturas inteligentes que se ayudan de sistemas de percepción del entorno para seguir objetos en movimiento y evitar obstáculos entre otros.

En definitiva, el desarrollo de los UAVs ha supuesto un gran impacto para la sociedad, destacando su eficiencia y la mejora de las tareas que realizan, esto es lo que me motiva y empuja a realizar este trabajo, ya que la tecnología avanza a pasos agigantados y estos robots con ella, mejorando y ampliando sus aplicaciones.

1.2 Drones

Los UAVs, también conocidos como drones, incluyen una gama muy amplia de vehículos, que pueden ser clasificados siguiendo criterios como su máxima altitud, alcance, peso, duración de vuelo, métodos de generación de sustentación y tipo de arquitectura. En particular, en este proyecto se utilizará un quadrotor, que es un vehículo o robot volador que se mueve en el espacio 3D; han adquirido una gran popularidad como plataformas róboticas por la facilidad de su control y modelado. Poseen seis grados de libertad lo que facilita su maniobrabilidad, y tienen como diferencia principal con respecto a robots no voladores que su movimiento está expresado en términos de fuerza y pares, es decir, utilizando el modelo dinámico en vez del cinemático.

Se mostrarán a continuación las distintas clasificaciones para determinar en qué categoría se encuentra el vehículo utilizado en este proyecto. Las bases teóricas para este apartado se apoyan en [5] y [6].

La clasificación de la OTAN fue realizada según el peso de la plataforma, el alcance y la altitud que podía conseguir el vehículo durante su vuelo. El tipo de drone que se utilizará en este proyecto se encontraría en la categoría micro-mini, ya que es un vehículo ligero, de uso comercial, que permitirá ser volado hasta una altura máxima de veinte metros aproximadamente, y con muy poca autonomía en cuanto a la duración de la batería.

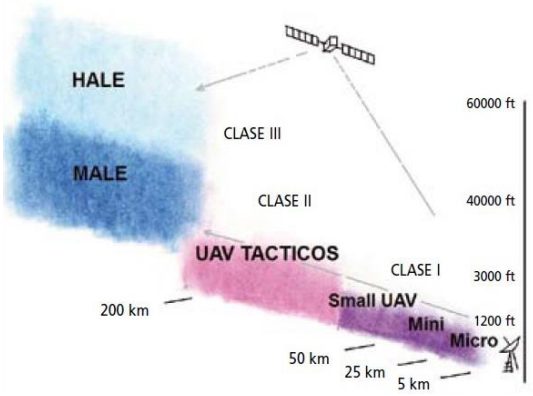

Figura 1-1 Clasificación esquemática según la OTAN [5]

Class	Category	Normal operating altitude	Normal mission radius	Example platforms
Class I (less than 150 kg)	Small >20 kg	Up to 5,000 ft AGL	50 km (LOS)	Luna, Hermes 90
	Mini 2–20 kg	Up to 3,000 ft AGL	25 km (LOS)	Scan Eagle, Skylark, Raven, DH3, Aladin, Strix
	Micro <2 kg	Up to 200 ft AGL	5 km (LOS)	Black Widow
Class II (150–600 kg)	Tactical	Up to 10,000 ft AGL	200 km (LOS)	Sperwer, Iview 250, Hermes 450, Aerostar, Ranger
Class III (>600 kg)	Strike combat	Up to 65,000 ft AGL	Unlimited (BLOS)	
	HALE	Up to 65,000 ft AGL	Unlimited (BLOS)	Global Hawk
	MALE	Up to 45,000 ft AGL	Unlimited (BLOS)	Predator B, Predator A, Heron, Heron TP, Hermes 900

Tabla 1-1 Clasificación detallada según la OTAN [5]

Al hablar de métodos de sustentación estamos clasificando los UAVs en dos grupos; aerostatos y aerodinos. El primer grupo corresponde a aquellos que emplean un gas más ligero que el aire para la suspensión del vehículo en el aire, mientras que el segundo grupo lo componen aeronaves que pesan más que el aire. Dentro del último grupo se podrá realizar otra división atendiendo al tipo de ala empleada, criterio que también se emplea en la clasificación según la arquitectura y tipo de despegue. Según estos criterios, el quadrotor que se emplea corresponde a un aerodino con ala rotatoria y despegue vertical, perteneciendo al grupo de los multirrotor.

En la siguiente página se ilustra el esquema de estos dos tipos de clasificaciones.

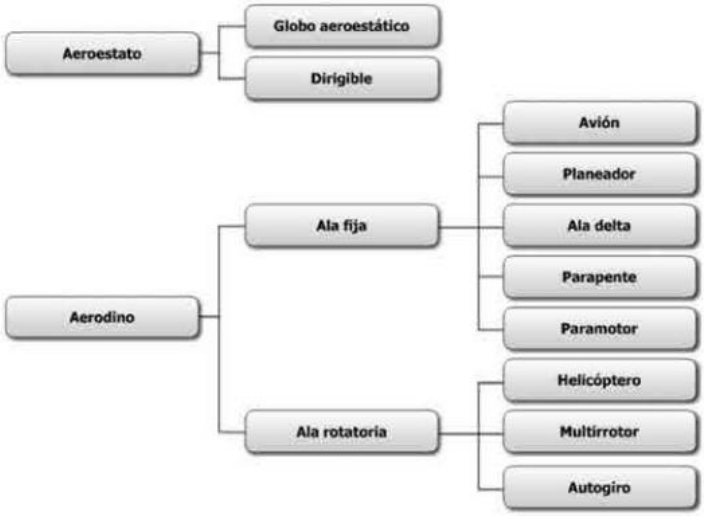

Figura 1-2 Clasificación según su método de sustentación [6]

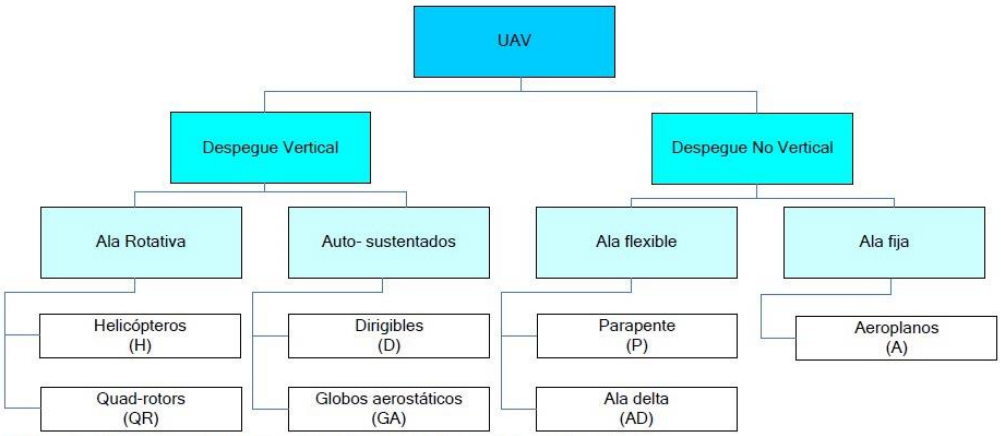

Figura 1-3 Clasificación según su tipo de despegue y arquitectura [5]

1.3 Arquitectura de control

El bucle de control se encargará de estimar el vector de estados del vehículo a partir de la información de los sensores, incluyendo una aproximación de la posición y la orientación del mismo. Esta estimación será mandada al controlador, que también recibirá el estado objetivo o referencia donde se incluyen especificaciones en los distintos ejes para que el drone realice un movimiento determinado. El controlador calculará el error cometido entre la posición deseada y la real para generar la señal de control pertinente que será enviada a los motores del multirrótor para que este trate de alcanzar la posición de referencia.

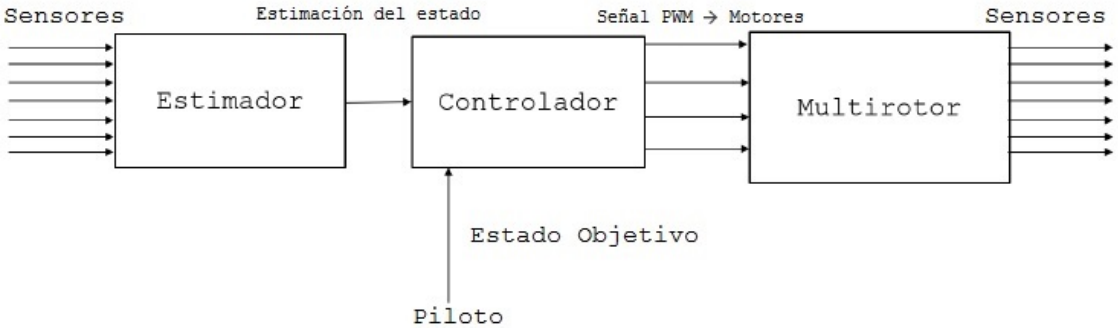

Figura 1-4 Clasificación según su tipo de despegue y arquitectura [4]

1.3.1 Control PID, LQR y asignación de polos

Como se verá a lo largo del proyecto, este se apoya en el trabajo fin de grado *"Plataforma de telecontrol de un drone comercial"* del alumno Emilio Marín Fernández [2], donde se pone en marcha el toolbox para Matlab que se va a utilizar para pilotar el quadrotor. En su proyecto desarrolló tres tipos de controladores diferentes, siendo todos sistemas en lazo cerrado realimentados, poniéndolos a prueba comparando la simulación con el vuelo real del drone en distintas situaciones, y sacando una serie de conclusiones que se expondrán a continuación.

Cabe destacar que ninguno de los siguientes controladores pudo conseguir resultados precisos, ya que el sistema es bastante complejo, y los resultados de las pruebas reales diferían bastante de las simulaciones.

- Control PID. Mejor respuesta en el control de yaw; para pitch y roll no se observaron diferencias notables entre los tres tipos.
- Control LQR *(Linear Quadratic Regulator)*. Buena respuesta ante el seguimieto de trayectoria.
- Control por asignación de polos. Buena respuesta ante el seguimieto de trayectoria y mejor respuesta en el control de altura.

El alumno concluyó que de estos tres tipos de controladores el óptimo, por obtener generalmente un mejor comportamiento en todos los experimentos, era el control por asignación de polos.

1.3.2 Control predictivo basado en modelo

Pretendiendo ir un paso más allá en el control del drone, se tratará de desarrollar un controlador predictivo basado en modelo o MPC (del inglés *Model Predictive Control*); una técnica más sofisticada que se apoya en un modelo interno del sistema que se va a controlar para preveer la respuesta futura ante cada acción de control, permitiéndonos actuar sobre el sistema en el instante actual.

Se explicarán las ecuaciones matemáticas de este controlador más adelante, pero las ideas básicas de esta técnica son las siguientes:

- Uso de un modelo explícito del sistema para tratar de preveer la respuesta del proceso en instantes de tiempo posteriores o futuros.
- Resolución de un problema de optimización que minimiza una función objetivo para calcular las acciones o señales de control.
- Estrategia de control deslizante; se desplaza el horizonte hacia el futuro, calculando la acción de control en cada periodo de muestreo.

La estrategia MPC ha ido ganado popularidad con el paso de los años en el sector industrial por su exitoso comportamiento en sistemas con múltiples entradas y salidas, resolviendo el problema de control considerando restricciones y siguiendo un criterio óptimo.

A priori las características de este tipo de controlador parecen satisfacer los requerimientos del vuelo de un UAV, pero esta técnica carece de suficientes datos empíricos que comprueben que se puede implementar de forma satisfactoria en sistemas de tiempo real. La dificultad para encontrar un modelo fiable del sistema y el gran tiempo requerido para el cálculo del problema de optimización suponen varios de los inconvenientes encontrados en la implementación del MPC a sistemas como UAV.

2 FUNDAMENTOS TÉCNICOS

La educación es el desarrollo en el hombre de toda la perfección de que su naturaleza es capaz.

Immanuel Kant

El software en el que se va a apoyar este proyecto se desarrolló en conjunto por el Instituto Tecnológico de Massachusetts, *MIT*, y la empresa Parrot. En la asignatura de sistemas de control realimentado; *16.30 Feedback Control Systems*, impartida por el departamento de aeronáutica y astronáutica del MIT, los profesores Sertac Karaman y Fabian Riether desarrollaron una librería para Matlab/Simulink. Esta se apoya en el firmware personalizado que Parrot LLC desarrolló para poder abrir una interfaz que permita, a través del toolbox de Matlab/Simulink diseñar y simular algoritmos de control en el mini drone Parrot Rolling Spider, resolviendo también la conexión con el mismo. La librería genera un código en C con el control programado, que será subido al drone para experimentar su funcionamiento y analizar los datos del vuelo.

Este capítulo se apoyará en el trabajo fin de grado '*Plataforma de telecontrol de un drone comercial'* del alumno Emilio Martín Fernández.

2.1 Descripción del drone

Se utiliza el drone Parrot Rolling Spider por su tamaño reducido; lo que lo hace seguro para vuelos en espacios cerrados, la posibilidad de añadirle unas ruedas de protección para mantener la integridad de la estructura del drone en caso de colisión, y su bajo coste. Este drone pertenece a la familia de los quadrotors dentro de la amplia gama de vehículos aéreos no tripulados, ya que su despegue es vertical y tiene cuatro alas rotativas. Las características de este robot son las siguientes:

Figura 2-1 Mini drone 'Parrot Rolling Spider' [7]

- El drone posee un diámetro de 140mm y una altura de 38.1mm.
- Su estructura de plástico le proporciona un peso reducido de 55g que se incrementa a 65g si le colocamos las ruedas de protección.

- Batería extraíble de polímero de litio (LiPo) de 550mAh y 3.7V. Tiene una autonomía de vuelo de 8 minutos, que se reduce a 6 minutos cuando las ruedas están colocadas, que será siempre en este proyecto, por seguridad. El tiempo de carga es de 90 minutos.
- La altura máxima que puede alcanzar es de 20 metros, con una velocidad máxima de vuelo de 5 metros por segundo.
- Para mantener la estabilidad durante el vuelo se tiene un sensor ultrasónico para medir la altura; ideal para vuelos a ras del suelo, un giroscopio de tres ejes; para mantener la orientación en el espacio, un acelerómetro de tres ejes, un sensor de presión; determina una altura más fiable que el sonar en vuelos de mayor altitud, y una cámara vertical de 0.3 megapíxeles situada en el inferior de la plataforma; toma una fotografía del suelo cada 16 milisegundos y la compara con la anterior para averiguar la velocidad a la que está navegando.
- El drone cuenta con unas ruedas de protección que pueden añadirse para mayor seguridad durante el vuelo; las colocaremos siempre antes de cada simulación, ya que así evitamos que un choque contra algún obstáculo o un aterrizaje forzoso dañen la estructura del drone. Estas también permiten que el drone ruede por superficies lisas como el suelo o las paredes.

Figura 2-2 Drone con ruedas de protección [8]

- En la parte frontal hay dos LEDs que darán información sobre el estado en el que se encuentre el drone.
    - LEDs en rojo: batería baja.
    - LED derecho en rojo: cargando (se apagará cuando esté totalmente cargado).
    - LEDs parpadeando verde-rojo: despegando.
    - LEDs en verde: conexión bluetooth exitosa.
- Al ser un quadrotor cuenta con cuatro motores eléctricos de dimensiones 20 x 8 x 3 mm, contando con una hélice de 55mm de diámetro por motor. Dos motores operan con giro horario y los dos restantes con giro antihorario. Los motores son del tipo ‘coreless’, se caracterizan porque su rótor no tiene un núcleo de hierro, lo que lo hace beneficioso para pequeños dispositivos por las siguientes características:
    - Al ser de tamaño reducido tiene baja inercia, lo que resulta en una rápida respuesta ante un cambio brusco de aceleración o de sentido de giro del motor.
    - Tienen baja pérdida de corriente, al desaparecer las denominadas ‘pérdidas en el hierro’, lo que los hace más eficientes.
    - Crean bajos niveles de campo electromagnético, reduciendo así el problema de las interferencias electromagnéticas con otros dispositivos electrónicos.
    - Bajo ruido eléctrico.

- Para el control del vuelo, Parrot dispone de una aplicación gratuita llamada 'FreeFlight3' pero que no será utilizada ya que le instalaremos el firmware personalizado desarrollado por Parrot para el MIT, con el que controlaremos el drone con el sistema operativo Linux.
- La conexión con otros dispositivos se hace utilizando la tecnología Bluetooth Smart, V4.0 o bluetooth de baja energía (BLE, low energy). En este caso, al no disponer el ordenador de dicha tecnología, se ha utilizado el adaptador de red bluetooth (USB 4.0) Trust 18187

2.1.1 Dinámica del drone

La estructura del drone consiste en un marco cruzado con una placa electrónica en su centro y un motor al final de sus cuatro extremidades. Los rotores se pueden dividir en dos grupos que se encuentran separados diametralmente, agrupándolos mediante su sentido de giro; la altitud y posición del vehículo se podrá controlar cambiando la velocidad de los motores.

Para describir la posición del vehículo se puede contar con dos sistemas de referencia; un sistema global fijo que hace referencia a las coordenadas respecto al centro de la tierra; *{E}* en el diagrama, y un sistema local móvil ligado al baricentro del drone; *{B}* en el diagrama.

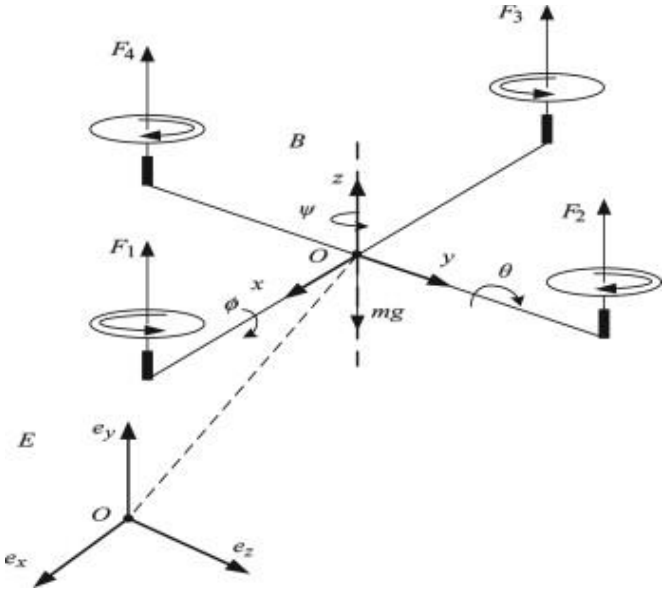

Figura 2-3 Sistemas de referencia quadrotor [12]

El movimiento de los motores genera una fuerza de empuje paralela a su eje de rotación; cada rótor contribuye al empuje total del vehículo (T_z), que se considerará siempre alineado con el eje Z. Se pueden ver las contribuciones individuales al empuje representadas como F_1, F_2, F_3 y F_4 en la imagen superior.

$$T = (0 \quad 0 \quad T_z)^T$$

Se generan tres momentos correspondientes a los pares de torsión producidos por la diferencia de empuje entre los cuatro motores. Cada uno está relacionado con un eje; momento *"roll"* o de alabeo (τ_{roll}); rotación del ángulo ϕ entorno al eje X, momento *"pitch"* o de cabeceo (τ_{pitch}); rotación de θ entorno al eje Y, y momento *"yaw"* o de guiñada (τ_{yaw}); rotación del ángulo ψ ntorno al eje Z.

$$\tau = (\tau_{yaw} \quad \tau_{pitch} \quad \tau_{roll})^T$$

La salida de nuestro controlador será la actuación sobre los motores y corresponderá con un vector de 4 componentes representando la fuerza de empuje y los tres momentos:

$$u = (T_z \quad \tau_{yaw} \quad \tau_{pitch} \quad \tau_{roll})^T$$

De acuerdo con estas cuatro actuaciones, los movimientos que podrá llevar a cabo el vehiculo son los siguientes:

- Vuelo vertical de forma ascendente o descendente: se generará aumentando/disminuyendo en la misma proporción la velocidad de los motores.
- Movimiento de alabeo o balanceo: se generará desequilibrando las fuerzas de empuje F_2 y F_4, resultando en un giro sobre el eje X.
- Movimiento de cabeceo: se generará desequilibrando las fuerzas de empuje F_1 y F_3, resultando en un giro sobre el eje Y.
- Movimiento de guiñada: se generará desequilibrando los conjuntos de fuerzas de empuje, es decir, F_1 y F_3 con F_2 y F_4, resultando en un giro sobre sí mismo (eje Z).

La rotación del quadrotor se define mediante la rotación de un sólido rígido, que se realiza a través de tres rotaciones sucesivas. Estos giros se expresan mediante matrices de rotación:

Rotación alrededor del eje X del ángulo de alabeo:

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

Rotación alrededor del eje Y del ángulo de cabeceo:

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

Rotación alrededor del eje Z del ángulo de guiñada:

$$R_z(\psi) = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Se puede construir una única matriz de rotación mediante la multiplicación de las tres matrices anteriores; combinándolas se expresa la orientación final del cuerpo según el sistema de referencia inercial, es decir, representa lo queha girado respecto al sistema global o fijo.

$$R_{xyz}(\phi, \theta, \psi) = R_z(\psi) \cdot R_y(\theta) \cdot R_x(\phi)$$

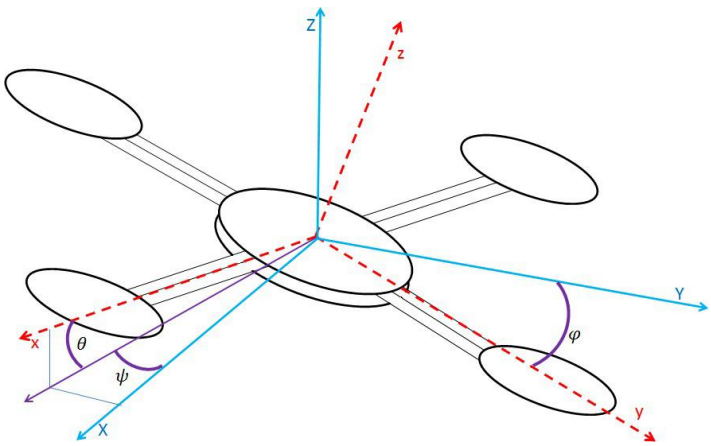

Figura 2-4 Orientación final del vehículo [11]

$$R_{xyz}(\phi,\theta,\psi) = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

$$\boldsymbol{R_{xyz}(\phi,\theta,\psi)} = \begin{pmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix}$$

Las expresiones que vienen a continuación han sido extraidas de [2], ya que el enlace de la página del *Massachusetts Institute of Technology* donde se explicaba el modelo dinámico del drone utilizado, junto con algunas hipótesis y simplificaciones, no se encuentra disponible.

Para continuar desarrollando la dinámica del drone es preciso definir algunos términos:

- Vector de estado, $\boldsymbol{x}$, de 12 componentes:

$$\boldsymbol{x} = \begin{pmatrix} x \\ y \\ z \\ \psi \\ \theta \\ \phi \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{bmatrix} P \\ O \\ \dot{p} \\ \dot{o} \end{bmatrix}$$

  - Vector de coordenadas XYZ, $\boldsymbol{P}$,

$$\boldsymbol{P} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

  - Vector de velocidades lineales, $\dot{\boldsymbol{p}}$

$$\dot{\boldsymbol{p}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix}$$

  - Vector de yaw, pitch y roll (ángulos de Euler), **O**

$$\boldsymbol{O} = \begin{pmatrix} \psi \\ \theta \\ \phi \end{pmatrix}$$

  - Vector de velocidades angulares en los ejes XYZ, $\dot{\boldsymbol{o}}$

$$\dot{\boldsymbol{o}} = \begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix}$$

  - Vector de velocidades angulares según los ángulos de Euler, $\dot{\boldsymbol{O}}$

$$\dot{\boldsymbol{O}} = \begin{pmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{pmatrix}$$

- $\boldsymbol{W^{-1}}$ matriz que relaciona los vectores $\dot{\boldsymbol{o}}$ y $\dot{\boldsymbol{O}}$

$$\boldsymbol{W^{-1}}(\phi,\theta) = \begin{pmatrix} 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \\ 0 & \cos\phi & -\sin\phi \\ 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \end{pmatrix}$$

- Otras variables que entran en juego en la descripción de la dinámica

VARIABLE	DESCRIPCIÓN
$\boldsymbol{m}$	Masa del drone
$\boldsymbol{J}$	Matriz de inercia del drone
$\boldsymbol{G}$	Vector de fuerza gravitacional

Tabla 2-1 Otras variables dinámicas

2.1.1.1 Ecuaciones dinámicas

Una vez definidas la notación empleada junto con los vectores y matrices pertinentes se presentan las siguientes ecuaciones:

$$\begin{cases} \dot{\boldsymbol{P}} = \boldsymbol{R_{xyz}}^{\boldsymbol{T}} \cdot \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{O}} = \boldsymbol{W^{-1}} \cdot \dot{\boldsymbol{o}} \\ \ddot{\boldsymbol{p}} = \boldsymbol{R_{xyz}} \cdot \boldsymbol{G} + \boldsymbol{T}/\boldsymbol{m} - \dot{\boldsymbol{o}} \times \dot{\boldsymbol{p}} \\ \boldsymbol{J} \cdot \ddot{\boldsymbol{o}} = \tau - \dot{\boldsymbol{o}} \times \boldsymbol{J} \cdot \dot{\boldsymbol{o}} \end{cases}$$

2.1.2 Procedimientos de seguridad

Cuando querramos probar un nuevo programa es mejor que primero lo simulemos o testeemos mediante el comando **DroneTest.sh**, que vuela a un 10% de la potencia máxima antes de probar el vuelo a máxima potencia con **DroneRun.sh**.

Se recomienda volar el drone siempre en espacios cerrados y con las ruedas de protección colocadas; esto impedirá un choque directo con las hélices, que, a la velocidad a la que rotan, pueden causar daños físicos o materiales.

Si durante el vuelo el drone sufre una colisión o pierde vision a través de la cámara inferior, los motores se detendrán inmediatamente como medida de seguridad. También se detendrán automáticamente en vuelos que superen los 20 segundos.

2.2 Herramientas

A continuación, se explicará la metodología que se ha de seguir para poner en funcionamiento la librería desarrollada por el MIT junto con la instalación del firmware personalizado de Parrot para poder comenzar a diseñar y simular los controladores deseados. Se explica la instalación del software necesario pasando por una primera toma de contacto con el diagrama de bloques encargado del control del drone hasta el vuelo del vehículo. Estos archivos se pueden conseguir a través de [1].

2.2.1 Software necesario

El toolbox que se va a instalar ha sido diseñado para Ubuntu 14.04, y al no disponer de este sistema operativo se ha instalado Ubuntu 14.04.4 de 64bits en una máquina virtual para Windows; VMWare Workstation 14 Player. No ha sido necesario descargar la imagen ISO de Ubuntu para instalarla en la máquina, ya que, como se ha mencionado anteriormente, este trabajo supone una continuación al trabajo fin de grado desarrollado por el alumno Emilio Marín Fernández en el curso 2016-2017, y se ha contado con su máquina virtual. Esta se ha conseguido en [2], y contenía todos los archivos de su proyecto, así como el toolbox del MIT instalado para nuestro drone y la version R2015a de Matlab, la cual contiene Simulink; necesario para la programación por bloques del controlador.

2.2.2 Instalación firmware y conexión Bluetooth

La instalación del firmware personalizado se tendrá que realizar solamente una vez para la puesta en marcha del drone. Se hará conectándolo mediante un cable USB a nuestro ordenador, pero es importante que este hardware se abra desde la máquina virtual de Ubuntu y no desde nuestro sistema operativo habitual, en este caso Windows. El drone se abrirá como si fuese un dispositivo de memoria; en su interior encontraremos un archivo llamado *fvt6.txt* con información acerca del mismo; como el nombre del producto, del drone, número de serie... Convendrá guardar el nombre del drone y la dirección MAC, esta última nos será de vital importancia para conectar el drone con el PC mediante bluetooth.

MyDrone ×

```
# Rolling Spider FVT6 firmware status
#

[General Information]
ProductName=RollingSpider
DroneName=RS_W121721
SerialNumber=PI040319AE4J121721
FWVersion=0.0.0
HWVersion=05
MACAddress=A0:14:3D:4A:AB:A0

[Sensors]
MPU6050=OK
MS5607=OK
ESP668F=KO
MT9V117=NOTTESTED
CSR8811=NOTTESTED

[Global Status]
FWStatus=NOTTESTED
```

Figura 2-5 Archivo *fvt6.txt*

Tras copiar la dirección MAC, abriremos una ventana del terminal para guardar esta dirección:

DroneSetMACaddress.sh [A0:14:3D:4A:AB:A0]

Para instalar el firmware basta con copiar el archivo *rollingspider.edu.plf* que se encuentra en RollingSpiderEdu/Parrot_customFirmware descargado desde [1] a nuestro drone, o bien en una ventana del terminal: **EDUfirmwareUploadSYS.sh**

Desconectaremos el drone expulsando el USB y retirándolo de nuestro ordenador, para posteriormente colocar la batería cargada. Cuando los LEDs dejen de parpadear el firmware actualizado se habrá subido correctamente a nuestro drone.

Como ya hemos guardado la dirección MAC de nuestro drone podremos proceder a realizar una conexión inalámbrica, es decir, sin necesidad de cable USB, por bluetooth. Para esto conectaremos el adaptador BLE Trust 18187 a nuestra máquina virtual, y nos conectaremos al drone desde el terminal: **DroneConnect.sh**

Subiremos los archivos del firmware al drone, y una vez subidos lo reiniciaremos:

EDUfirmwareUploadFILES.sh

DroneReboot.sh

Finalmente, conectaremos de nuevo el drone para inicializarlo, y también para realizar la inicialización del nuevo firmware que hemos instalado.

DroneConnect.sh

EDUfirmwareInitialize.sh

DroneInitialize.sh

Si queremos abortar la conexión del drone con el PC mediante el terminal: **DroneDisconnect.sh**

2.2.3 Diseño de controladores

Denotaremos el directorio principal donde se encuentran ubicados los archivos del toolbox como [RST] (de Rolling Spider Toolbox), que coincide con la siguiente dirección:

[RST] = ~/RollingSpiderEdu-master/MIT_MatlabToolbox

Para comenzar a diseñar nuestros controladores tendremos que comezar abriendo MATLAB, se recomienda hacerlo mediante el terminal con el comando **sudo matlab** ya que es frecuente que de problemas al intentar abrirlo mediante un acceso directo. Es importante no cerrar nunca la ventana de terminal con la que hemos abierto el programa porque sino abortará su ejecución.

Una vez iniciado el programa, como directorio de trabajo elegiremos la carpeta [RST]/trunk/matlab; en la que nos encontraremos otras tres carpetas y un archivo '.m'.

- *Simulation/* contiene los ficheros de Simulink para diseñar y simular el vuelo deseado del drone con sus respectivos estimadores y controladores.
- *libs/* contiene la dinámica del drone Parrot Rolling Spider, particularizando sus parámetros a la dinámica del drone expuesta en el toolbox de robótica de Peter Corke.
- *ExperimentAnalyzer/* contiene los ficheros necesarios para analizar los datos recogidos por los sensores y la dinámica del drone durante el vuelo o simulaciones

.

Se ejecutará el archivo '.m' llamado *startup.m*, lo que abrirá la carpeta de simulación, pero añadiendo las otras dos a la ruta, para tener todos los archivos en un mismo directorio y no tener que trasladarnos de uno a otro.

Abriremos el archivo de simulink que contiene el diagrama de bloques que se encarga del control y simulación del vuelo, llamado *sim_quadrotor.slx*. Nos encontraremos la siguiente ventana:

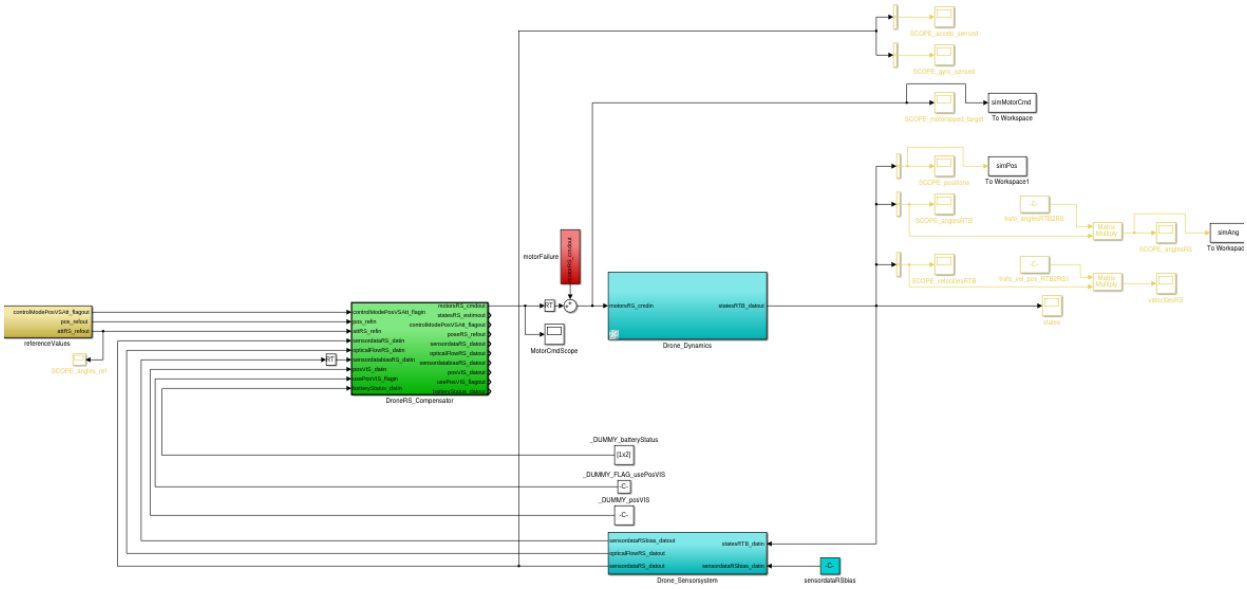

Figura 2-6 *sim_quadrotor.slx*

En este caso, como hemos mencionado anteriormente, el archivo se ha recuperado de la máquina virtual de un alumno de la escuela que había desarrollado distintos controladores para este drone, por lo que hay ligeros cambios en la estructura del diagrama de bloques con respecto a los que estarían en el archivo original del toolbox del MIT. Se procede a describir los bloques que componen el diagrama anterior, de izquierda a derecha:

- *ReferenceValues:* fija la referencia del pitch, roll, yaw y altura del drone para la simulación; las referencias para el vuelo en tiempo real se introducirán por teclado. Para el seguimiento de trayectorias, podemos definir una referencia variable mediante escalones a diferentes instantes de la simulación.

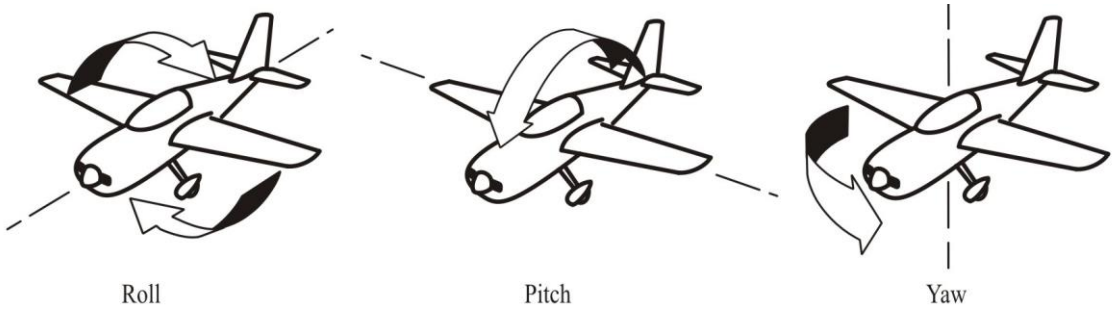

Figura 2-7 Ejes del drone [9]

- *DroneRS_Compensator:* es el bloque principal para el diseño de nuestro controlador; se incluye una estimación de la posición y orientación del robot, estas son comparadas con los datos recibidos de los sensores (presión, sonar, cámara...) y calcula el error que está cometiendo el estimador.

  Se va a intentar compensar o corregir este error con un controlador; por defecto, con el toolbox del MIT recién instalado, vendrá el 'hueco' para que coloquemos el bloque del sistema de control deseado. Si se quiere usar un controlador PID nos iremos a la carpeta controller_PID2W, que se encuentra en el directorio [RST]/trunk/matlab/Simulation/controllers/ y abriremos el archivo que tenga extensión de Simulink (.mdl o .slx) para copiar el bloque del controlador en el diagrama principal.

Se adjunta en la siguiente imagen lo que encontramos al abrir el proyecto desarrollado por Emilio, ya que en su trabajo fin de grado se encargó de poner a prueba la respuesta del drone ante distintos controladores como un PID (Proporcional-Integral-Derivativo), LQR ("Linear Quadratic Regulator") o por asignación de polos; también conocido como FSF ("Full-State Feedback"). Para sintonizar estos últimos dos controladores se programan mediante un archivo '.m' (*LQRcontrol.m* y *asignacionPolos.m*) que podemos encontrar en sus respectivas carpetas.

La actuación calculada por el controlador para tratar de contrarrestar el error en la estimación irá directamente a los motores.

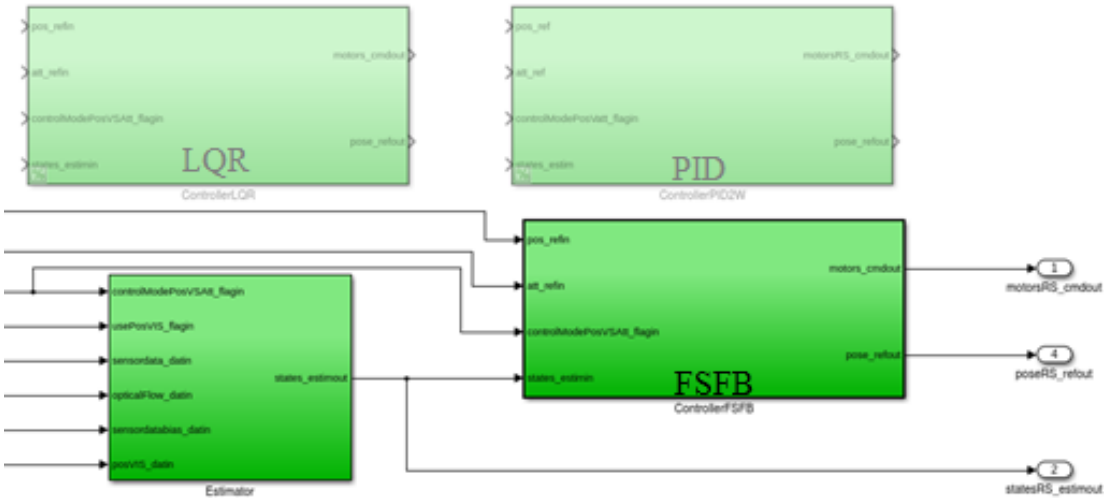

Figura 2-8 DroneRS_Compensator

- *Drone_Dynamics:* contiene el modelo dinámico de la respuesta del drone, el cual está contenido en el archivo *mdl_quadrotor.m*, que se encuentra en [RST]/trunk/matlab/libs/RoboticsToolbox. Gracias al toolbox de Peter Corke se dispone de este modelo dinámico que, recibiendo la salida del bloque de control que será la actuación que se manda a los motores, calculará el vector de las variables de estado del drone ante esta entrada. Dentro de este código se definen los parámetros dinámicos para el quadrotor, además de constantes características del entorno de vuelo del drone, todo en unidades del sistema internacional.

VARIABLE	DESCRIPCIÓN
quad	Estructura que define los parámetros del drone como el número de rótores junto con su masa, dimensión e inercia entre otras caracterísitcas. También se definen constantes como la gravedad o la densidad y viscosidad del aire.
quadEDT	Estructura que contiene características de los sensores; como ganancias de cada tipo de sensor, ruido, valores de saturación, velocidad del flujo óptico... También define parámetros de la simulación como banderas que habilitan características o tiempos para el fallo de los motores.
sampleTime_qcsim	Periodo de muestreo *(5ms)*

Tabla 2-2 Variables definidas en *mdl_quadrotor*

- *Drone_Sensorsystem:* recibe el vector de estados resultante del bloque del modelo dinámico del drone y estima el valor de los sensores, que serán enviados a *DroneRS_Compensator*.

- *Scopes:* sirven para representar graficamente los distintos parámetros que varían durante la simulación como son el ángulo, la velocidad y la posición del drone, los datos recogidos por el acelerómetro y giróscopo...

Cuando hayamos terminado de modificar o diseñar nuestro controlador, lo tendremos que incluir en el sitio que tiene destinado para esto dentro del diagrama de bloques principal. La genereración el código C que será subido al drone se hará presionando el botón derecho del ratón sobre el bloque *DroneRS_Compensator*. En el menu que se despliega pulsaremos **C/C++ Code** y **Build This Subsystem**; en el cuadro de diálogo que se desplegará tendremos que seleccionar **Build**.

Para subir el código compilado se abrirá una ventana de terminal, y en primer lugar conectaremos el drone con el comando **DroneConnect.sh**. Una vez conectado, se subirá el código compilado al vehículo a través de la instrucción: **DroneUploadEmbeddedCode.sh**, finalizados estos dos pasos, ya estará el sistema listo para ser probado en un vuelo a tiempo real.

2.2.4 Vuelo y análisis

La secuencia de vuelo del drone se puede dividir en tres fases distintas: calibración, despegue y vuelo. Durante la primera fase el drone permanecerá varios segundos inmóvil en el suelo antes de despegar para calibrar sus sensores; acelerómetro, giróscopo, sensor de presión, sensor ultrasónico y cámara. El despegue durará el tiempo que se le especifique, para proseguir con el vuelo, que no durará más de veinte segundos por defecto, ya que el sistema detendrá los motores en ese caso.

Estos tiempos; encendido, calibración y despegue, se podrán modificar en el código principal en lenguaje C; *rsedu_control.c*, que se encarga del control del drone. Se define en este *RSEDU_control*; la función principal de este código. Al principio de este código se podrá personalizar la configuración de los tiempos a través de las siguientes variables:

VARIABLE	DESCRIPCIÓN
onCycles	Ciclos en los que está el drone encendido; manipular este parámetro habilita que el tiempo de vuelo sea mayor a 20 segundos, que es lo que se encuentra programado por defecto.
calibCycles	Ciclos empleados para la calibración de los sensores.
takeoffCycles	Ciclos empleados para el despegue del vehículo.

Tabla 2-3 Variables de tiempo

Como se mencionó anteriormente, todos los bloques del diagrama de simulink menos *DroneRS_Compensator* se encargarían de la simulación del control del drone pero no del vuelo real. En el código principal, *rsedu_control.c*, estará incluido todo el código referente a la dinámica del vehículo y sus sensores; los bloques *Drone_Dynamics* y *Drone_SensorSystem*.

En el inicio del código, al igual que con las variables de tiempo, existen otras características que se pueden habilitar o deshabilitar cambiando un 1 por un 0 en su valor, se definen a continuación:

VARIABLE	DESCRIPCIÓN
FEAT_TIME	Registra las marcas de tiempo entre la entrada y salida de funciones.
FEAT_OF_ACTIVE	Utiliza datos del flujo óptico para estabilizar la posición.
FEAT_POSVIS_RUN	La cámara busca la configuración de un punto de referencia y reconstruye la posición si se encuentran todos los puntos de referencia. Se registra la posición que se ha reconstruido visualmente.
FEAT_POSVIS_USE	Utiliza la posición que se ha reconstruido visualmente para mejorar la estimación de posición.
FEAT_NOLOOK	Calcula la conversión de color, coincidencia de referencias y otras opciones en línea, en vez de utilizar tablas de búsqueda precalculadas.
FEAT_IMSAVE	Si es igual a 1 guarda todas las imágenes que la cámara va tomando; esta funciona a 60Hz y la captura o guardado de imágenes se hace a 10Hz. Si es igual a 2 las imágenes se están transmitiendo a la máquina de ubuntu en *streaming*.
FEAT_NOSAFETY	Si está a 1 supone que no detendrá los motores en caso de despegue en superficies no niveladas, aceleración el eje Z positiva, aceleración en los ejes X-Y mayor que $6m/s^2$ o en caso de choque. Es una configuración que puede resultar ser peligrosa.

Tabla 2-4 Características personalizables

Durante el vuelo, el drone tratará de alcanzar las referencias que se le especifiquen por teclado gracias a la acción del controlador que se esté utilizando. Con el drone conectado y el código compilado subido, se abrirá otra ventana de terminal y escribiremos:

DroneKeyboardPilot.sh

Cuando se ejecuta, se habilita el archivo *ReferenceValueServer.c*, que se encuentra en [RST]/trunk/embcode, este código es el equivalente al bloque *ReferenceValues* del diagrama de Simulink. Incluye el protocolo seguido cuando se detecta una tecla pulsada, con el fin de cambiar la referencia de sus ejes, y el envío de esta al drone mediante un socket.

Se comprobará siempre que aumentemos o decrementemos la referencia si se está superando el valor máximo o mínimo permitido para cada eje; si se da este caso, se saturará el valor para evitar problemas.

Es importante que, cuando se quiera modificar la referencia, no se mantega presionada la tecla durante un periodo de tiempo; solo con pulsarla una vez es suficiente.

Este comando habilitará el cambo de referencia de los ejes del drone por teclado durante el vuelo de la manera en la que se ilustra en la tabla que se adjunta.

EJE	ACCIÓN	TECLA
Roll	+ 0.04	D
	- 0.04	A
Pitch	+ 0.04	S
	- 0.04	W
Yaw	+ 0.2	L
	- 0.2	J
Altura	+ 0.2	K
	- 0.2	I
	+ 0.6	H
	- 0.6	Y
Parada forzosa		E
Aterrizaje progresivo		Z

Tabla 2-5 Control por teclado de la referencia

En otra ventana de terminal ejecutaremos primero:

DroneTest.sh

Es aconsejable, por razones de seguridad, probar primero nuestro vuelo con este comando, ya que el drone volará a un 10% de su potencia máxima. Para volarlo a máxima potencia se introducirá en el terminal:

DroneRun.sh

Cuando se introduce alguno de estos dos comandos, tendremos que cambiar inmediatamente a la ventana de terminal en la que se ha mandado la instrucción **DroneKeyboardPilot.sh** para comenzar el control de la referencia por teclado. Si en algún momento se quiere abortar el vuelo, bastará con pulsar la tecla 'E' estando en la ventana del terminal en la que hemos abierto el control por teclado.

Para el análisis del vuelo tras su finalización se tendrá que introducir el siguiente comando en el terminal:

DroneDownloadFlightData.sh

Esto hará que los datos del vuelo se descarguen en una matriz de datos llamada *RSdata.mat*, que se encontrará ubicada en la ruta [RST]/DroneExchange. Abriremos una ventana de MATLAB y cargaremos los datos escribiendo 'load(RSdata.mat)' o bien haciendo doble click en el archivo. Es importante comprobar que previamente se ha ejecutado el archivo *startup.m*, que se encuentra en [RST]/trunk/matlab. Seguidamente, se ejecutará el script *FlightAnalyzer*, el cual analizará los datos de vuelo y presentará distintas gráficas con los resultados del vuelo:

- Voltaje de la batería; *'Battery voltage'*. Se presenta una gráfica en la que se observa cómo el voltaje disminuye gradualmente con el paso del tiempo, ya que la batería se va agotando.
- Altitud; *'Attitude'*. Se representa cómo el drone trata de alcanzar poco a poco las nuevas referencias de altura; eje Z, y se observa el error que presenta en régimen permanente, ya que oscila entorno a la referencia.

- Sensores; *'IMU, attitude, motor commands'*. Se representan cuatro gráficas distintas en la misma ventana; destacar que las dos primeras tienen que ver con la unidad de medición inercial (IMU: *Inertial Measurement Unit*) del acelerómetro y el giróscopo, lo que nos da una idea de la estabilidad del vuelo.

2.2.4.1 Ejemplo

En este apartado se presentarán los resultados de una simulación de vuelo con el vuelo real, utilizando, para la misma referencia de altura, dos controles de distinto tipo: PID y LQR. En primer lugar, para un vuelo de aproximadamente 20 segundos, se ha introducido una referencia de 0.7m en el eje Z como una entrada en escalón a los 0.8s de vuelo en el bloque *ReferenceValueServer*. La orientación del vehículo no se ha variado.

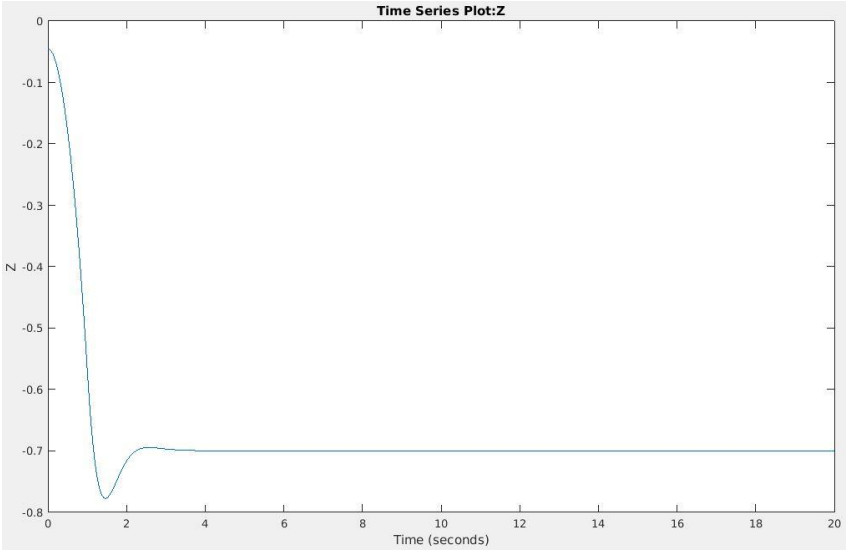

Figura 2-9 Simulación Altura PID

En la gráfica superior, podemos observar la simulación del controlador PID que viene por defecto en el diagrama de Simulink es el PID. El vuelo del quadrotor con el controlador PID fue abortado antes de los 20s, y el resultado proporcionado por los sensores de presion y ultrasónico para la medición de la altura es el siguiente:

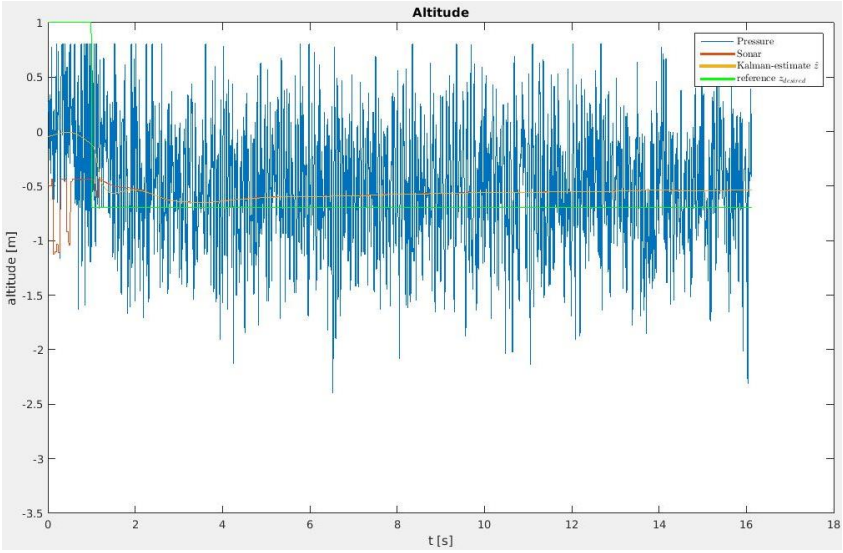

Figura 2-10 Altura PID en vuelo

Para el controlador LQR se ha ejecutado en primer lugar el archivo *LQRControl.m* para obtener la ganancia de control que se implentará en el bloque del controlador dentro de *DroneCompensator*.

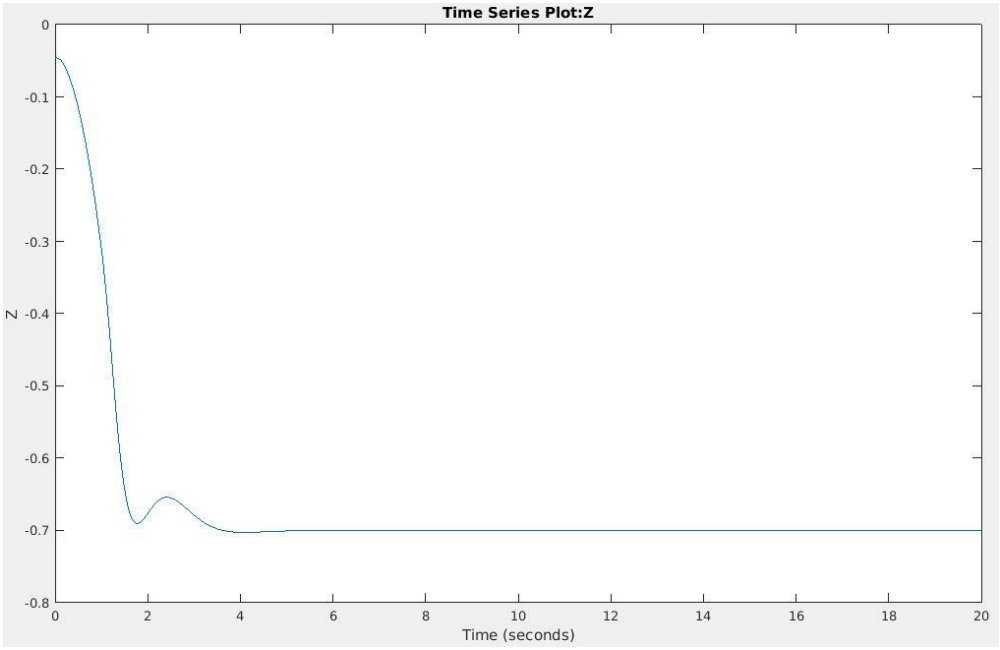

Figura 2-11 Simulación Altura LQR

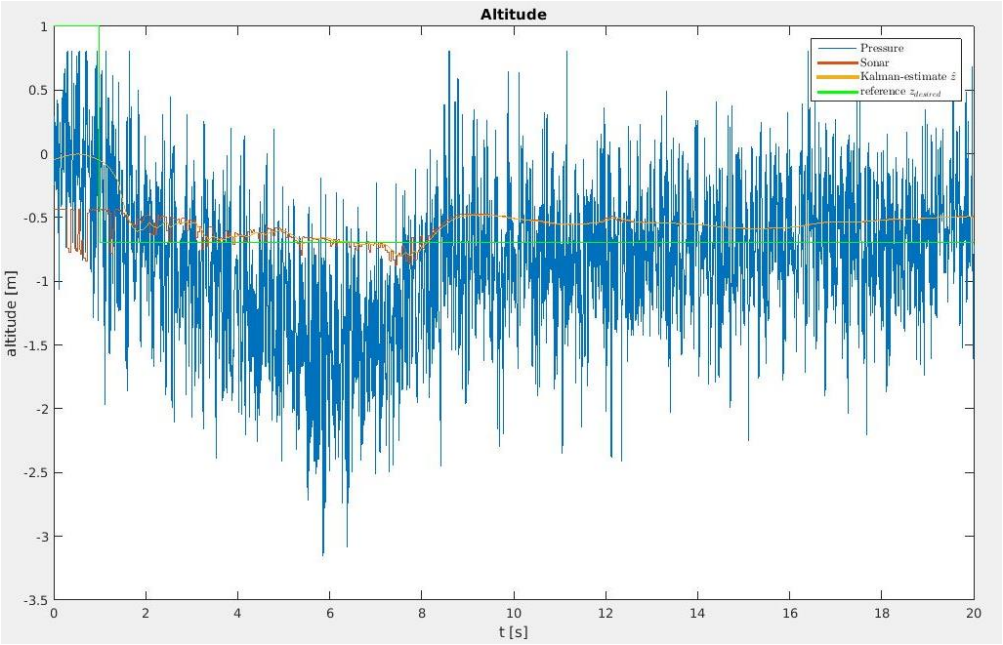

Figura 2-12 Altura LQR en vuelo

3 ESTRATEGIA MPC

3.1 Introducción

El control predictivo basado en modelo, MPC haciendo referencia a sus siglas en inglés (*Model Predictive Control*), consiste en una técnica de control más sofisticada en comparación con, por ejemplo, el control PID, que ha ido ganando popularidad en los últimos años para convertirse en la técnica con mayor éxito en el ámbito industrial. Esta estrategia trata de preveer la respuesta de la planta ante una determinada acción de control, ayudándose de un modelo interno del sistema a controlar para realizar esta predicción. Gracias al modelo, las acciones de control serán calculadas atendiendo a un criterio óptimo para conseguir que el estado del sistema se asemeje a las referencias deseadas. Se establecerá un horizonte de predicción N y se calcularán las actuaciones óptimas para los N siguientes instantes, aplicando solamente la primera componente y descartando el resto para cada periodo de tiempo.

La resolución del problema de optimización puede derivar en una serie de limitaciones debido al coste computacional y al elevado tiempo de cálculo, debido a esto, en sus inicios, la estrategia MPC se aplicaba a sistemas con dinámicas lentas y con periodos de muestreo largos, como por ejemplo la industria química, petrolera o de procesos. A causa del desarrollo de técnicas de optimización cuadrática, se ha conseguido agilizar el cálculo de la solución del problema de optimización, permitiendo así la aplicación de esta estrategia de control a dinámicas más rápidas.

Además de la limitación anterior, la estrategia MPC presenta un gran inconveniente al tener que contar con un modelo fiable del sistema, ya que el éxito de esta técnica de control dependerá de las discrepancias entre el sistema real y el modelo. Sin embargo cuenta con múltiples ventajas como la compensación de retrasos y la posibilidad de trabajar con sistemas multivariables con restricciones de entrada y salida; valores límites de la acción de control y estados del sistema. También cabe destacar la simplicidad para sintonizar el controlador, ya que solo hay que configurar dos matrices diagonales que se encargan de ponderar la función de coste, que será la función a optimizar, con el fin de priorizar el seguimiento de referencias o minimizar la brusquedad en la variación de la acción de control.

3.2 Etapas de control

Se pueden distinguir tres etapas diferenciadas en el algoritmo de los controladores predictivos:

1. Predicción.

Se predice el estado futuro del sistema en cada periodo de muestreo y durante el horizonte de predicción especificado, es decir, para un horizonte de predicción N, en el instante k calcularemos el estado del sistema desde este hasta el $k+N$. El comportamiento del sistema será función de las entradas o actuaciones pasadas, salidas o estados pasados, y entradas o acciones de control futuras.

2. Optimización.

Para obtener las acciones de control futuras se tratará de resolver un problema de optimización que minimice una función de coste, obteniendo un conjunto de actuaciones óptimas que serían las futuras entradas a nuestro sistema.

3. Horizonte deslizante.

Se aplicará únicamente la primera componente de nuestro conjunto óptimo de acciones de control y se desecharán el resto, repitiendo en el siguiente tiempo de muestreo el mismo proceso, por lo que el horizonte se desplazará un instante hacia el futuro para cada iteración.

3.3 Elementos básicos

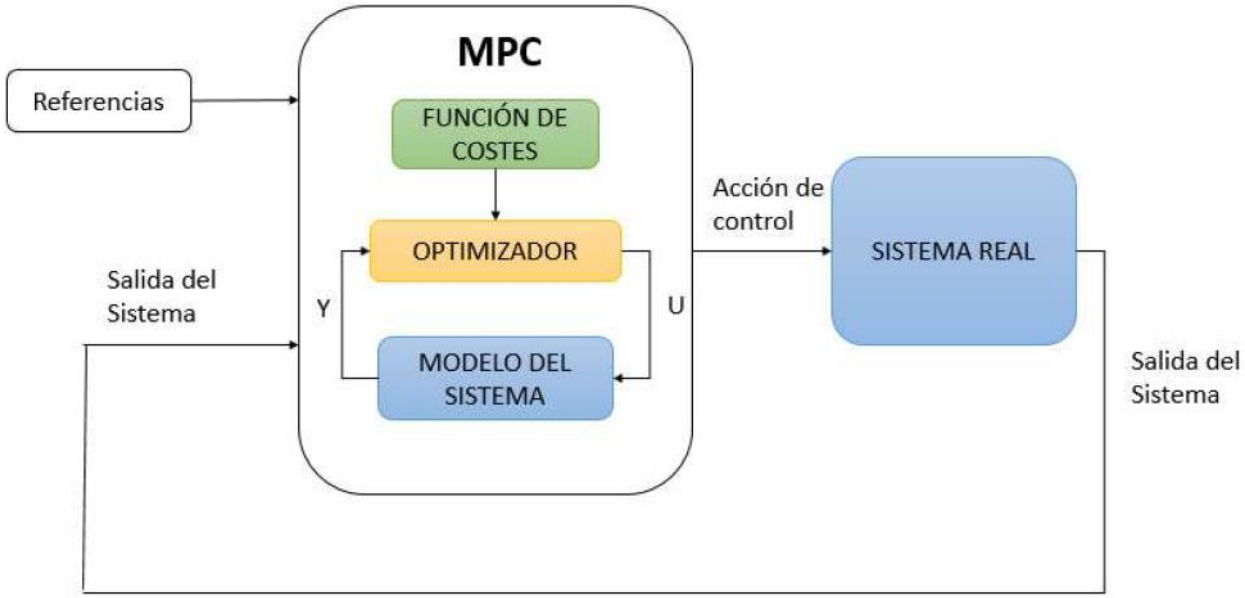

Figura 3-1 Esquema de control MPC [10]

En el bloque central del esquema anterior podemos observar los distintos elementos que componen el controlador predictivo.

- Modelo del sistema.

  Elemento fundamental para esta técnica de control, ya que se utilizará para predecir el estado del proceso a lo largo del horizonte de predicción ante la aplicación de distintas señales de control. El MPC es una técnica de control potente cuando el modelo reproduce de una manera fiable y precisa al sistema real, ya que de esta manera la solución del problema de optimización será mejor.

- Función de costes

  Esta función define define al controlador y, al ser minimizada, proporciona las acciones de control. Para definir el controlador solamente tendremos que sintonizar dos parámetros, que se encargarán de establecer la estrategia de control según si se quiere priorizar la minimización de la diferencia entre las salidas predichas y las referencias, es decir, el seguimiento de referencias, o la suavidad en el cambio de acciones de control. De esto se encargarán respectivamente las matrices diagonales Q y R, por lo que en general resulta en un compromiso entre el error que se comete y el esfuerzo de control.

- Optimizador

  Resuelve el problema de optimización buscando acciones de control que minimicen la funcion de costes, atendiendo a restricciones si se requiere. Para cada instante de muestreo el optimizador calcula las acciones de control futuras que minimizan la función de costes objetivo pero solo aplicando la primera componente y desechando el resto.

Generalmente las funciones de coste son cuadráticas, por lo que la solución que proporciona el optimizador utilizará algoritmos de programación cuadrática. Una herramienta muy útil para el MPC es la función *"quadprog"* (*quadratic programming*) de Matlab que se encarga de buscar el mínimo de una función sujeto a restricciones si las presenta.

3.4 Algoritmo para modelo en espacio de estados

En este caso, el modelo del sistema se expresará en espacio de estados, representación compacta que sirve para sistemas multivariables, dicho modelo tendrá que ser discreto ya que la implementación del controlador es discreta. El vector de estados del sistema contiene todas las variables necesarias para definir completamente el estado dinámico de una planta en un instante de tiempo. Este vector dependerá de los estados, entradas y perturbaciones anteriores de la siguiente forma:

$$x_{k+1} = A \cdot x_k + B \cdot u_k + D \cdot w_k \tag{3-1}$$

Donde:

x_{k+1} vector de estados del sistema en el instante *k+1*

x_k vector de estados del sistema en el instante *k*

$x \in \mathbb{R}^{n_x \times 1}$ siendo n_x el número de estados

u_k vector de entrada o control del sistema en el instante *k*

$u \in \mathbb{R}^{n_u \times 1}$ siendo n_u el número de entradas

w_k vector perturbación en el instante *k*

$w \in \mathbb{R}^{n_w \times 1}$ siendo n_w el número de perturbaciones

Matriz *A* modela la relación entre el estado actual y el anterior. $A \in \mathbb{R}^{n_x \times n_x}$

Matriz *B* modela la relación entre el estado actual y la entrada anterior. $B \in \mathbb{R}^{n_x \times n_u}$

Matriz *D* modela la relación entre el estado actual y la perturbación anterior. $D \in \mathbb{R}^{n_x \times n_w}$

El estado x_k podrá calcularse conociendo las condiciones iniciales de mi sistema x_0, las entradas u_k pueden ser calculadas resolviendo el problema de optimización, y las perturbaciones w_k deberán de ser estimadas.

Definimos el parámetro *N* horizonte de predicción, donde para cada instante de muestreo, las entradas óptimas serán calculadas para los siguientes *N* instantes de tiempo. La siguiente representación matricial permite expresar las *N* componentes del vector de estados.

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} \in \mathbb{R}^{N \cdot n_x \times 1}, \qquad U = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} \in \mathbb{R}^{N \cdot n_u \times 1}, \qquad W = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-2} \\ w_{N-1} \end{bmatrix} \in \mathbb{R}^{N \cdot n_w \times 1}$$

$$G_x = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^{N-1} \\ A^N \end{bmatrix}, \qquad G_u = \begin{bmatrix} B & 0 & 0 & 0 & 0 \\ AB & B & 0 & 0 & 0 \\ A^2B & AB & B & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \dots & AB & B \end{bmatrix}, \qquad G_w = \begin{bmatrix} D & 0 & 0 & 0 & 0 \\ AD & D & 0 & 0 & 0 \\ A^2D & AD & D & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A^{N-1}D & A^{N-2}D & \dots & AD & D \end{bmatrix}$$

$$G_x \in \mathbb{R}^{N \cdot n_x \times n_x}, \qquad G_u \in \mathbb{R}^{N \cdot n_x \times N \cdot n_u}, \qquad G_w \in \mathbb{R}^{N \cdot n_x \times N \cdot n_w}$$

El estado del sistema se expresa mediante la ecuación matricial

$$X = G_x \cdot x_0 + G_u \cdot U + G_w \cdot W \tag{3-2}$$

Minimizando la función de coste de la ecuación 3-3 obtendremos la acción de control óptima para nuestro sistema. El objetivo de la función será tratar de llevar a cero el estado del sistema, por lo tanto, para el seguimiento de referencias bastará con realizar un cambio de variables en el que $x = x_{real} - x_{ref}$, ya que tendiendo la resta a cero el sistema se aproximará al valor deseado.

$$V = \sum_{0}^{N-1} ({x_{k+1}}^T \cdot Q \cdot x_{k+1} + {u_k}^T \cdot R \cdot u_k) \tag{3-3}$$

El primer sumando representa el coste del estado y el segundo el coste de control, se ponderarán ambos costes con dos matrices diagonales de pesos, *Q* para el seguimiento de referencias y *R* para la acción de control, determinando así la estrategia del controlador.

Para expresar la función de costes según la notación matricial anterior construimos las matrices

$$\hat{Q} = \begin{bmatrix} Q & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & Q & 0 \\ 0 & 0 & 0 & Q \end{bmatrix} \in \mathbb{R}^{N \cdot n_x \times N \cdot n_x}, \qquad \hat{R} = \begin{bmatrix} R & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & R & 0 \\ 0 & 0 & 0 & R \end{bmatrix} \in \mathbb{R}^{N \cdot n_u \times N \cdot n_u}$$

Quedando la función de costes de esta manera

$$V = X^T \cdot \hat{Q} \cdot X + U^T \cdot \hat{R} \cdot U \tag{3-4}$$

Sustituyendo en la ecuación 3-4 la expresión matricial del vector de estados para el horizonte de predicción *N*; ecuación 3-2, y eliminando términos constantes que no participan en el problema de optimización se tiene

$$V = 2\big((G_x x_0)^T \hat{Q} G_u + (G_w W)^T \hat{Q} G_u\big)U + U^T\big({G_u}^T \hat{Q} G_u + \hat{R}\big)U \tag{3-5}$$

Agrupando términos definimos las matrices *F* y *H*

$$F = (G_x x_0)^T \hat{Q} G_u + (G_w W)^T \hat{Q} G_u \tag{3-6}$$

$$H = {G_u}^T \hat{Q} G_u + \hat{R} \tag{3-7}$$

Quedando la función a optimizar de la siguiente forma

$$\frac{1}{2} U^T \cdot H \cdot U + F \cdot U \tag{3-8}$$

El problema de optimización anterior se podrá resolver de forma computacional con el software Matlab; pasando como argumento de la función *quadprog* las matrices *H* y *F* se obtiene la matriz U como resultado de la minimización de la función de costes.

$$U = quadprog(H, F)$$

Si queremos satisfacer las restricciones en estado y entradas expresadas a continuación deberemos formularlas acorde con la notación matricial para el horizonte *N* de predicción.

$$x_{min} \leq x_k \leq x_{max} \ \forall k > 0 \tag{3-9}$$

$$u_{min} \leq u_k \leq u_{max} \ \forall k > 0 \tag{3-10}$$

Se pueden compactar las expresiones en forma matricial

$$A_x x_k \leq b_x \ \forall k \in [1, N] \tag{3-11}$$

$$A_u u_k \leq b_u \ \forall k \in [0, N-1] \tag{3-12}$$

Construimos las matrices

$$\hat{A}_x = \begin{bmatrix} A_x & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & A_x & 0 \\ 0 & 0 & 0 & A_x \end{bmatrix}, \quad \hat{b}_x = \begin{bmatrix} b_x \\ \vdots \\ b_x \\ b_x \end{bmatrix}, \quad \hat{A}_u = \begin{bmatrix} A_u & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & A_u & 0 \\ 0 & 0 & 0 & A_u \end{bmatrix}, \quad \hat{b}_u = \begin{bmatrix} b_u \\ \vdots \\ b_u \\ b_u \end{bmatrix}$$

Desarrollamos las desigualdades para expresarlas en función de *X* y *U*

$$\hat{A}_u U \leq \hat{b}_u \tag{3-13}$$

$$\hat{A}_x X \leq \hat{b}_x \tag{3-14}$$

Para expresar todas las restricciones en función de la misma variable, *U*, se sustituirá la expresión 3-2 en la 3-14.

$$\hat{A}_x X \leq \hat{b}_x \rightarrow \hat{A}_x(G_x x_0 + G_u U + G_w W) \leq \hat{b}_x \rightarrow \hat{A}_x G_u U \leq \hat{b}_x - \hat{A}_x G_x x_0 - \hat{A}_x G_w W \tag{3-15}$$

Agrupamos todo en una misma desigualdad

$$\begin{bmatrix} \hat{A}_x G_u \\ \hat{A}_u \end{bmatrix} U \leq \begin{bmatrix} \hat{b}_x - \hat{A}_x G_x x_0 - \hat{A}_x G_w W \\ \hat{b}_u \end{bmatrix} \rightarrow \hat{A}U \leq \hat{b} \tag{3-16}$$

Se expresan las restricciones en función de *U* porque es la variable de control, y la que se minimizará en el problema de optimización a resolver, que queda de la siguiente forma

$$\frac{1}{2} U^T H U + F U, \qquad s.a. \ \hat{A}U \leq \hat{b} \tag{3-17}$$

Tambien podemos resolver el problema anterior con la función *quadprog*, pasandole como argumento las matrices $\hat{A}$ y $\hat{b}$ además de *H* y *F*.

$$U = quadprog(H, F, \hat{A}, \hat{b})$$

Para observar la programación y el funcionamiento del controlador se propone un ejemplo sencillo de implementación en un sistema unidimensional, el código empleado puede consultarse en el anexo A.

Ejemplo 3-1. *Implementación de la estrategia MPC en Matlab*

Se propone el control predictivo del siguiente sistema con perturbaciones

$$x_{k+1} = {}^{1}/_{2} \cdot x_k + {}^{1}/_{2} \cdot u_k - w_k$$

Con condiciones iniciales

$$x_0 = 6$$

Horizonte de predicción

$$N = 5$$

Sujeto a restricciones de estado x_k y entrada u_k

$$-10 \leq x_k \leq 10 \qquad -5 \leq u_k \leq 5$$

Se ha formulado el algoritmo del controlador en Matlab, simulando la respuesta del sistema en un bucle de control de 100 iteraciones. Se puede observar en la siguiente figura cómo el controlador lleva a nuestro sistema desde el estado inicial hacia cero, contrarrestando el efecto de las perturbaciones para tratar de mantener al sistema entorno a este valor. Si no se hubiese introducido en el modelo del sistema el efecto de las perturbaciones no observaríamos ruido alrededor de cero; el estado se anularía y se mantendría continuo.

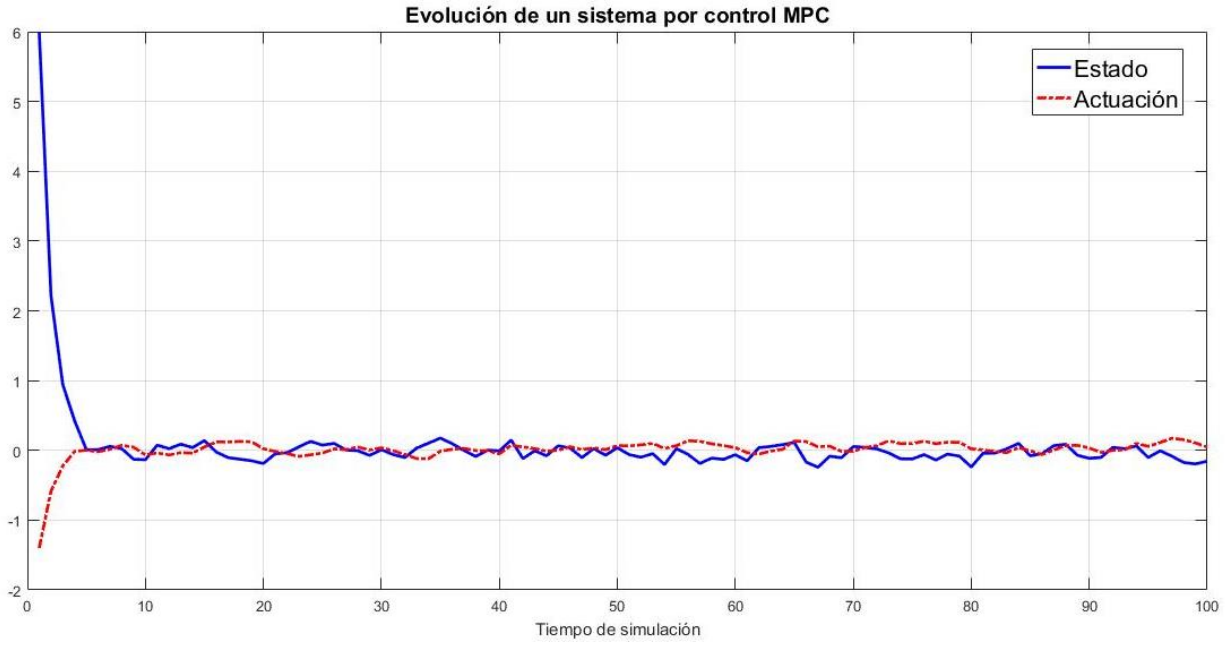

Figura 3-2 Evolución del sistema por control MPC

4 SOLUCIÓN AL PROBLEMA MPC IMPLÍCITO

Una vez explicada la estrategia seguida por el controlador predictivo, así como sus ecuaciones generales, se procederá a la particularización de estas para la resolución del problema que se propone en este proyecto; el telecontrol de un drone comercial.

4.1 Particularización del problema

El caso que se aborda cuenta con un vector de estados de dimensión 12, conteniendo la posición en los tres ejes, la orientación según los ángulos de Euler y las velocidades lineales y angulares del drone para cada instante de tiempo. El vector de entrada al sistema, que envía a los motores el empuje y los pares necesarios para llevar al drone a la referencia deseada, es de dimensión 4. Ambos vectores han sido previamente detallados en el apartado del modelo dinámico del drone.

El objetivo del control será llevar al sistema al estado de referencia deseado, para esto se le introducirá al controlador un vector de dimensión 12 que contenga la diferencia entre el estado en el que se encuentra el sistema y la referencia especificada. El MPC se encarga de llevar a cero el sistema, por lo que si se introduce como vector una resta en la que el vector de referencias permanece constante para un determinado periodo de tiempo, el estado del sistema irá tendiendo hacia la referencia para anular esta diferencia.

Como bien indica el nombre del controlador, es preciso partir de un buen modelo del sistema. En este caso, dentro del diagrama de Simulink donde se realizan todas las pruebas o simulaciones se encuentra un bloque que contiene el modelo dinámico del drone utilizado, sacado del Toolbox de robótica de Peter Corke. Será necesario linealizar este bloque para obtener el modelo linealizado de la planta de trabajo; para ello seleccionamos la opción *Linaer Analysis >Linearize block* dentro del bloque *Drone_Dynamics*.

Del análisis anterior se obtienen las matrices que relacionan el estado del sistema en un instante con el estado y la entrada inmediatamente pasados, $A_{12\times12}$ y $B_{12\times4}$ respectivamente. Estas hacen referencia al modelo continuo del sistema, expresado de la siguiente forma:

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) \tag{4-1}$$

El controlador MPC se implementa de forma discreta, por lo que no podemos utilizar directamente las matrices obtenidas del análisis anterior al ser en tiempo continuo; habrá que discretizarlas para contar con el modelo discreto linealizado del sistema, que sí será válido para el tipo de control elegido. La elección del tiempo de muestreo para el proceso de discretización también será influyente en nuestro proceso de control; tiempos muy pequeños generan imprecisiones en el sistema, ya que se partió con un periodo de muestreo de 5ms, como el tiempo en el que se ejecuta el controlador, pero la respuesta no fue favorable. Finalmente, tras muchas pruebas, se toma como periodo de discretización $T_s = 40ms = 0.04s$, y se realiza la conversión aplicando las siguientes ecuaciones:

$$Ad = e^{A\cdot\ Ts} \tag{4-2}$$

$$Bd = (e^{A\cdot\ Ts} - I) \cdot A^{-1} \cdot B \tag{4-3}$$

De esta manera se consigue pasar del modelo del sistema continuo mostrado en la ecuación 4-1 al modelo discreto de la 4-4.

(4-4)

$$x_{k+1} = Ad \cdot x_k + Bd \cdot u_k$$

Además, se podrá introducir un término más a la ecuación en espacios de estados, que modele las perturbaciones del sistema D y w_k. Se comenzará suponiendo el efecto de las perturbaciones nulo, y posteriormente se modelarán de forma aleatoria.

(4-5)

$$x_{k+1} = Ad \cdot x_k + Bd \cdot u_k + D \cdot w_k$$

Las matrices del modelo discreto quedarían de la siguiente forma:

$$Ad = \begin{bmatrix} 1 & 0 & 0 & 0 & -0.008 & 0 & 0.040 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0.008 & 0 & 0.040 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0.040 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.040 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0.001 & 0 & 0 & 0 & 0.039 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -0.001 & 0 & 0.038 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.392 & 0 & 0.997 & 0 & 0 & 0 & -0.002 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.392 & 0 & 0.997 & 0 & 0.002 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.008 & 0 & -0.039 & 0 & 0.916 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.006 & 0 & 0.029 & 0 & 0 & 0 & 0.937 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.995 \end{bmatrix}$$

$$Bd = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.0002 & -0.0002 & -0.0002 & 0.0002 \\ 0.0003 & 0.0003 & -0.0003 & -0.0003 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.0004 & 0.0004 & -0.0004 & 0.0004 \\ 0.0161 & 0.0161 & -0.0161 & -0.0161 \\ 0.0121 & -0.0121 & -0.0121 & 0.0121 \\ -0.0005 & -0.0005 & -0.0005 & -0.0005 \end{bmatrix}$$

4.1.1 Matrices de ponderación

Una vez obtenido el modelo del sistema, para sintonizar el controlador es necesario definir las matrices de pesos Q y R que ponderarán el seguimiento de referencias y el esfuerzo de control respectivamente en la función de costes. Estas matrices son diagonales y con elementos positivos que ponderan la importancia de la variable a controlar que corresponda, es decir, si priorizamos el control del estado del eje Z antes que el del eje X, el peso asignado al eje Z deberá ser mayor que el del eje X.

Los elementos de la diagonal se calcularán mediante una relación entre el valor máximo que puede tomar la variable y el peso de ponderación de dicho parámetro, ambas al cuadrado, lo que se conoce como la ley de Bryson. Se debe asegurar que la suma del cuadrado de todos los pesos debe ser igual a uno, por lo que se establecerá la ponderación de cada variable y posteriormente se normalizará para asegurar esta condición:

(4-6)

$$\sum_{i=1}^{n_x} {\alpha_i}^2 = 1$$

La matriz *R*, que pondera el esfuerzo de control, no tendrá distintos pesos porque los cuatro motores tienen la misma importancia a la hora de controlar el sistema.

La ley de Bryson establece la siguiente metodología de construcción de las matrices de pesos:

(4-7)

$$Q = \begin{bmatrix} \frac{{\alpha_1}^2}{{x_{1_{max}}}^2} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \frac{{\alpha_{n_x}}^2}{{x_{n_{x_{max}}}}^2} \end{bmatrix} \in \mathbb{R}^{n_x \times n_x} \text{ para } n_x = 12$$

(4-8)

$$R = \begin{bmatrix} \frac{1}{{u_{1_{max}}}^2} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \frac{1}{{u_{n_{u_{max}}}}^2} \end{bmatrix} \in \mathbb{R}^{n_u \times n_u} \text{ para } n_u = 4$$

El ajuste de estos pesos se hace iterativamente, variando los siete valores y observando cómo afectan a la respuesta del sistema tanto en control de altura como en seguimiento de referencias; se ha escogido un par de matrices que se han considerado óptimas para ambos tipos de control. La importancia que se le atribuye al seguimiento de referencias y al esfuerzo de control se establece mediante el parámetro ρ, en este proyecto se priorizará el primer caso, por lo que los elementos de la matriz *Q* serán mayores que los de *R*.

A continuación se presentan dos tablas con los valores máximos de las variables al igual que la ponderación de los estados. Se puede observar que se prioriza el control de altura y el de orientación frente al de posición en X y en Y, ya que las referencias que se le van a introducir al sistema serán en el eje Z, yaw, pitch y roll.

VARIABLE DE ESTADO	VALOR MÁXIMO
Posición (X, Y, Z)	1.5
Orientación (Yaw, Pitch, Roll)	0.3
Velocidad lineal	1.0
Velocidad angular	1.0
Motores	500.0

Tabla 4-1 Valores máximos de los parámetros

VARIABLE	VALOR PONDERACIÓN
Posición eje X	0.0128
Posición eje Y	0.0128
Posición eje Z	0.4257
Orientación (Yaw, Pitch, Roll)	0.5108
Velocidad lineal	0.0234
Velocidad angular	0.1064
ρ	0.0100

Tabla 4-2 Ponderación de los parámetros del drone

Comprobación de la ecuación 4-6

$$(2 \times 0.0128^2) + 0.4257^2 + (3 \times 0.5108^2) + (3 \times 0.0234^2) + (3 \times 0.1064^2) = 1$$

Para sintonizar este tipo de controlador, solamente hace falta ajustar las matrices de pesos y los horizontes de control y predicción, pero para el caso multivariable que nos presenta la dinámica del drone, el proceso iterativo para la obtención de las matrices de ponderación puede convertirse en un procedimiento complicado.

Las matrices de ponderación que se han considerado al obtener con ellas el mejor comportamiento del sistema tras nuestro proceso iterativo son las siguientes:

$$Q = \begin{bmatrix} 0.0001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.08 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0005 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0005 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0005 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.011 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.011 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.011 \end{bmatrix}$$

$$R = 10^{-7} \begin{bmatrix} 0.4000 & 0 & 0 & 0 \\ 0 & 0.4000 & 0 & 0 \\ 0 & 0 & 0.4000 & 0 \\ 0 & 0 & 0 & 0.4000 \end{bmatrix}$$

Para el caso del drone, se ha priorizado el control de posición al de velocidad, por lo que los términos de la matriz *Q* referentes a las velocidades lineales son más pequeños que los del control de posición. Se ha ponderado con bastante fuerza el seguimiento de referencias para los ángulos yaw, pitch y roll, para lo cual también se han valorado las velocidades angulares por encima de las lineales. Se ha decidido hacer de esta forma porque era lo con lo que se obtenía una mejor respuesta durante el proceso iterativo de sintonización.

4.1.2 Modelo de las perturbaciones

Una de las ventajas que presenta el control predictivo frente a otras estrategias de control es la posibilidad de introducir en el sistema un término que modele las perturbaciones. Se considera perturbación toda incertidumbre que afecta en cierta medida a nuestra planta, resultando ser un parámetro que no podemos controlar. Posibles perturbaciones que se pueden encontrar en el caso estudiado son:

- Ruido y entradas no medibles.
- Errores en el modelo del sistema. En la descripción del algoritmo MPC se destacó la importancia de emplear un modelo fiable del sistema para obtener un comportamiento exitoso de este control. Al trabajar con un sistema multivariable de 12 variables de estados y 4 entradas a la planta o salidas del controlador, lo más común es que, al generar el modelo, existan discrepancias con la realidad. A esto se le puede sumar las imprecisiones resultantes de la necesidad de linealizar y discretizar el sistema.

La matriz *D*, que establece la relación entre el estado del sistema y las perturbaciones de este, se va a modelar como una matriz columna con tantas filas como estados presentes; para el caso estudiado $D \in \mathbb{R}^{12\times 1}$. Cuando no se quiera observar el efecto de las incertidumbres sobre el sistema esta matriz se anulará. En caso contrario, la matriz *D* ha sido elegida tras un estudio iterativo en el que se han ido ponderando los términos de esta matriz y observando el efecto en la respuesta del sistema. Observando el orden de magnitud de los términos de las matrices del sistema discreto se ha escogido el factor *0.01* para que multiplique a la matriz unidad; cuanto mayor fuese ese factor mayor, crecería la influencia de las perturbaciones sobre el sistema.

$$D = 0.01\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\end{bmatrix}$$

La matriz *W* modela las perturbaciones, y se utiliza dentro de la función de coste a minimizar, por lo que tiene una dimensión de $N \cdot n_w \times 1$ siendo $n_w = 1$ el número de perturbaciones por estado, correspondiendo este tamaño al número de columnas de la matriz *D*. Como ocurría con la matriz *D*, se anulará *W* cuando no se quiera estudiar el efecto de las incertidumbres.

Las perturbaciones, según *W*, se generarán de manera aleatoria utilizando una herramienta que nos proporciona Matlab para obtener distribuciones normales según una determinada media y varianza aleatoriamente. De nuevo, como con la matriz *D*, se han escalado estos valores con un factor de *0.1* para ajustar la influencia sobre el comportamiento del sistema.

Se adjunta el trozo de código del anexo B donde se genera esta matriz.

Ejemplo 4-1. *Matriz de perturbaciones*

```
Disturb= 0.1*abs(normrnd(0.25,0.5,100+N,1));
k=randi([1 100],1);
W=Disturb(k:k+N-1)+0.1*abs(normrnd(0,0.2,N,1));
```

Por ejemplo, una posible matriz aleatoria para para $N = 10$.

$$W = \begin{bmatrix} 0.2207 \\ 0.1906 \\ 0.0639 \\ 0.1960 \\ 0.0638 \\ 0.0506 \\ 0.1000 \\ 0.0187 \\ 0.0429 \\ 0.1576 \end{bmatrix}$$

4.1.3 Horizonte de predicción

El control MPC es una estrategia de horizonte deslizante en la que se ha elegido un horizonte de control y predicción $N = 10$, esto determinará el tamaño de las matrices del sistema; que se verá aumentado ante un incremento de este horizonte. A mayor dimensión matricial mayor será el coste computacional, por lo que no debe elegirse un horizonte muy largo. Se recomienda que las matrices que permanezcan constantes, como en el caso de *Gx*, *Gu* y *Gw*, se calculen de forma previa a la implementación del código del controlador. En el caso multivariable en el que nos encontramos la matriz de mayor dimensión tiene 120 filas y 120 columnas; el cálculo mediante bucles de los elementos de una matriz de este tipo supone un gran tiempo de cómputo.

A continuación se presentan las dimensiones de las principales matrices del problema estudiado:

$$G_x \in \mathbb{R}^{120 \times 120}, \qquad G_u \in \mathbb{R}^{120 \times 40}, \qquad G_w \in \mathbb{R}^{120 \times 10}$$

$$\hat{Q} \in \mathbb{R}^{120 \times 120}, \qquad \hat{R} \in \mathbb{R}^{40 \times 40}$$

4.2 Resolución Matlab/Simulink

Como se ha explicado en este documento, el toolbox desarrollado por el MIT junto con la empresa Parrot del drone utilizado para el entorno de Matlab/Simulink permite diseñar controladores y simular la respuesta del sistema ante esta entrada gracias a la introducción de los parámetros dinámicos de nuestro drone.

Se diseñará el controlador predictivo dentro del bloque *DroneRS_Compensator* que consta de dos bloques; uno se encarga de estimar los parámetros del drone gracias a la simulación de los valores proporcionados por los sensores y el otro implementa el controlador.

La salida del primer bloque consiste en una estimación del vector de estados de nuestro sistema, *states_estim*, que es recibida por el segundo bloque; el de control, junto con las referencias de posición y orientación o actitud del drone. Todas las referencias se agruparán en un vector, *states_refout*, que tendrá como dimensión el número de variables de estado. La diferencia de estos dos vectores de estado corresponderán al error que estamos cometiendo al tratar de llevar nuestro sistema hacia la referencia deseada, lo que constituye el vector *state_error*. Este vector será la entrada a la función que calcula la acción de control a través del algoritmo de control predictivo basado en modelo, el controlador tratará de anular el error consiguiendo de esta manera aproximar el estado de mi sistema a la referencia deseada.

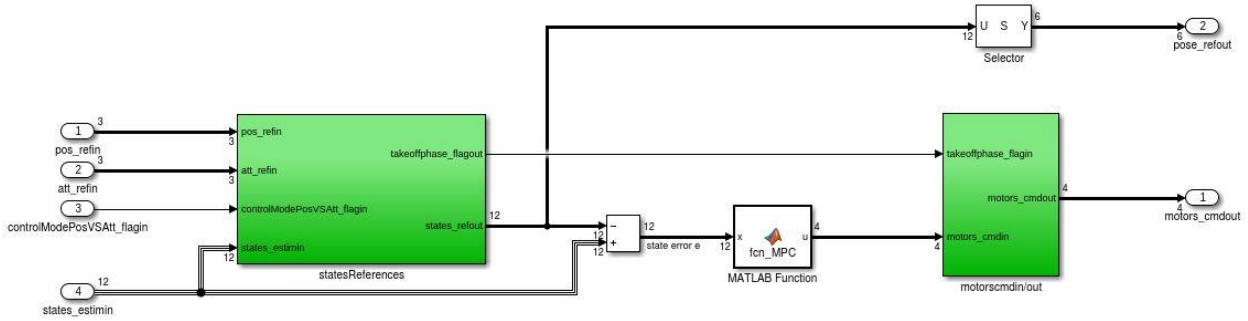

Figura 4-1 Diagrama simulink del bloque de control

El algoritmo del controlador se implementa en un bloque de Simulink que permite hacer referencia a una función escrita en Matlab, en este caso se le ha atribuido el nombre *fcn_MPC* tal y como se puede observar en el diagrama de bloques superior. El código que contiene esta función se puede encontrar en el Anexo B.

Hay que tener en cuenta varios puntos para generar la función que se utilizará en Simulink:

- No se puede utilizar, por ejemplo, la orden *'load'* de Matlab para cargar parámetros como las matrices que definen nuestro sistema; en este caso se tendrán que introducir directamente en el código.
- Se recurre en el código a funciones como *quadprog;* para resolver el problema de programación cuadrática y *c2d;* para pasar de un sistema continuo a uno discreto, que son conocidas como funciones extrínsecas en el entorno de Matlab. Para la simulación, tendremos que incorporar al principio de nuestro código la función *coder.extrinsic(...)* y colocar entre paréntesis la función extrínsica que se va a utilizar para que durante la generación de código no se incluya el código interno de estas funciones.

Existen ciertas discrepancias entre el código adjunto del Anexo B y el implementado, pero son diferencias a nivel de optimización del tiempo de cómputo. En el código real no se realiza la conversión del sistema continuo al discreto, sino que se le introducen directamente las matrices discretas del sistema, por lo que la discretización se realiza de forma previa. Las matrices de grandes dimensiones que permanecen invariantes para cualquier iteración se introducen directamente; sin cálculo mediante bucles ni fórmulas, como las matrices de pesos, ya que esto ralentiza el proceso.

El último bloque del diagrama que se muestra al inicio de esta página se encarga de realizar la conversión de la acción de control óptima que ha sido calculada por la estrategia predictiva, para traducirla a los comandos de referencia que se les darán a los motores.

5 SIMULACIÓN MPC IMPLÍCITO

5.1 Representación de resultados

A continuación se exponen los resultados de tres experimentos distintos, en los que se ha variado el modelo de la perturbación del sistema. Como se ha mencionado anteriormente, se han modelado las perturbaciones con una matriz *D* que pondera al vector aleatorio *W;* que modela las perturbaciones de forma distinta para cada instante de muestreo. En el primer caso, se ha estudiado el sistema sin perturbación alguna, a diferencia de los dos casos siguientes. En el segundo las perturbaciones tenían una mayor influencia en el sistema dada la escala elegida para la matriz *D.* Se decidió finalmente modelar el sistema con $D = 0.01$, ya que era el valor que daba resultados más acordes a los demás componentes del estado, es decir, el producto $D \cdot W$ era del mismo orden de magnitud que $A \cdot X$ o $B \cdot U$.

En el control de altura se le pide al sistema que varíe su posición en el eje Z:

Intervalo (s)	Eje Z (m)
[1, 5]	1
[5, 10]	1.5
[10, 15]	0.5
[15, 20]	1

Tabla 5-1 Referencias altura

En el control de orientación se le pide al sistema que varíe su posición en los tres ángulos de Euler:

Intervalo (s)	Yaw (rad)	Pitch (rad)	Roll (rad)
[1, 2]	0	0	0
[2, 5]	0	0.04	0
[5, 8]	0	0	0
[8, 11]	0.2	0	0
[11, 14]	0	0	0
[14, 17]	0	0	-0.04
[17, 20]	0	0	0

Tabla 5-2 Referencias orientación

En el control de orientación, existen determinados intervalos de tiempo en los que la referencia para todos los ángulos de Euler es nula, con el fin de que el sistema se estabilice.

5.1.1 Sistema sin perturbación: D = 0, W=0

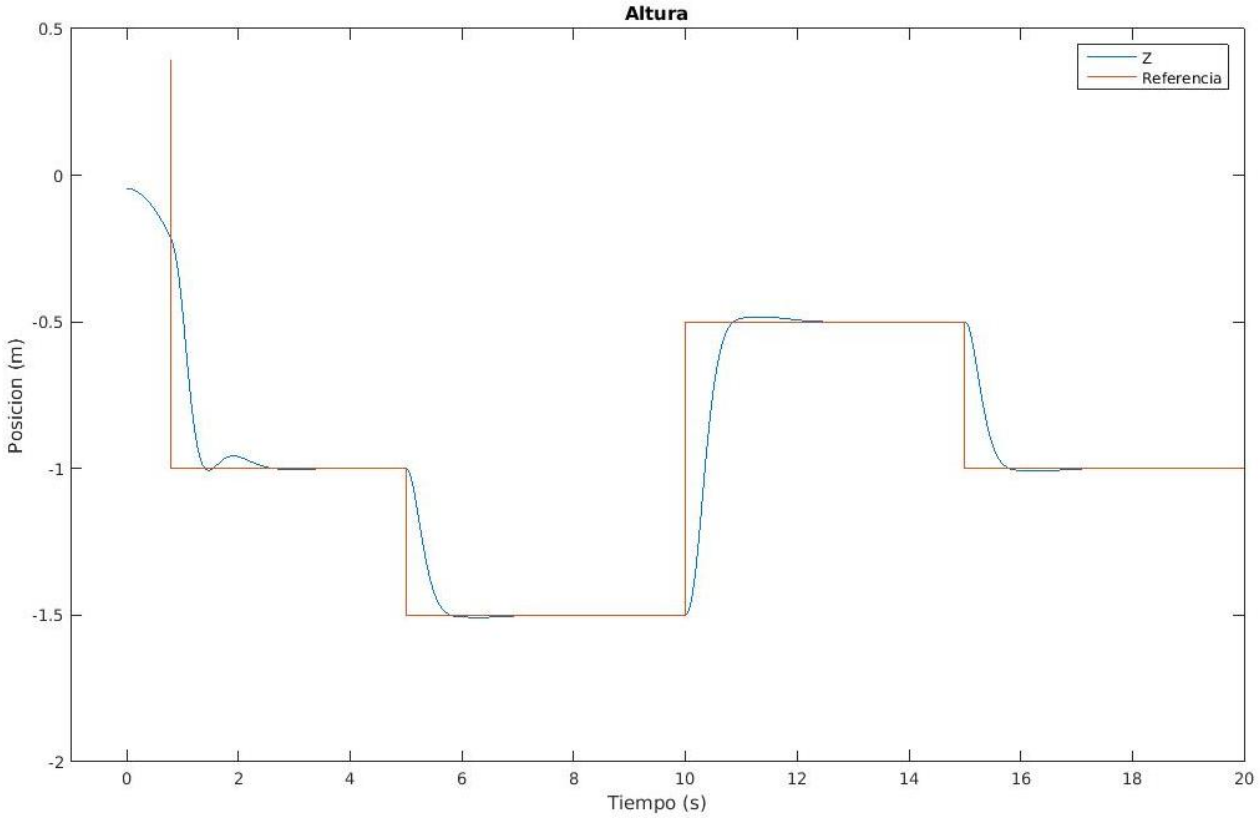

Figura 5-1 Altura MPC sin perturbación

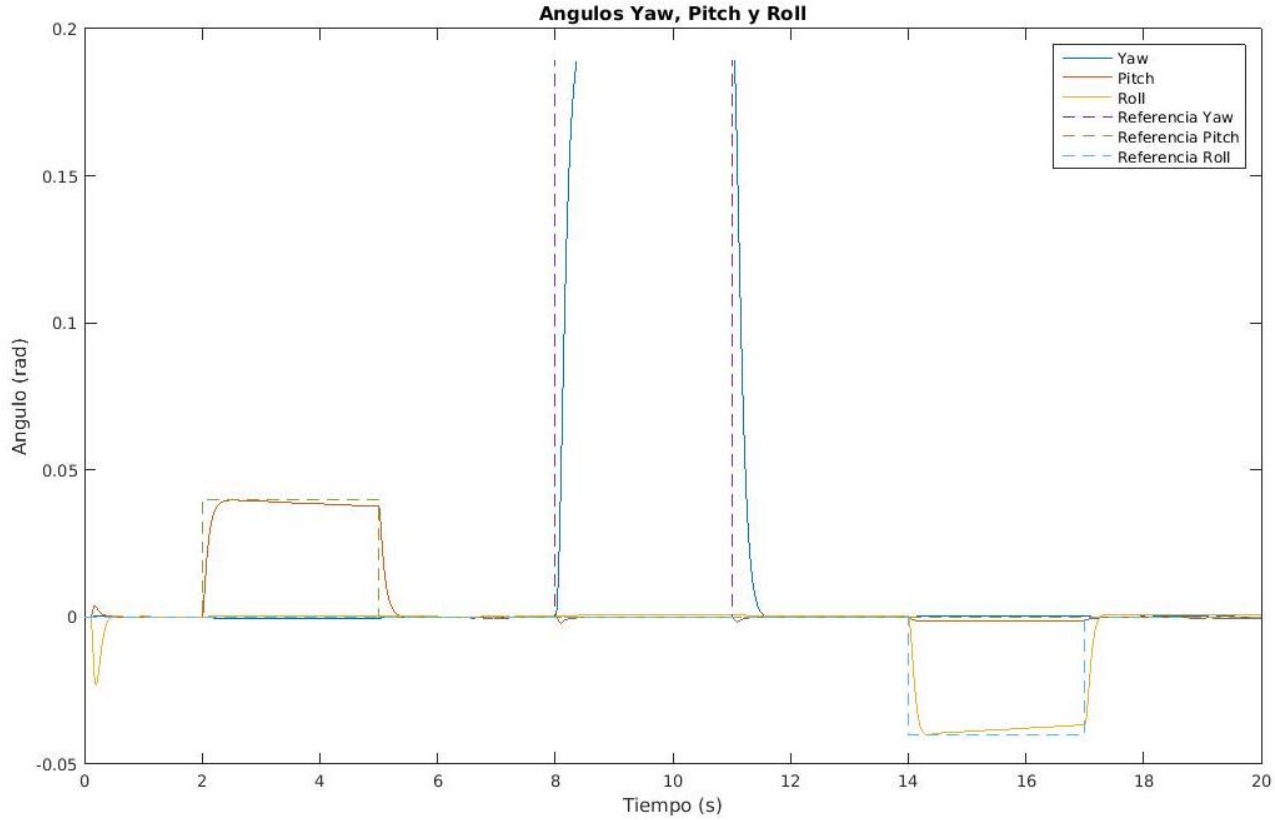

Figura 5-2 Orientación MPC sin perturbación

5.1.2 Sistema con perturbación: D = 0.1, W≠0

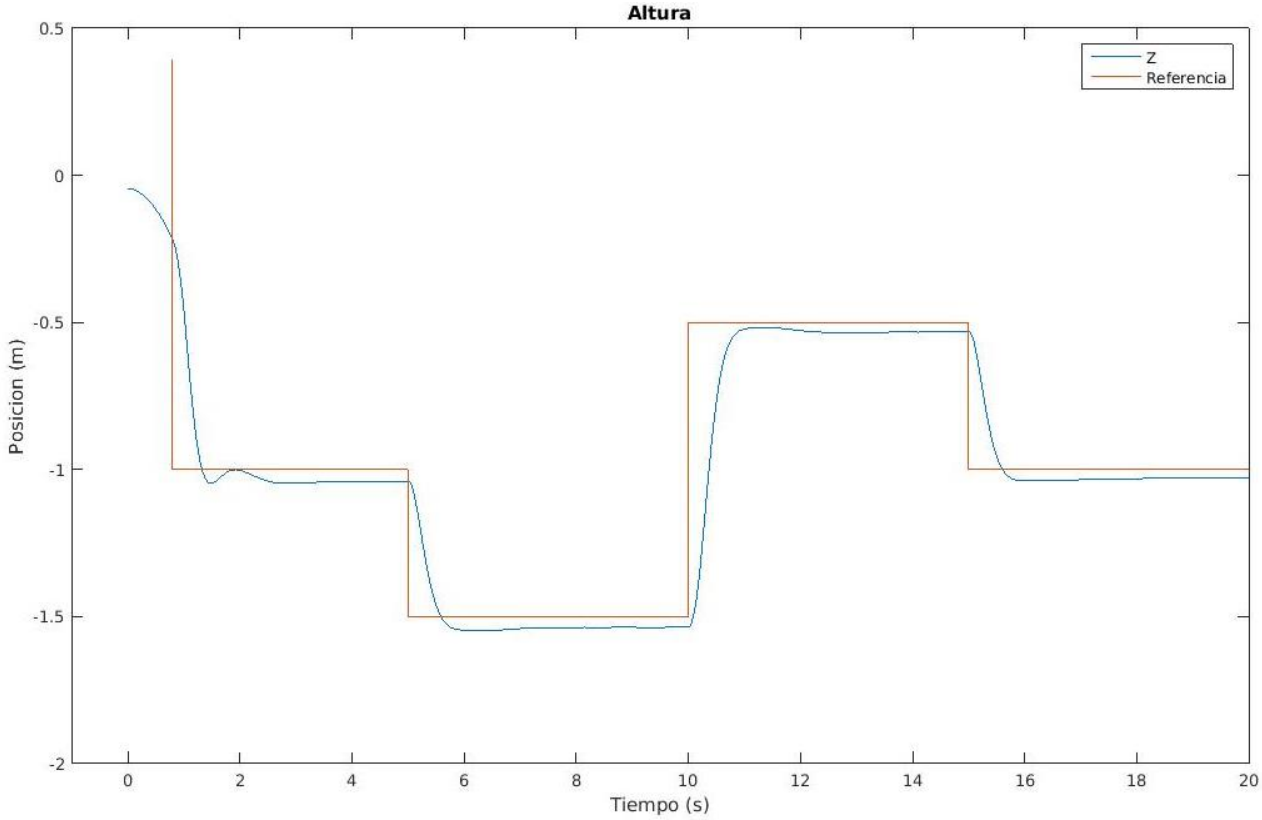

Figura 5-3 Altura MPC perturbacion D=0.1

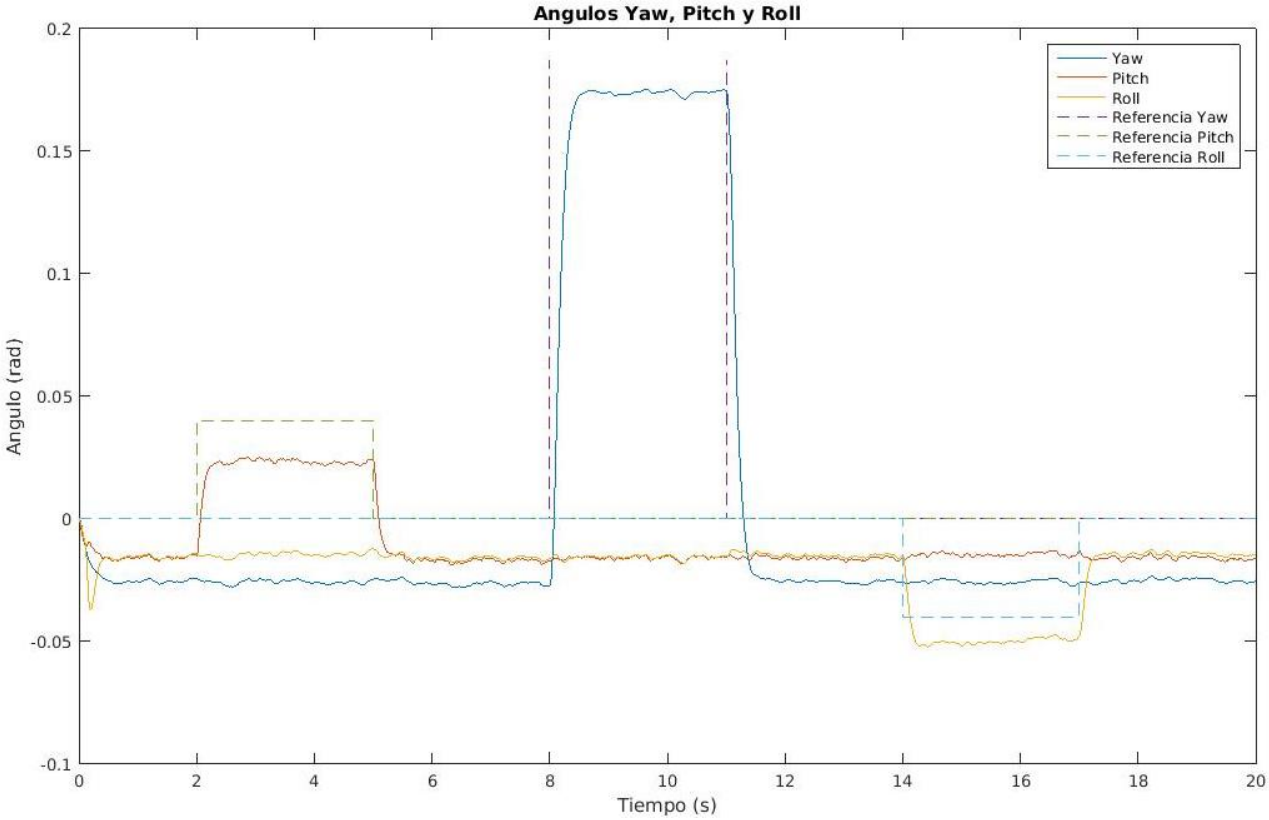

Figura 5-4 Orientación MPC con perturbación D=0.1

5.1.3 Sistema con perturbación: D = 0.01, W≠0

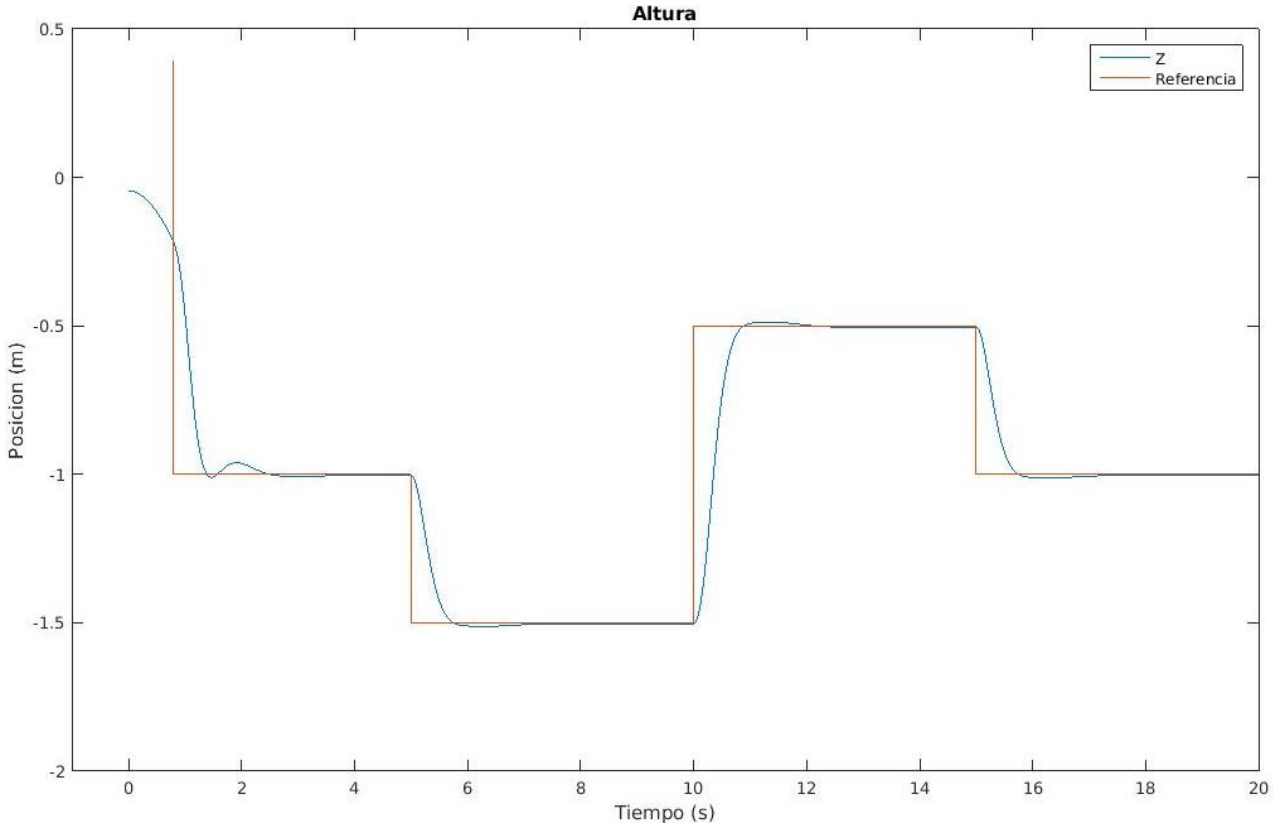

Figura 5-5 Altura MPC perturbacion D=0.01

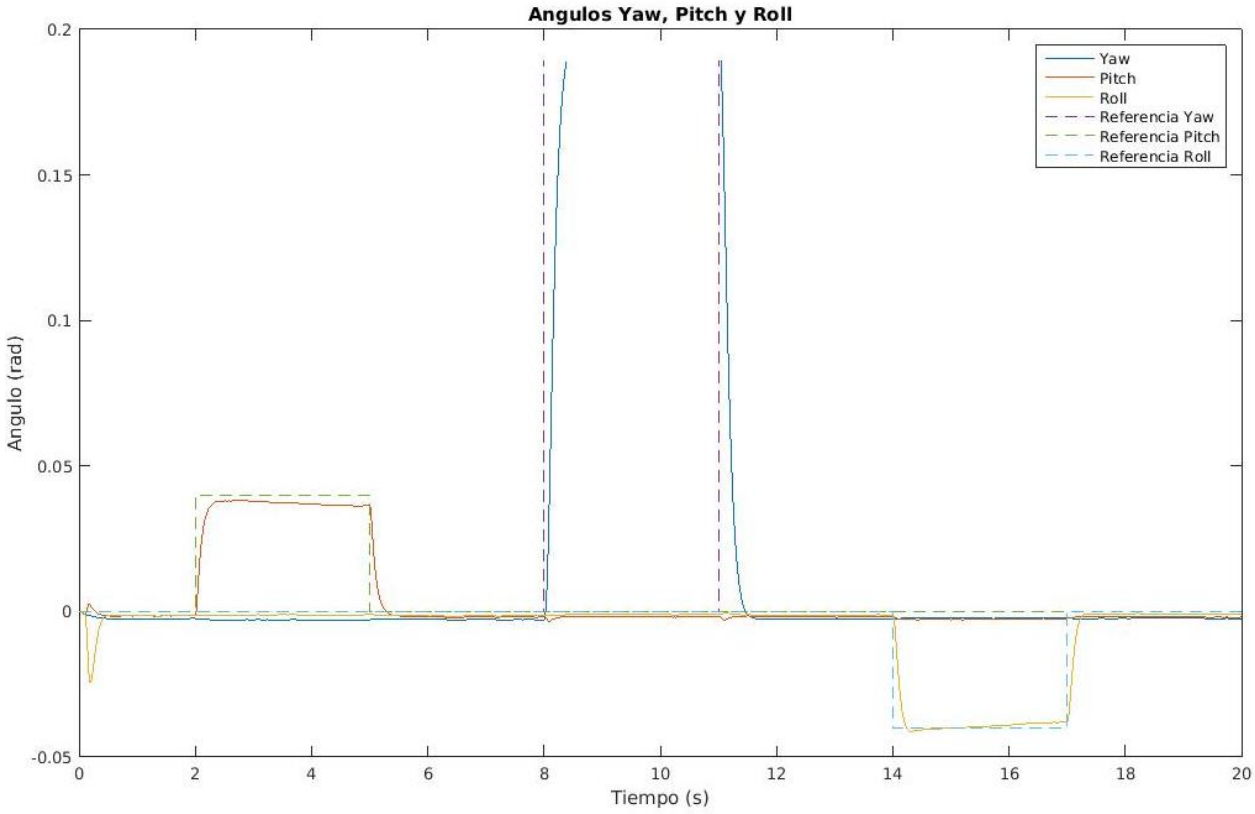

Figura 5-6 Orientación MPC con perturbación D=0.01

5.2 Comentarios

Estudiando la acción del controlador ante distintas influencias de perturbaciones en el sistema, se puede observar que, para el caso en el que $D = 0.01$, existe ruido en el sistema cuando trata de alcanzar la referencia, ya que no será una línea constante como en el primer caso, pero no se aleja tanto de la referencia como cuando $D = 0.1$. Recordando la expresión del estado del sistema en espacios de estados (5.1), se ve que si el tercer producto es muy grande en comparación con los dos primeros, el efecto de la perturbación será más notable en el vector de estados. Esto es lo que sucede cuando $D = 0.1$, por lo que, observando el comportamiento del sistema y también el orden de magnitud de los elementos de *Ad* y *Bd*, se ha elegido $D = 0.01$ como mejor opción para ponderar este efecto.

(5-1)

$$x_{k+1} = Ad \cdot x_k + Bd \cdot u_k + D \cdot w_k$$

Además, se puede observar que el efecto de las perturbaciones es más notable en el control de orientación que en el de altura, ya que, en el caso más desfavorable en el que $D = 0.1$, el sistema se encuentra bastante lejos de alcanzar la referencia especificada en radianes.

Se encontraron limitaciones al formular las restricciones del problema, ya que no era posible establecer unos valores máximos y mínimos que comprendiesen un intervalo muy estrecho. Es decir, aunque el sistema para las simulaciones realizadas no alcanzase una altura mayor a 1.5m ni menor a 0m, las restricciones del eje Z se tendrían que definir como mínimo $-1 < z < 2$ como se puede observar en el código del Anexo B. Por esto no se han representado en el proyecto simulaciones restringiendo más el intervalo, ya que Simulink paraba la ejecución debido a un error procedente de las restricciones impuestas a *quadprog*.

5.3 Comparación de controladores

Del trabajo del alumno Emilio Marín, recogido en [2], se concluyó que para el controlador de asignación de polos, el sistema respondía mejor que con un control PID o LQR, por lo que compararemos a continuación el comportamiento del sistema con este controlador y con el MPC generado. Para llevar esto a cabo se le exigirán a ambos controladores las mismas referencias en altura y en orientación, que corresponden a las que el alumno llevó a cabo en su proyecto, del cual se han recuperado sus gráficas. Destacar que, para el caso MPC, se ha elegido como matriz que pondera a las perturbaciones $D = 0.01$, justificando esta elección con las conclusiones sacadas del punto anterior, en el que se estudia el efecto de la magnitud de la perturbación sobre el sistema.

En el control de altura se le pide al sistema que varíe su posición en el eje Z:

Intervalo (s)	Eje Z (m)
[1, 5]	1
[5, 9]	1.5
[9, 12]	0.5
[12, 20]	1

Tabla 5-3 Referencias altura

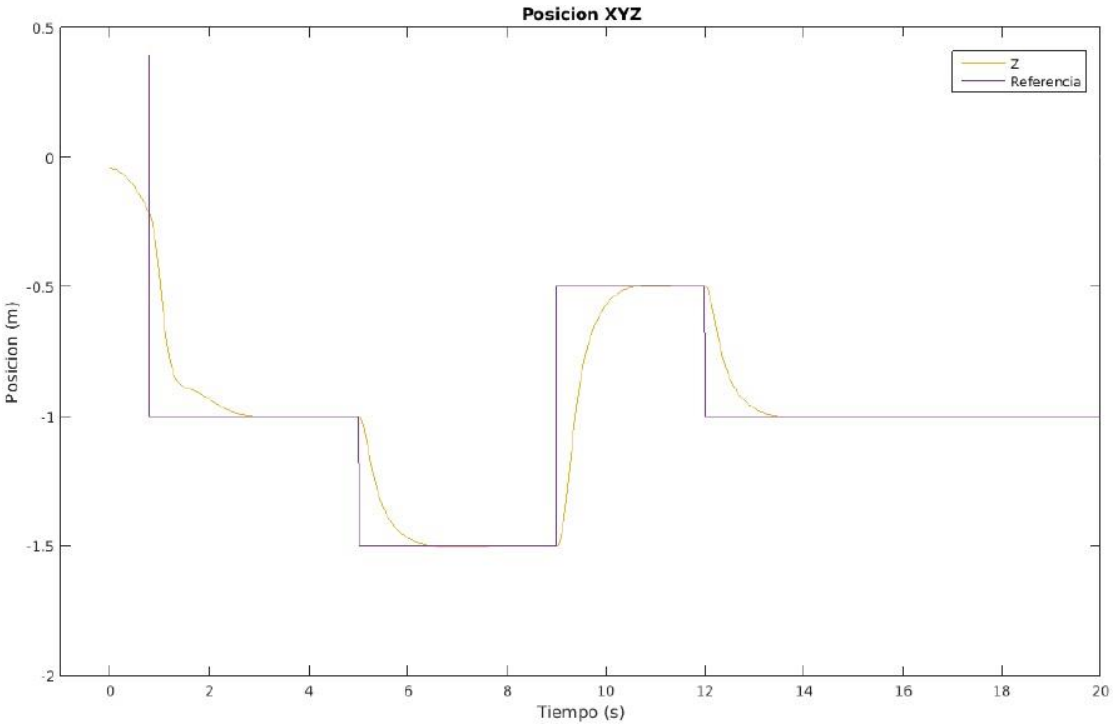

Figura 5-7 Control de altura con asignación de polos [2]

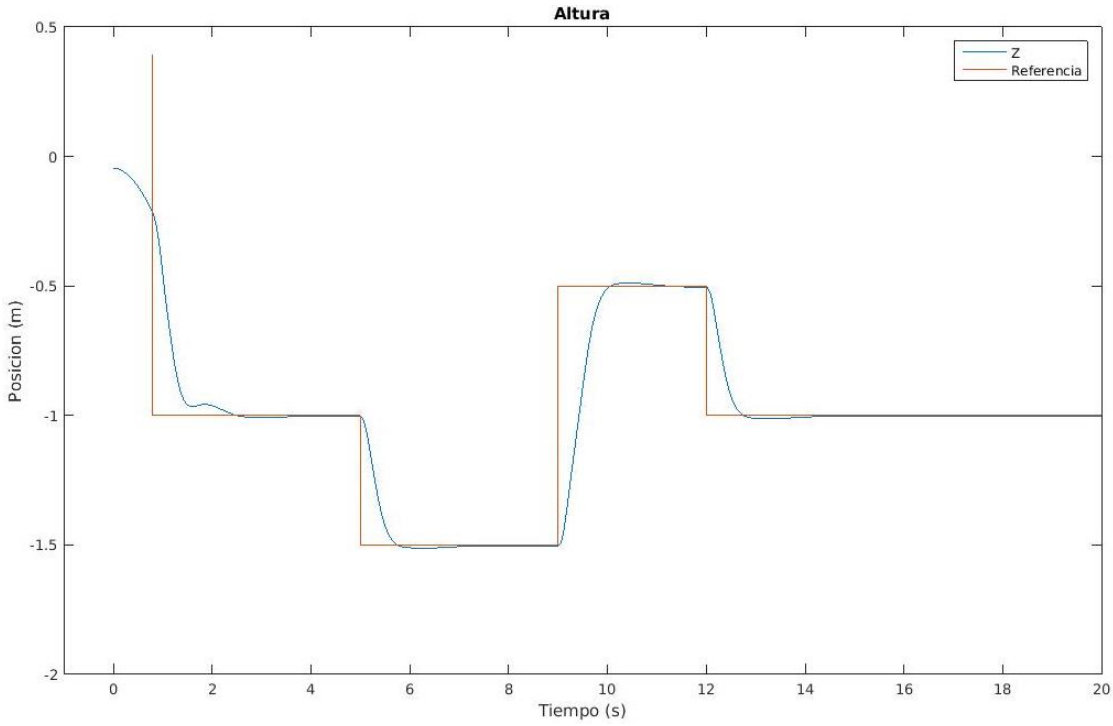

Figura 5-8 Control de altura con MPC

En el control de altura, el sistema controlado con el MPC presenta una pequeña disminución en el tiempo de subida con respecto al caso del controlador por asignación de polos, siendo esta diferencia del orden de décimas de segundo solamente. Con el MPC y las matrices de ponderación escogidas, el sistema sufre una pequeña sobreoscilación, pero poco notable. Hay que destacar que en la segunda gráfica el sistema se está modelando con el efecto de las perturbaciones, lo que supone un punto a favor del control predictivo, ya que se obtiene una buena respuesta a pesar de este fenómeno aleatorio.

En el control de orientación se le pide al sistema que varíe su posición en los tres ángulos de Euler:

Intervalo (s)	Yaw (rad)	Pitch (rad)	Roll (rad)
[1, 3]	0	0	0
[3, 6]	0	-0.03	0
[6, 9]	0	0	0
[9, 12]	0	0	0.03
[12, 15]	0	0	0
[15, 18]	0.2	0	0
[18, 25]	0	0	0

Tabla 5-4 Referencias orientación

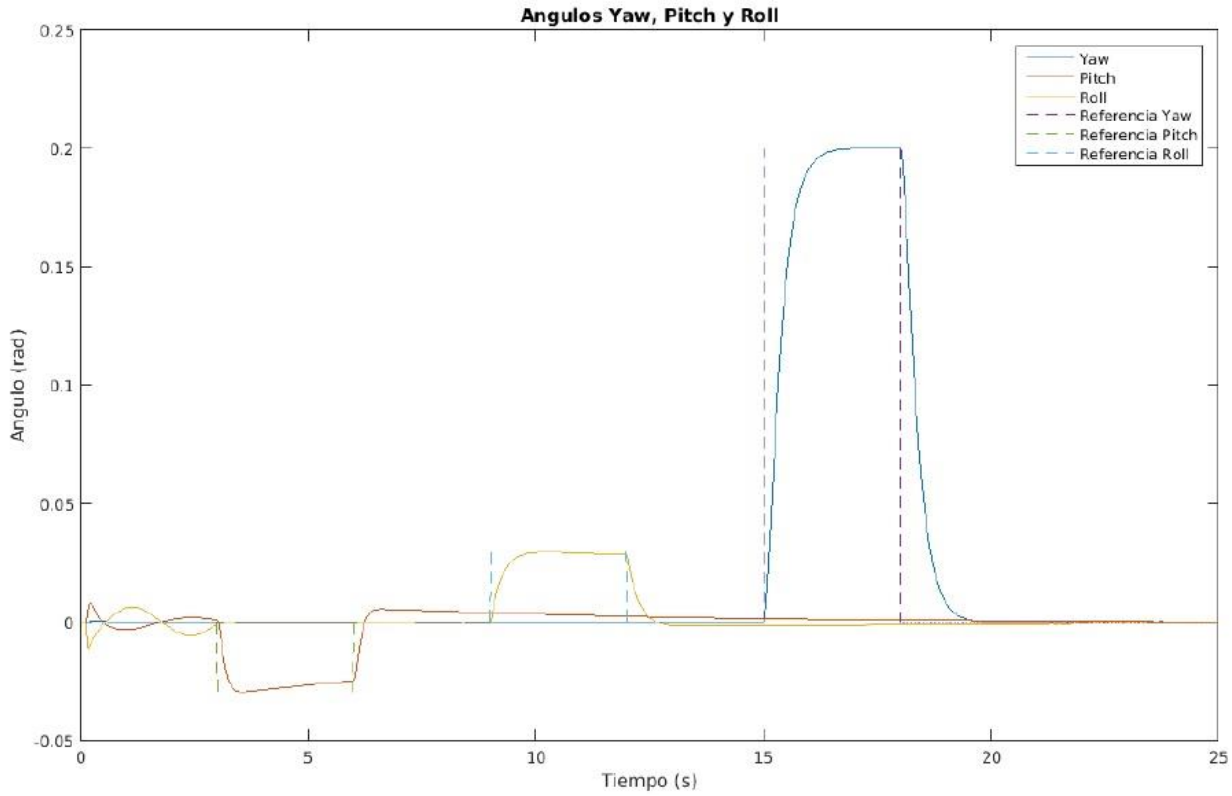

Figura 5-9 Control de altura con asignación de polos

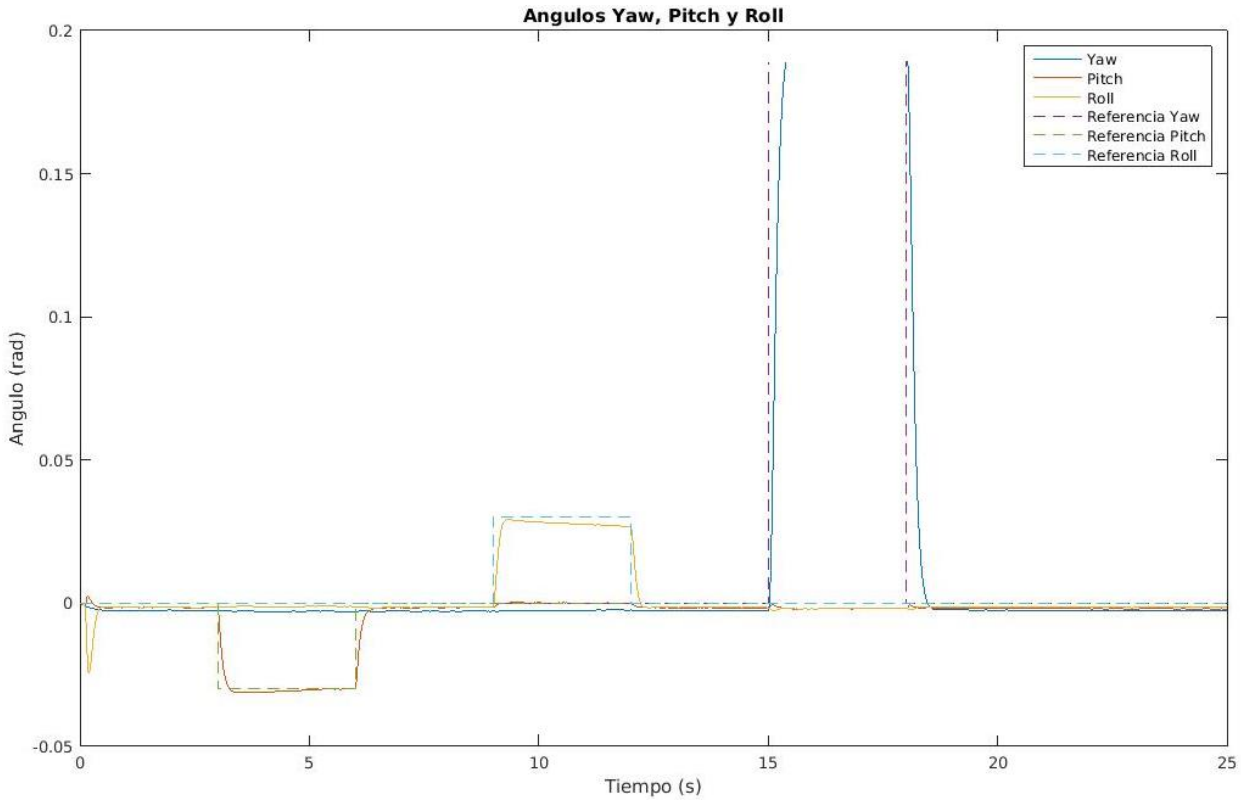

Figura 5-10 Control de orientación con MPC

En el estudio de la orientación según los ángulos de Euler la diferencia entre ambos controladores se hace más notable. Aun habiendo modelado el sistema con perturbaciones para el caso del control predictivo; repercutiendo esto en que el sistema se queda ligeramente por debajo de su referencia en algunos casos, se pueden observar los beneficios que proporciona la estrategia MPC. Entre ellos el tiempo de subida se disminuye considerablemente, y las oscilaciones del iniciales son menores y se estabilizan rapidamente.

6 MPC EXPLÍCITO

6.1 Problemática

Un controlador predictivo basado en modelo tradicional o implícito resuelve un problema de programación cuadrática para cada instante de muestreo (de forma *on-line*), con el objetivo de encontrar el valor óptimo de la acción de control o entrada al sistema. La resolución de este problema de optimización presenta una gran limitación en cuanto a complejidad del algoritmo y tiempo de computación, además de no ser un valor constante a lo largo de las iteraciones. En aplicaciones de tiempo real con dinámicas rápidas que necesitan un tiempo fijado de muestreo de un orden muy pequeño, como en el caso estudiado para el que se necesitan poco milisegundos, la aplicación de la estrategia MPC de forma implícita es impracticable.

La forma implícita de este controlador es válida para la simulación en el entorno de Matlab/Simulink, respondiendo satisfactoriamente para un periodo de muestreo de 5ms en el que se utilizará la herramienta *quadprog* para la resolución del problema de optimización de programación cuadrática; *QP*. A la hora de generar el código C para poder subirlo al drone no podremos utilizar esta función, ya que está definida en el entorno de Matlab, si se quiere utilizar un controlador implícito habría que incorporar un método que resuelva el problema de QP en el hardware de control. Este método de resolución o *solver* deberá ser lo suficientemente veloz para su implementación en sistemas con intervalos de muestreo cortos, y estará limitado por el tamaño de la memoria para almacenar el código y los resultados del problema de optimización.

6.2 Estrategia explícita

El planteamiento explícito de la estrategia MPC presenta las características adecuadas para su implementación en aplicaciones de tiempo real con tiempos de muestreo rápidos. Las ventajas que presenta la forma explícita frente a la implícita es el cálculo *off-line* de la ley de control con la finalidad de reducir las operaciones realizadas de forma *on-line* para ahorrar tiempo de procesamiento o cómputo y memoria, por lo que se elimina la necesidad de *solvers on-line*. Para cada instante de muestreo las operaciones necesarias se limitarán a evaluar una función; convirtiendo la ley de control en una función lineal del estado. Este método puede expresarse como una división en regiones, cada una con una ganancia asignada, aplicándose una u otra según el valor del estado del sistema.

La idea básica de este controlador es, dado los valores máximos, x_{max}, y mínimos, x_{min}, de las variables de estado, resolver previamente a la implementación del controlador el problema QP de forma *off-line* para todo el conjunto posible de estados.

$$x_{max} \leq x_k \leq x_{min}$$

Esto resultará en un número de regiones, n_r, donde para cada un se define mediante cuatro constantes; $H_i, K_i, F_i \; y \; G_i$, que constituyen una restricción para determinar si el estado pertenece a dicha región y la correspondiente ley de control que se ha de aplicar; expresándose u como una función explícita del estado x.

$$H_i x_k \leq K_i \tag{6-1}$$

$$u_k = F_i x_k + G_i \tag{6-2}$$

$$i = 1, 2, \dots, n_r$$

Ejemplo 6-1. *Visualización de la estrategia explícita*

Número de estados: $n_x = 2$

Siendo las variables de estado

$x_1 \equiv posición\ (position)$ con $-5 \leq x_1 \leq 5$

$x_2 \equiv velocidad\ (speed)$ con $-3 \leq x_2 \leq 3$

Número de regiones: $n_r = 24$

La estrategia *off-line* se encarga de calcular la ley de control que se aplicará según el valor del estado del sistema. En la imagen que se adjunta se observan distintas regiones, cada una con su propio valor de u según la x. Para cada instante de muestreo, el programa deberá comprobar en qué región se encuentra mi sistema, para así aplicar la acción de control que corresponda. Esta búsqueda de región se hará comprobando el cumplimiento de la restricción de la ecuación 6-1, lo que supone una disminución considerable del coste computacional en aplicaciones de tiempo real en comparación con la estrategia implícita.

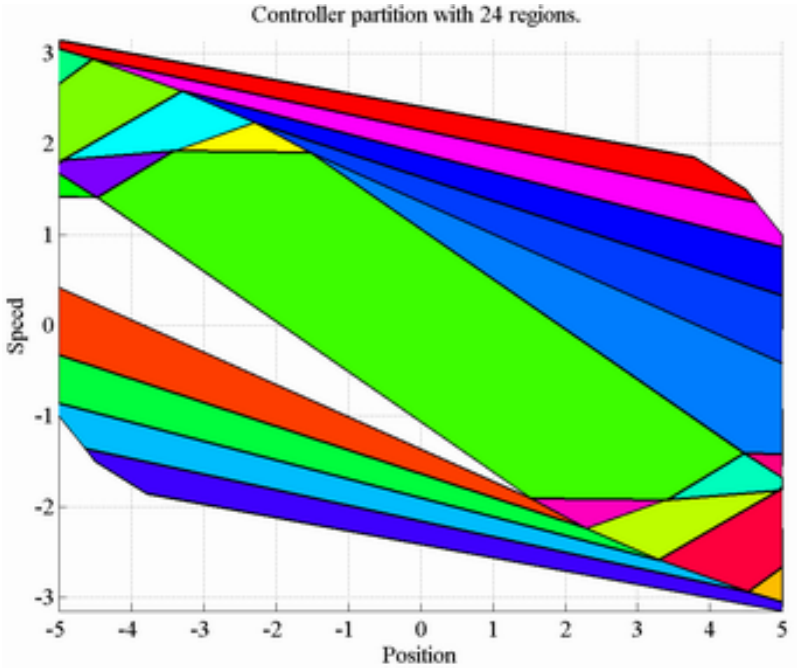

Figura 6-1 División del espacio en regiones para el MPC explícito [15]

6.3 Conversión del controlador

Para el cálculo de la estrategia explícita se emplearán distintas funciones proporcionadas por Matlab para poder generar las regiones del espacio que caracterizan a este controlador. Se pueden dividir en tres los pasos que han sido necesarios para elaborar la estrategia de control:

1) Cálculo *off-line* de las regiones del controlador.
2) Conversión de la solución al tipo de datos aceptado por Simulink.
3) Cálculo *on-line* de la estrategia de control.

Es preciso mencionar que, dadas las dimensiones del problema a tratar, se han realizado una serie de simplificaciones a la hora de modelar el sistema. Como se ha explicado con anterioridad, el vector de estados del sistema, x_k, contiene 12 variables distintas correspondientes a las posiciones en los ejes XYZ, la orientación según los ángulos de Euler, y siendo las 6 últimas las derivadas de las 6 anteriores, pero a la hora de controlar solamente estoy observando las 6 primeras. La simplificación realizada ha sido determinar que la salida, y_k, del sistema consta solamente de las 6 primeras variables de estado, ya que son las que verdaderamente se van a controlar y variar la referencia de sus valores. Esto se expresa de la siguiente forma:

$$x_{k+1} = A \cdot x_k + B \cdot u_k \tag{6-3}$$

$$y_k = C \cdot x_k + D \cdot u_k \tag{6-4}$$

Siendo

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

La necesidad de realizar alguna simplificación en el modelo surge cuando la búsqueda de las regiones en las que se divide el espacio de trabajo resultaba en un problema que no convergía en ningún número finito de soluciones tras ejecutarlo durante más de 24 horas.

Una vez conseguido un número finito de regiones se debía obtener un número razonable de estas para que el cálculo *on-line* de la acción de control (en el que se compara una desigualdad matricial para saber en qué región se encuentra el estado del sistema) fuera posible en tiempo de ejecución. El parámetro a modificar con respecto al problema implícito para disminuir el número de regiones es el horizonte de predicción y de control, que se redujo a N=5 consiguiendo 216 regiones.

6.3.1 Cálculo *off-line*

Para el cálculo de regiones se ha seguido la ayuda de Matlab que se puede consultar en [16], [17] y [18] para generar la estrategia explícita apoyándose en la función *generateExplicitMPC*. Se han tenido que seguir distintos pasos que se detallarán a continuación adjuntando el código que ha sido generado para cada uno de ellos.

El primer paso será generar un controlador predictivo implícito, que será uno de los argumentos principales de la función que transforma el controlador en uno explícito. Dadas las simplificaciones realizadas, es inmediato pensar que los parámetros del controlador que se describen en el punto 5 de esta memoria no nos serán útiles para este apartado porque ni el modelo del sistema es el mismo ni los horizontes empleados. Esto tiene como consecuencia que no se sabe a priori si con ese horizonte el sistema va a responder bien, ni tenemos sintonizadas las matrices *Q* y *R* de pesos que se calcularon a través de un largo proceso iterativo.

Para crear el MPC implícito se utilizarán las matrices del modelo continuo del sistema resultantes del análisis lineal realizado, y se pondrán en forma de espacios de estado. El tiempo de muestreo para el cálculo de la acción de control lo fijaremos igual que en el caso implícito, en 5ms. Estos parámetros, junto con los horizontes de predicción y control serán los argumentos de la función *mpc* que devuelve un objeto MPC de tipo impícito.

Código para la creación del MPC implícito

```
% SISTEMA CONTINUO
load('analisisLinear')
aux = LinearAnalysisToolProject.Results.Data.Value;
A = aux.c*aux.a*inv(aux.c);%linsys1.a;
B = aux.c*aux.b;%linsys1.b;
C=zeros(6,12);
C(1:6,1:6)=eye(6);
D = zeros(6,4);
% SISTEMA EN ESPACIOS DE ESTADOS
Plant=idss(A,B,C,D);
% INTERVALO DE CONTROL
tau=0.005;
% HORIZONTES
p=5; m=5;
% OBJETO MPC
MPCobj=mpc(Plant,tau,p,m);
```

Una vez generado el objeto MPC que corresponde al controlador implícito lo que se hará será ajustar los valores máximos y mínimos de las variables manipuladas; acción de control, y las variables de salida; posición y actitud del drone, es decir, imponer restricciones al problema.

Restricciones del problema

```
% MATRIZ Q DE PESOS
MPCobj.Weights.OutputVariables=ones(1,6);
% VALOR MAXIMO DE LOS MOTORES
MPCobj.MV(1).Max=500; MPCobj.MV(2).Max=500;
MPCobj.MV(3).Max=500; MPCobj.MV(4).Max=500;
% VALOR MINIMO DE LA SALIDA
MPCobj.OV(1).Min=-1.5; MPCobj.OV(2).Min=-1.5; MPCobj.OV(3).Min=-1.5;
MPCobj.OV(4).Min=-0.3; MPCobj.OV(5).Min=-0.3; MPCobj.OV(6).Min=-0.3;
% VALOR MAXIMO DE LA SALIDA
MPCobj.OV(1).Max=0; MPCobj.OV(2).Max=0; MPCobj.OV(3).Max=0;
MPCobj.OV(4).Max=0.3; MPCobj.OV(5).Max=0.3; MPCobj.OV(6).Max=0.3;
```

Las restricciones impuestas al implícito se tendrán que volver a introducir para el explícito. Entregándole a la función de *generateExplicitMPC* estas restricciones y el objeto MPC generado previamente procederá a la búsqueda de regiones. Existe una opción en Matlab para representar el espacio de trabajo resultante, dividido en el número de regiones que corresponda, pero solamente es valida para un problema bidimensional, por lo que no será posible visualizar la solución para este problema.

Como en el caso anterior, el objeto MPC corresponde con una estructura, de la que destacaremos el campo *PiecewiseAffineSolution* por ser una estructura que contiene las matrices *F*, *G*, *H* y *K* para cada región. Este campo se guardará en otra estructura llamada 'struct_sol' en el archivo 'solucion.mat' que será entregado a la siguiente fase.

Transformo controlador implícito a explícito

```
% RESTRICCIONES/RANGOS DE LAS VARIABLES
range=generateExplicitRange(MPCobj);
range.ManipulatedVariable.Max(:)=[500, 500, 500, 500];
range.Reference.Min(:)=[-1.5, -1.5, -1.5, -0.3, -0.3, -0.3];
range.Reference.Max(:)=[0, 0, 0, 0.3, 0.3, 0.3];
% LA OPCION polyreduction ELIMINA DESIGUALDADES REDUNDANTES
opt=generateExplicitOptions(MPCobj);
opt.polyreduction=1;
% FUNCION DEL TIPO EMPCobj=generateExplicitMPC(MPCobj, range)
EMPCobj=generateExplicitMPC(MPCobj, range, opt);
% SIMPLIFICAR PARA REDUCIR EL NUMERO DE REGIONES
EMPCreduced=simplify(EMPCobj,'exact');
% GUARDO MATRICES SOLUCION
struct_sol=EMPCobj.PiecewiseAffineSolution;
save solucion.mat struct_sol
```

6.3.2 Ajuste del tipo de dato de la solución

Conviene recordar las ecuaciones 6.1 y 6.2 que determinan respectivamente la región en el espacio en la que se encuentra mi problema en un determinado instante y su correspondiente acción de control. La acción de control calculada siempre tendrá la misma dimensión, por lo que no existirá variación en el tamaño de las matrices *F* y *G* para cada región. Esto no ocurre en las matrices *H* y *K*, ya que el tamaño de las regiones no es fijo, por lo que existirán variaciones entre las dimensiones de estas. Este último punto supuso un problema al entregarle la estructura resultado 'struct_sol' a la función de Matlab que se encarga de calcular la acción de control, ya que no aceptaba distintas dimensiones para un mismo campo de la estructura; el tamaño de struct_sol(1).H era 23x10 mientras que el de struct_sol(5).H, por ejemplo, era 25x10 y esto era inaceptable.

En el Anexo C se presenta el código generado para solventar el error de dimensiones anterior. El programa carga la estructura solución 'struct_sol' y hace una copia de esta, que será la estructura que utilicemos para realizar las modificaciones necesarias. Se calcularán las dimensiones máximas de las matrices *H* y *K* comparando el tamaño de estas a lo largo de las distintas regiones. A continuación, aquellas que no tengan la máxima dimensión se rellenarán con las filas de ceros necesarias para conseguir este tamaño. La copia de la estructura incial que ha sido modificada para homogeneizar el tamaño de las matrices *H* y *K*; 'sol', se guardará en el archivo 'dimension.mat'.

6.3.3 Cálculo *on-line*

Llegados a este punto, se ha conseguido expresar el problema *off-line* de la manera adecuada para poder pasar esta solución al bloque del diagrama de Simulink que se encarga de calcular la acción de control. Se procederá de la misma manera que en el caso implícito; utilizando un bloque de Simulink que haga referencia a una función escrita en Matlab. Las discrepancias respecto al caso anterior, que se puede recordar en la figura 1 del capítulo 4 de esta memoria (4.1), es que la función, a parte de recibir el error existente entre el estado de referencia y el estado actual del sistema, recibirá la acción de control del instante anterior. Destacar que no es posible conectar directamente la salida del bloque con una entrada de este; *u* con *uant* del diagrama inferior, ya que se obtendrá un error cuando se trate de ejecutar la simulación. Se utilizará el bloque *Delay* de la biblioteca de bloques de Simulink para introducir un pequeño retraso, de esta forma solventaremos el problema anterior.

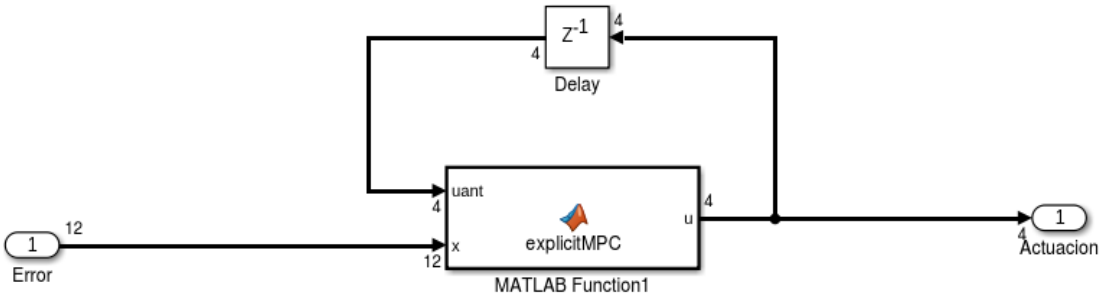

Figura 6-2 Bloque de cálculo de la acción de control explícita

A continuación se adjunta el código *on-line*, es decir, el que se tendrá que ejecutar para cada periodo de muestreo. A partir del vector de estados en *k* y la acción de control en el instante *k-1*, la función generará la actuación en *k*. La función cargará la estructura solución con dimensiones de matrices modificadas del archivo 'dimension.mat'. Señalar que esto solamente se podrá hacer en simulación, para su implementación en el drone se tendría que buscar una alternativa para subirle la estructura, ya que la orden *load* no se puede generar en un código C porque estamos haciendo referencia a un archivo del ordenador.

Las dimensiones de las matrices *F*, *G*, *H* y *K* generadas nos dan una idea de que el vector *x* al que multiplican algunas de estas tiene dimensiones 10x1, lo que significa que no corresponde con el vector de estados de nuestro sistema ya que este tiene 2 filas más. En esta función se lamará a este vector *vect_x* para distinguirlo del vector de errores que le entra como argumento. Se plantea entonces el problema de averiguar a qué 10 variables hace referencia este vector, ya que la ayuda proporcionada por Matlab para la función *generateExplicitMPC* no lo detalla. Finalmente se interpretó que, dado que se establecieron especificaciones en 6 variables de estado y en las 4 acciones de control, el vector *x* estaría formado por las 6 primeras componentes del vector de estado y las 4 componentes de la acción de control anteriores.

Se formula un bucle de control con la longitud de la estructura (número de regiones) como número máximo de iteraciones. Para cada valor del *vect_x* se comprobará la desigualdad matricial y si se cumple para un determinado valor se aplicará la acción de control correspondiente y nos saldremos del bucle.

Cálculo de la acción de control en tiempo de ejecución

```
function u = explicitMPC(uant,x)
s=load('dimension.mat');
u=zeros(4,1);
% Creo el vect_x con dimension 10x1 que contiene las 6 variables de estado
% de posicion y de actitud, y las 4 componentes de la accion de control
% anteriores
```

```
vect_x=[x(1:6); uant];
for i=1:length(s.sol)
    if norm(s.sol(i).H*vect_x)<norm(s.sol(i).K)
        u=s.sol(i).F*vect_x+s.sol(i).G;
        break
    end
end
end
```

6.4 Resultados

El resultado de la implementación de todo este código en simulación no fue el esperado, ya que el controlador no ejercía bien su función y el comportamiento del sistema era divergente. Se estudiaron distintas posibles fuentes de errores para intentar mejorar el comportamiento, aunque ninguna resultó ser exitosa:

- Fallos en el modelo del sistema

  Inicialmente, en las primeras simulaciones, se estaba empleando el modelo discretizado del sistema para crear el objeto MPC. Como el comportamiento del sistema era muy similar al que se obtuvo tratando de sintonizar el MPC implícito con un sistema continuo cuando se tendría que utilizar uno discreto, se le pasó al objeto MPC el sistema en continuo, pero esto no mejoró el comportamiento.

- Incorrecta iterpretación del vector *x*

  Como ya se ha hecho referencia, la ayuda de Matlab no especifica con claridad cuáles son los parámetros que componen el vector *x* que ha de utilizarse para la multiplicación de matrices generadas con el controlador explícito. Se invirtió el orden en el que se interpretaron inicialmente las variables, colocando primero las acciones de control previas y después las variables de estado de posición y actitud, pero de nuevo esto no consiguió mejorar el resultado.

- Posible acción de control incremental

  El hecho de que el vector *x* incluyese las actuaciones aplicadas en el instante anterior planteó la posibilidad de que la acción de control calculada a partir de las matrices *F* y *G* junto con el vector *x* fuese un incremento con respecto a la acción de control anterior. En consecuencia se calculó la *u* y se le sumó la *uant* para aplicar el resultado de esta suma como acción de control, pero tampoco generó cambios significativos.

Como consecuencia, no se implementó el controlador predictivo explícito en el drone al no conseguir que este funcionase en simulación. Recordar de nuevo que este no hubiese sido el código final que tendría que compilarse para subirlo al drone, ya que habría surgido la necesidad de entregarle a la función la estructura sin utilizar la orden *load*.

7 CONCLUSIONES Y LÍNEAS FUTURAS

Nuestra recompensa se encuentra en el
esfuerzo y no en el resultado
Un esfuerzo total es una victoria completa.

Mahatma Gandhi

7.1 Conclusiones

El objetivo del proyecto era continuar la puesta en marcha desarrollada por un alumno de la escuela, cuyo trabajo se puede encontrar en [2], de la librería diseñada por el *MIT* junto con la empresa *Parrot* para generar un controlador de mayor complejidad a los que se desarrollan en su proyecto. Se optó por el control predictivo basado en modelo, desarrollando en Matlab un código capaz de implementar este controlador dentro del diagrama de bloques de Simulink de la librería mencionada, queriendo plantear el problema desde cero y sin utilizar el toolbox que Matlab proporciona para generar estos controladores.

Se han encontrado numerosas limitaciones al generar el código para la estrategia del controlador, ya que la acción de control se calcula para cada instante de muestreo, siguiendo unos criterios óptimos y acordes a unas restricciones impuestas en el modelo. Al tratarse de una estrategia *on-line,* el tiempo de computación se ve aumentado considerablemente, y más tratándose de un problema con un vector de estados de 12 componentes en el que se manejan matrices de alta dimensión. Este tiempo de cómputo, depende de la construcción en cada instante de muestreo de matrices que no se mantienen constantes para todo el problema, pero sobretodo, se ve afectado por la resolución del problema de programación cuadrática. Esto no ocurre con los controladores desarrollados en [2] ya que, por ejemplo, para el caso del LQR o asignación de polos, bastaba con ejecutar un archivo de Matlab, que venía incluido en esta libreria para generar la ganancia del controlador de forma previa a su implementación; por lo que en tiempo de ejecución el control se limita a calcular el producto del error que estamos cometiendo por esta ganancia.

Próximos a la finalización del proyecto, cuando se quiso exportar el código generado del controlador para poder subirlo al drone y realizar pruebas reales con el objetivo de compararlas con los resultados simulados en el entorno de Simulink, surgió el gran inconveniente que consistía en estar utilizando una función de Matlab, llamada *quadprog,* para la resolución del problema de optimización que calculaba la acción de control para cada instante. Como parece lógico, no se puede generar un código que esté utilizando una función externa, y tampoco era viable copiar la definición de dicha función dentro del código para que fuese posible utilizarla, ya que esta incluía más funciones con las que se presentaba el mismo problema. Por lo que no ha sido posible implementar este controlador en el drone.

La simulación del controlador explícito fue exitosa, pero la función *quadprog* no aceptaba un intervalo de restricciones estrecho para el vector de estados, ya que daba error en la simulación y dejaba de funcionar, por lo que se tuvo que ampliar este intervalo. Como ventaja en comparación con los otros controladores mencionados, se destaca la posibilidad de tener en cuenta en el modelo la acción aleatoria de las perturbaciones; manteniendo el sistema próximo a su referencia ante influencias que no se pueden modelar. Se puede concluir que; si el objetivo es controlar el minidrone utilizando las herramientas proporcionadas por el *MIT* para Matlab, la respuesta de controladores como PID, LQR y asignación de polos, es suficientemente buena y además, es posible que el compilador de Matlab genere un código en C que servirá para su implementación en el quadrotor.

7.2 Futuras líneas de trabajo

Si el objetivo es acabar implementando un controlador predictivo en el drone se pueden plantear distintas alternativas:

- Construir un *solver* propio que resuelva el problema de programación cuadrática; de tal manera se podría subir el código al drone. Al escribir el código directamente en C, el paso de generar el código a través del compilador de Matlab no sería necesario; ya que este se encargaba de compilar un código escrito en un archivo '.m' a C, introduciéndose ineficiencias en esta traducción entre lenguajes de programación. El problema que se podría encontrar es que la disminución del tiempo de computación haya sido la suficiente como para que el drone pueda ejecutarlo para cada instante de muestreo; ya que este periodo se mueve entorno a los 5ms.
- Generar de forma exitosa un controlador predictivo basado en modelo acorde con la estrategia explícita. En el punto anterior de esta memoria se exponen las características y ventajas de este tipo de controlador frente a la forma implícita del mismo. Si se consigue un MPC explícito para nuestro sistema que funcione bien en simulación, sería mas sencillo de implementar en el drone. Lo único que habría que resolver es cómo se le entrega al código el trabajo *off-line* que contiene las características de cada región. En este proyecto, cuando se vio que no era posible implementar el controlador implícito sin un solver, se trató de llevar a cabo este punto, pero el controlador no tuvo la respuesta esperada.

Aplicando modificaciones considerando las características del control predictivo (principalmente las restricciones), se pueden llegar a generar aplicaciones más complejas, muchas de las cuales expuestas en [10] como evitar obstáculos que aparezcan a lo largo de la trayectoria, recuperación de datos cuando no se obtiene información de sensores, minimizar la energía empleada y maximizar la calidad de la información de los sensores.

REFERENCIAS

[1] Parrot-Developers. *RollingSpiderEdu*. [En línea]. Disponible en: https://github.com/Parrot-Developers/RollingSpiderEdu. [Último acceso: 14 Abril 2018].

[2] Marín, Emilio. *TFG-ParrotRollingSpider*. [En línea]. Disponible en: https://github.com/emiliomarin/TFGParrotRollingSpider. [Último acceso: 12 Febrero 2018].

[3] Corke, P. 2017. *Robotics, Vision & Control*. Springer.

[4] Ollero Bartuone, A. 2001. *Robótica. Manipuladores y robots móviles.* Marcombo-Boixareu.

[5] Sanchiz, E. *Drones*.

[6] *Tipos de drones - Clasificación de drones - Categorías de drones*. [En línea]. Disponible en: http://noticias.compudemano.com/discusion/tipos-de-drones-clasificacion-de-drones-categorias-de-drones.145944/. [Último acceso: 8 Agosto 2018]

[7] WSJ. *One-Minute Review: Parrot's Rolling Spider MiniDrone*. [En línea]. Disponible en: https://www.wsj.com/video/one-minute-review-parrots-rolling-spider-minidrone/89871542-AE13-4536-886A-03D71E80D4C5.html. [Último acceso: 5 Mayo 2018]

[8] Stakelums Home & Hardware. *Parrot minidrone rolling spider white 96-PF723000*. [En línea]. Disponible en: https://www.stakelums.ie/product/parrot-minidrone-rolling-spider-white-96-pf723000/. [Último acceso: 5 Mayo 2018]

[9] Drones de carreras. [En línea]. Disponible en: http://dronesdecarreras.com/wp-content/uploads/2015/02/axis.jpg. [Último acceso: 17 Mayo 2018]

[10] López Torres, P., Muñoz Cueva, A., Arce, A. y Galán, R. *Implementación de algoritmos predictivos de control de vuelo en UAVs*.

[11] Sabatino, F. 2015. *Quadrotor control: modeling, nonlinear control design, and simulation*.

[12] Elsevier. 2014. *Position and attitude tracking control for a quadrotor UAV*. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S0019057814000081. [Último acceso: 21 Agosto 2018]

[13] Rodríguez Ramírez, D. y Bordóns Alba, C. 2007. *Apuntes de ingeniería de control*.

[14] Bemporad, A. 2014. *Explicit Model-Predictive Control*.

[15] Multi Parametric Toolbox. *Allow to deal with holes in explicit solutions.* [En línea]. Disponible en: http://people.ee.ethz.ch/~mpt/2/downloads/25/. [Último acceso: 20 Octubre 2018]

[16] The MathWorks, Inc. *Explicit MPC*. [En línea]. Disponible en: https://es.mathworks.com/help/mpc/ug/explicit-mpc.html. [Último acceso: 3 Noviembre 2018].

[17] The MathWorks, Inc. *generateExplicitMPC*. [En línea]. Disponible en: https://es.mathworks.com/help/mpc/ref/generateexplicitmpc.html. [Último acceso: 3 Noviembre 2018].

[18] The MathWorks, Inc. *Design Workflow for Explicit MPC*. [En línea]. Disponible en: https://es.mathworks.com/help/mpc/ug/design-workflow.html. [Último acceso: 3 Noviembre 2018].

[19] *16.30 Feedback Control Systems*. [En línea]. Disponible en: http://fast.scripts.mit.edu/dronecontrol/. [Último acceso: 20 Julio 2018].

[20] F. Camacho, E. y Bordons, C. 2007. *Model Predictive Control*. 2nd. ed. Springer.

[21] Ocampo Martínez, C. y Maestre Torreblanca, J.M. *Low level implementation of MPC.*

[22] Rossiter, J.A. 2004. *Model-Based Predictive Control. A Practica Approach.* CRC PRESS.

ANEXO A

```
% Matrices del sistema
A = 1/2;
B = 1/2;
D = -1;

% Estado inicial x(0)
x= 6;

% Definir horizonte de prediccion y otras variables
N=5;
nx=size(A,2); % Numero de estados
nu=size(B,2); % Numero de entradas
nw=size(D,2); % Numero de perturbaciones


% Estados futuros se pueden escribir como X=Gx*x(0)+Gu*U+Gw*W
% Construir matriz Gx
Gx=zeros(N*nx,nx);
for i=1:N
    Gx((i-1)*nx+1:i*nx,:)=A^i;
end

% Construir matriz Gu
Gu=zeros(N*nx,N*nu);
for i=1:N
    for j=1:i
        Gu((i-1)*nx+1:i*nx,(j-1)*nu+1:j*nu)=A^(i-j)*B;
    end
end

% Construir matriz Gw
Gw=zeros(N*nx,N*nw);
for i=1:N
    for j=1:i
        Gw((i-1)*nx+1:i*nx,(j-1)*nw+1:j*nw)=A^(i-j)*D;
    end
end

% Calculo de las matrices de coste Q y R
Q=1;
R=1;

% Construir matriz Q_hat
Q_hat=kron(eye(N),Q);
% Construir matriz R_hat
R_hat = kron(eye(N),R);

% Perturbaciones esperadas
W=ones(N*nw,1);

% Construir funcion de costes
H=Gu'*Q_hat*Gu+R_hat;
F=x'*Gx'*Q_hat*Gu+W'*Gw'*Q_hat*Gu;
% Restricciones
Ax=[1;-1];
```

```
bx=[10;10];
Au=[1;-1];
bu=[5;5];

% Transformar en restricciones de U
Ax_hat=kron(eye(N),Ax);
bx_hat=kron(ones(N,1),bx);
Au_hat=kron(eye(N),Au);
bu_hat=kron(ones(N,1),bu);

% Agrupar restricciones de U
AU=[Ax_hat*Gu; Au_hat];
bU=[bx_hat-Ax_hat*Gx*x-Ax_hat*Gw*W;bu_hat];

% MPC en accion
Disturb= 0.1*normrnd(0.5,1,100+N,1); % Perturbaciones aleatorias
% Bucle de simulacion
for k=1:100
    % Perturbaciones esperadas
    W=Disturb(k:k+N-1)+0.1*normrnd(0,0.2,N,1);
    % Actualizar matrices para estado y perturbaciones actuales
    F=x'*Gx'*Q_hat*Gu+W'*Gw'*Q_hat*Gu;
    bU=[bx_hat-Ax_hat*Gx*x-Ax_hat*Gw*W; bu_hat];

    UMPC=quadprog(H,F,AU,bU);
    u=UMPC(1); % Aplicar solo la primera componente de la accion de control

    x=A*x+B*u+D*Disturb(k); % Actualizo el valor el estado anterior
end
```

ANEXO B

```
function u = fcn_MPC(x)
u=zeros(4,1);
coder.extrinsic('quadprog')
coder.extrinsic('c2d')

%% 1) Modelo del Drone

A=[0 0 0 0 0 0 1 0 0 0 0 0;
0 0 0 0 0 0 0 1 0 0 0 0;
0 0 0 0 0 0 0 0 1 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 1;
0 0 0 0 0 0 0 0 0 0 1 0;
0 0 0 0 0 0 0 0 0 1 0 0;
0 0 0 0 -9.81000000000000 0 -0.0644000000000000 0 0 0 0.139000000000000 0;
0 0 0 0 0 9.81000000000000 0 -0.0644000000000000 0 -0.139000000000000 0 0;
0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 -1.01360000000000 0 -2.18760000000000 0
0;
0 0 0 0 0 0 0.755800000000000 0 0 0 -1.63120000000000 0;
0 0 0 0 0 0 0 0 0 0 0 -0.124800000000000];

B=[0 0 0 0;0 0 0 0;
0 0 0 0;0 0 0 0;
0 0 0 0;0 0 0 0;
0 0 0 0;0 0 0 0;
-0.00960000000000000 0.00960000000000000 -0.00960000000000000
0.00960000000000000;
0.420200000000000 0.420200000000000 -0.420200000000000 -0.420200000000000;
0.313300000000000 -0.313300000000000 -0.313300000000000 0.313300000000000;
-0.0115000000000000 -0.0115000000000000 -0.0115000000000000 -
0.0115000000000000];


%% Modelo discreto
Ts=0.04;
[Ad,Bd]=c2d(A,B,Ts);


% SIN Perturbaciones
D=zeros(size(Ad,1),1);

% CON Perturbaciones
D = 0.01*ones(size(Ad,1),1);

% Definir horizonte de prediccion y otras variables
N=10;
nx=size(Ad,2); % Numero de estados
nu=size(Bd,2); % Numero de entradas
nw=size(D,2); % Numero de perturbaciones


%% 2) Estados futuros se pueden escribir como X=Gx*x(0)+Gu*U+Gw*W
% Construir matriz Gx
Gx=zeros(N*nx,nx);
for i=1:N
```

```
    Gx((i-1)*nx+1:i*nx,:)=Ad^i;
end

% Construir matriz Gu
Gu=zeros(N*nx,N*nu);
for i=1:N
    for j=1:i
        Gu((i-1)*nx+1:i*nx,(j-1)*nu+1:j*nu)=Ad^(i-j)*Bd;
    end
end

% Construir matriz Gw
Gw=zeros(N*nx,N*nw);
for i=1:N
    for j=1:i
        if nw==1
            Gw((i-1)*nx+1:i*nx,j)=Ad^(i-j)*D;
        else
            Gw((i-1)*nx+1:i*nx,(j-1)*nw+1:j*nw)=Ad^(i-j)*D;
        end
    end
end

%% 3) Funcion de costes
% Limite de los estados
pos_max     = 1.5;
att_max     = 0.3;
dpos_max    = 1;
datt_max    = 1;

motor_max = 500;

% Ponderacion de los estados
pos_x_wght        = 0.3/3;
pos_y_wght        = 0.3/3;
pos_z_wght        = 0.2/3*50;

orient_ypr_wghts  = 0.2/3*60;
dpos_wghts        = 0.55/3;
dorient_pqr_wghts = 2.5/3;

rho = 0.01;

% Normalizacion de la ponderacion
maxs    = [pos_max pos_max pos_max att_max att_max att_max dpos_max dpos_max
dpos_max datt_max datt_max datt_max];
weights = [pos_x_wght pos_y_wght pos_z_wght orient_ypr_wghts orient_ypr_wghts
orient_ypr_wghts dpos_wghts dpos_wghts dpos_wghts dorient_pqr_wghts
dorient_pqr_wghts dorient_pqr_wghts];

weights = weights/sqrt(sum(weights.^2)); % Normalizo: suma de los pesos2 = 1

% Calculo de las matrices de coste Q y R
Q           = diag((weights.^2)./(maxs.^2));
R           = rho*diag(1./([motor_max motor_max motor_max motor_max].^2));

% Construir matriz Q_hat
Q_hat=kron(eye(N),Q);
% Construir matriz R_hat
R_hat = kron(eye(N),R);
```

```
% SIN Perturbaciones
W=zeros(N*nw,1);

% CON Perturbaciones
Disturb= 0.1*abs(normrnd(0.25,0.5,100+N,1));
k=randi([1 100],1);
W=Disturb(k:k+N-1)+0.1*abs(normrnd(0,0.2,N,1));

% Construir funcion de costes
H=Gu'*Q_hat*Gu+R_hat;
F=x'*Gx'*Q_hat*Gu+W'*Gw'*Q_hat*Gu;

%% 4) Restricciones
Ax=[eye(12); -eye(12)];
bx=[15; 15; 1; 0.6; 0.6; 0.6; 1.5; 1.5; 1.5; 1.5; 1.5; 1.5;...
    15; 15; 2; 0.6; 0.6; 0.6; 1.5; 1.5; 1.5; 1.5; 1.5; 1.5];
Au=[eye(4); -1*eye(4)];
bu=[1000; 1000; 1000; 1000;...
    1000; 1000; 1000; 1000];

% Transformar en restricciones de U
Ax_hat=kron(eye(N),Ax);
bx_hat=kron(ones(N,1),bx);
Au_hat=kron(eye(N),Au);
bu_hat=kron(ones(N,1),bu);

% Agrupar restricciones de U
AU=[Ax_hat*Gu; Au_hat];
bU=[bx_hat-Ax_hat*Gx*x-Ax_hat*Gw*W;bu_hat];

%% 5) MPC en accion

UMPC=zeros(N*nu,1);
UMPC=quadprog(H,F,AU,bU);
u=UMPC(1:4,1);

end
```

ANEXO C

```
clear
clc
load solucion.mat
% REALIZO UNA COPIA DE LA ESTRUCTURA ANTERIOR PARA MODIFICARLA
sol=struct_sol;

%% H
% CALCULO LA DIMENSION MAXIMA DE LA MATRIZ H
max_H=size(struct_sol(1).H);
for i=1:length(struct_sol)
    if size(struct_sol(i).H,1)>max_H(1)
        max_H=size(struct_sol(i).H);
    end
end
% HAGO QUE TODAS LAS MATRICES TENGAN LA MAXIMA DIMENSION RELLENANDO CON 0
for i=1:length(struct_sol)
    dim_H=size(struct_sol(i).H,1);
    if(dim_H<max_H(1))
        sol(i).H=zeros(max_H);
        sol(i).H(1:dim_H,:)=struct_sol(i).H;
    end
end

%% K
% CALCULO LA DIMENSION MAXIMA DE LA MATRIZ K
max_K=size(struct_sol(1).K);
for i=1:length(struct_sol)
    if size(struct_sol(i).K,1)>max_K(1)
        max_K=size(struct_sol(i).K);
    end
end
% HAGO QUE TODAS LAS MATRICES TENGAN LA MAXIMA DIMENSION RELLENANDO CON 0
for i=1:length(struct_sol)
    dim_K=size(struct_sol(i).K,1);
    if(dim_K<max_K(1))
        sol(i).K=zeros(max_K);
        sol(i).K(1:dim_K,:)=struct_sol(i).K;
    end
end

%% GUARDO EN ESTRUCTURA QUE RESUELVE PROBLEMA DIMENSIONES
save dimension.mat sol
```