UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

Diseño de un sistema de generación y control de trayectorias para un cuatrirrotor Parrot AR.Drone 2.0 utilizando técnicas de control predictivo

AUTOR: Hector Zaragozá Pajarón

TUTOR: Javier Sanchís Saez

Seleccion NOMBRE DEL COTUTOR

Curso Académico: 2019-20

AGRADECIMIENTOS

"A mi familia

A mi tutor, Javier Sanchís

A el personal sanitario, incluido a mi madre, que hizo cosas más importantes mientras yo desarrollaba este TFM."

RESUMEN

En este trabajo de Fin de Máster se explican las características básicas del sistema a controlar, que es el dron Parrot AR. Seguidamente, se diseña y se implementa una arquitectura de control predictivo con restricciones para el control y seguimiento de trayectorias. Se simula el funcionamiento con Simulink y Flightgear con un generador de trayectorias que resulta en un piloto automático, y se expone su implementación en un dron real y se compara con las simulaciones

Palabras Clave: Control predictivo, cuadrático, Simulink, dron, Flightgear, piloto automático.

RESUM

A aquest treball de fi de Màster s'expliquen les característiques bàsiques del sistema a controlar, que es el dron parrot AR. Seguidament, es dissenya i s'implementa una arquitectura de control predictiu amb restriccions per al control i seguiment de trajectòries. Es simula el funcionament en Simulink y Flightgear amb un generador de trajectòries que resulta en un pilot automàtic, i s'exposa la seua implementació a un dron real i es compara amb les simulacions.

Paraules clau: Control predictiu, quadràtic, Simulink, dron, Flightgear, piloto automàtic.

ABSTRACT

In This document the basic characteristics of the controlled system, the drone Parrot AR are explained. Secondly, a model predictive control with restrictions and trajectory following is implemented. Next, the system is simulated in Flightgear and Simulink, along with a trajectory generator that results in an auto-pilot, and the real drone implementation is exposed and compared

Keywords: Predictive Control, quadratic, drone, Simulink, Flightgear, autopilot.

ÍNDICE

Memoria del documento

Donde se describe el desarrollo teórico y simulación del control de un dron y la implementación del piloto automático utilizando dicho control.

1. INTRODUCCIÓN

1.1. OBJETIVO DEL DOCUMENTO

El objetivo de este documento es diseñar, implementar y simular un uso de autopiloto basado en un sistema de control predictivo para un dron cuatrirrotor. Este sistema dispone de restricciones de tipo cuadrático para un control de segundo nivel. En el documento se describe el modelado del sistema, implementación de los algoritmos de control y de gestión de las referencias del sistema; además de la simulación gráfica utilizada.

Los objetivos específicos que se pretenden conseguir en este trabajo son:

- Estudiar y analizar el sistema para la comprensión clara y sencilla de la estructura de control que se diseñe para el control de este tipo de drones, cuáles son los parámetros más importantes en vuelo, y como se consigue un cambio en estos.
- Diseñar una capa de control de segundo nivel, su objetivo y funcionamiento. Crear los modelos de control predictivo utilizando "Matlab Sistem Identification Toolbox"; que serán necesarios para que el sistema de control funcione de forma adecuada, y justificar la elección de los modelos.
- Implementar diferentes algoritmos prototipo de control predictivo en el programa matemático y de control Matlab; hasta conseguir un algoritmo de control final que implementaremos en bucle de simulación con el dron simulado.
- Implementar los parámetros del controlador cuadrático para conseguir una respuesta que se adapte a nuestro objetivo de llegada con una robustez razonable, y diseñaremos un generador de trayectorias que se dedique a la creación automática de referencias al controlador de segundo nivel.
- Simular el vuelo del dron con el sistema de control diseñado en un entorno gráfico capaz de simular el funcionamiento de esta simulación de forma realista.

En definitiva, el objetivo es la simulación del vuelo del dron con el sistema en un entorno gráfico según los parámetros ajustables de un control predictivo cuadrático, y la creación de trayectorias para guiar a este autopiloto, además de simularlo gráficamente con software libre.

1.2. ESTRUCTURA DEL DOCUMENTO

En primer lugar, se describe el marco teórico básico del movimiento de un dron cuatrirrotor de en cada una de las direcciones del espacio y como utiliza los cuatro rotores para conseguirlo, además de los principios básicos del control predictivo con modelo en espacio de estados y restricciones cuadráticas. Seguidamente, se describe la identificación de modelos para las señales del dron en simulación que nos ayuden a predecir el estado de este en el futuro. Una vez conseguido el modelo del dron, se procede a la programación de los algoritmos de control para el seguimiento de trayectorias y a la sintonía de los controladores. Por último, se simulará su funcionamiento en el programa de software libre Flightgear, adaptando la simulación para transmitir la información necesaria.

2. MARCO TEÓRICO DEL SISTEMA DRON Y EL CONTROLADOR

2.1. GENERALIDADES

En este trabajo consideraremos el control con un piloto automático basado en control predictivo de un dron cuatrirrotor simulado; capaz de volar en todas las direcciones del espacio y estabilizarse en cualquiera de ellas gracias al uso independiente de sus cuatro rotores.

El dron es un dron de marca Parrot, un fabricante de diferentes clases de drones controlados a distancia; del modelo de cuatrirrotor AR 2.0.

Ilustración 1. Dron Parrot AR en reposos con carcasa exterior acoplada, Fuente: https://support.parrot.com/global/support/products/parrot-ar-drone-2 (2020).

Para conseguir todas las maniobras de vuelo, el dron Parrot AR (como el que aparece en la ilustración 1) utiliza los rotores típicos de todos los cuatrirrotores; estas maniobras se explican en detalle en el segundo punto, y permiten conseguir con cuatro motores verticales el desplazamiento en todas las direcciones y la rotación en todas las direcciones angulares.

2.2. ESTRUCTURA DE CONTROL DEL DRON

La estructura de control del dron utilizado en este trabajo estará basada en dos niveles de control distintos; cómo se puede ver en la figura 2:

- El control empotrado del dron, que ya es capaz de controlar los movimientos básicos de dron como son la velocidad vertical y el ajuste de los ángulos de roll, pitch y yaw.
- En segundo lugar, tendremos el control de segundo nivel a diseñar en este TFM. Este control recibirá una estimación de la posición del dron en X, altura y heading basada en la integración de las velocidades indicadas por los sensores del dron. Como el control de primer nivel desacopla las entradas de control para que tan solo dependan de una salida, el sistema es completamente diagonal. Su composición se puede observar en la figura 1.

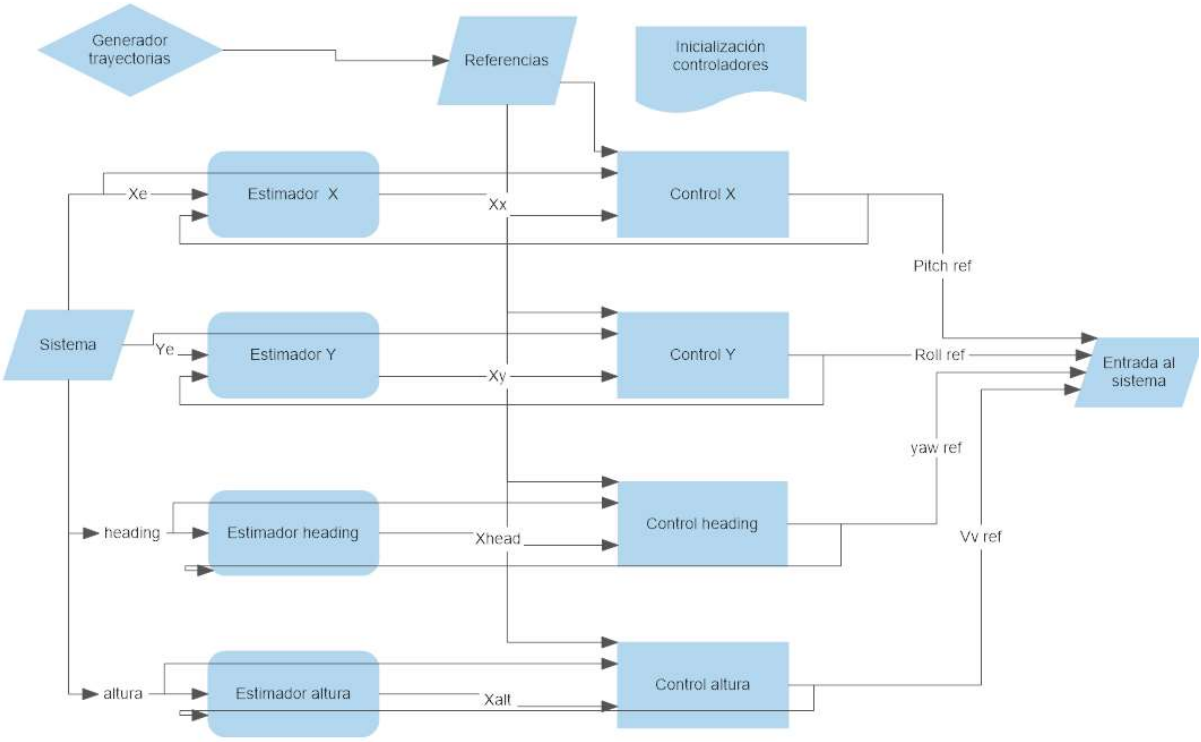


Figura 1. Control de segundo nivel en detalle, con sus correspondientes estimadores del estado. Fuente: Elaboración Propia

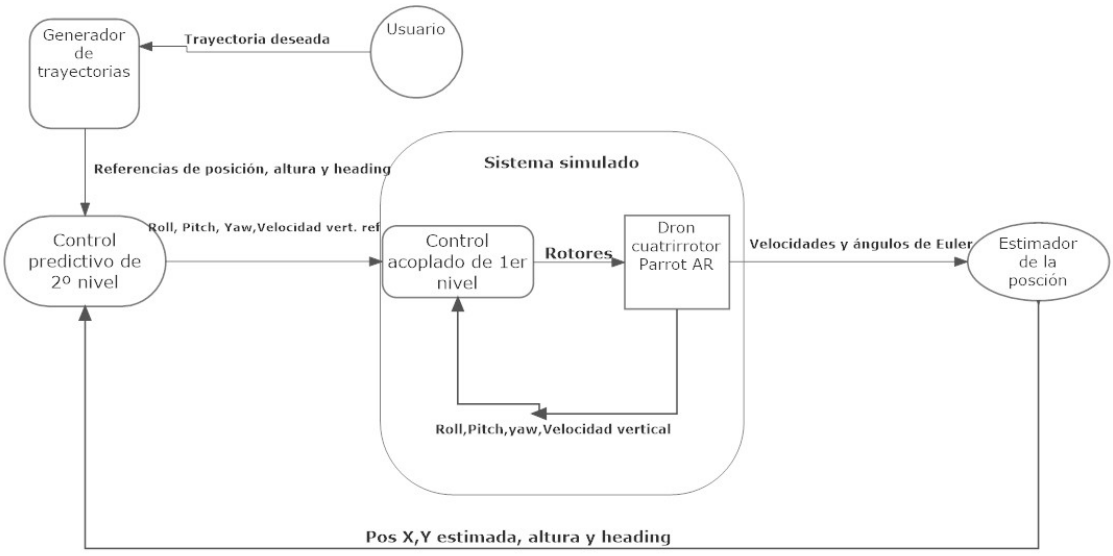


Figura 2. Esquema de control conceptual del control del dron. Fuente: Elaboración Propia

Una vez se ha logrado identificar y controlar el sistema diagonal de forma exitosa; se ha elaborado un generador de trayectorias que alimenta las referencias que damos al controlador de segundo nivel; con indicar tan solo una trayectoria y un tiempo de llegada se interpola esa información para conseguir diferentes trayectorias y movimientos de forma sencilla. Su funcionamiento lógico es el que se puede ver en el esquema de la figura 3.

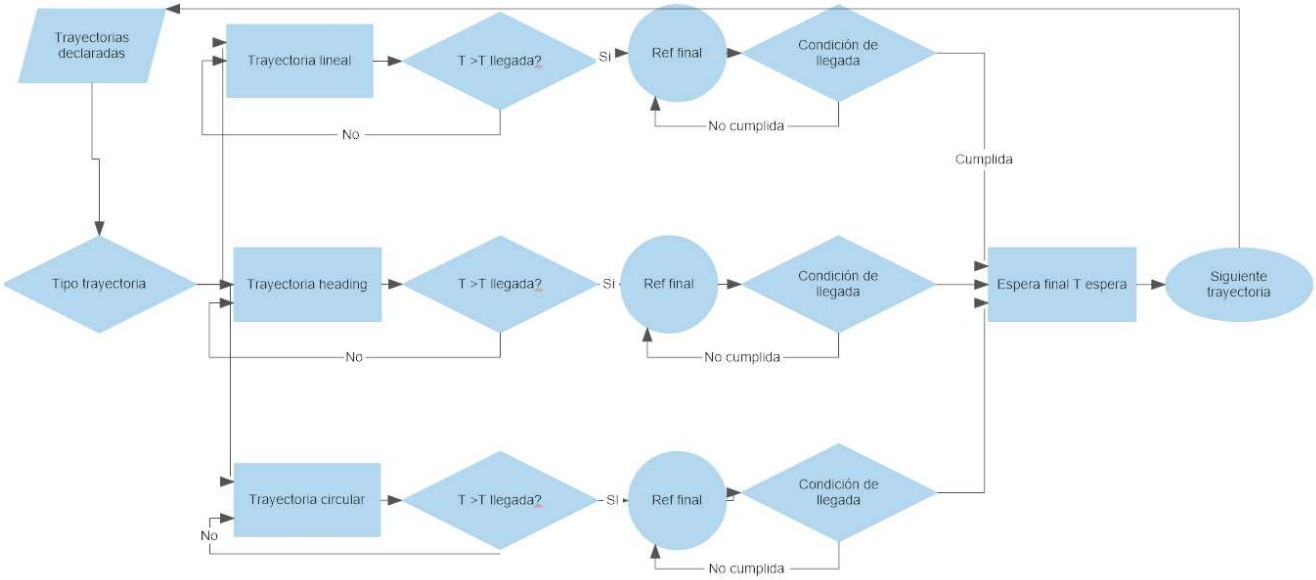


Figura 3. Generador de trayectorias detallando su funcionamiento lógico. Fuente: Elaboración Propia.

2.3. MANIOBRAS TEÓRICAS DEL DRON

Como desarrollan F. A. González, M. E. Afanador Cristancho y E. F. Niño López en su artículo de investigación (1), este tipo de vehículos aéreos no tripulados son perfectos para labores de vigilancia o inspección de áreas abiertas, y por ello un sistema de autopiloto es relevante para el uso de este tipo de drones, ya que su uso no recreativo es el seguimiento de rutas concretas. El modelado físico de los movimientos del dron se puede realizar con una estructura en forma de X o en forma de cruz. Como ejemplo físico, utilizaremos los diagramas del Artículo de F. A. González y otros (1), que ejemplifican el modelo en cruz:

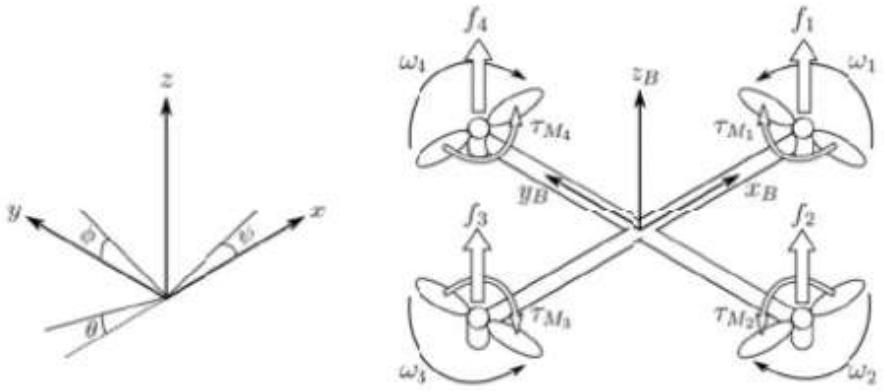


Ilustración 2. Modelo de cuatrirrotor en cruz con fuerzas, sistema de coordenadas y vectores de rotación indicados. Fuente: : F. A. González y otros(1).

En este artículo, se considera el sentido de giro roll a la variación del ángulo Θ;el sentido de giro pitch se llama la variación del ángulo Φ, y el sentido de giro yaw se considera a la variación del ángulo ψ. Aunque en el artículo en si no se utiliza esta terminología, en este trabajo se utilizará después a la hora de diseñar el control.

Podemos determinar qué movimientos son necesarios para los desplazamientos en cada una de las direcciones del dron mirando las ecuaciones del modelo que tenemos. En la ecuación 1, podemos observar como el modelo se puede simular introduciendo una fuerza externa durante el modelado que dependa de la velocidad de giro al cuadrado por una constante de empuje, siendo la fuerza vertical total la creada por cada uno de los motores.

$$f = k_t * w_i^2$$

Ecuación 1. Expresión de la fuerza vertical de empuje provocada por un motor cualquiera *i*. *kt* es la constante de empuje y es un parámetro aerodinámico del dron. Fuente: F. A. González y otros(1).

$$f = \Sigma f_i = K_t * \sum \omega_i^2$$

$$F = \begin{bmatrix} 0 \\ 0 \\ f \end{bmatrix}$$

Ecuación 2. Expresión de la fuerza vertical de empuje para varios motores. Fuente: F. A. González y otros(1).

Para crear giros en el eje z o cambio de yaw, es necesario crear diferentes tipos de momentos de arrastre generados por el rozamiento de los rotores con el aire. En la ecuación 3, podemos observar la expresión de cómo se genera el momento de arrastre. Sin embargo; el término de la inercia I que multiplica a la aceleración angular suele despreciarse, por valor normalmente pequeño, y solo queda dependiente de Kd, una constante mayor que 0; y la velocidad angular.

$$\tau_{Mi} = Kd * w^2 + I_m * \dot{w}_i$$

Ecuación 3. Expresión del momento de arrastre aerodinámico, que depende de una constante por la velocidad angular y la inercia por la aceleración angular. Fuente: F. A. González y otros(1).

Para el giro del dron en otras direcciones, es necesario crear una inclinación, lo cual se consigue con la diferencia de velocidades de giro entre las diferentes hélices. Observando la ilustración 2; diferencias la fuerzas entre 4 y 2 provoca giros alrededor del eje x; o cambio de roll, mientras que giros alrededor del eje y, o cambio de pitch, se crean con desequilibrios entre 1 y 3. Sin embargo, el heading o giro en z es generado de forma diferente, y depende de una constante y la diferencia entre la velocidad de las hélices positivas y negativas (en sentido).

$$\tau_\psi = \sum \tau_{Mi} = Kd * (-w_1^2 + w_2^2 - w_3^2 + w_4^2)$$

$$\tau = \begin{pmatrix} \tau_\Phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix} = \begin{pmatrix} l * (f_4 - f_2) \\ l * (f_3 - f_1) \\ \sum \tau_{Mi} \end{pmatrix}$$

Ecuación 3. Formulación de los momentos en las diferentes direcciones de rotación del dron cuatrirrotor, y su relación con las velocidades angulares del sistema. Fuente: F. A. González y otros(1).

De esa forma, podemos crear momentos y fuerzas en direcciones que necesitamos utilizando diferencias de velocidad entre los diferentes rotores, que se traducen en diferencias de fuerza y en momentos según la ecuación 3. Y si queremos realizar desplazamiento en x o y, primero giramos el dron en una dirección específica, y utilizamos la inclinación para impulsar el dron en una dirección.

2.4. DESCRIPCIÓN DEL MODELO UTILIZADO

Para simular el vuelo y condiciones de un Dron real en Simulink, se ha utilizado el "AR Drone Simulink Development-Kit V1.1 (5)"; donde se tiene un modelo identificado de un dron Parrot AR, que dispone de la capacidad de simular el controlador durante la fase de diseño y probarlo en un control por Wi-Fi durante la fase de aplicación. El modelo consiste en diferentes scripts, modelos y funciones de Matlab y Simulink que permiten la simulación y control de un dron tanto real como simulado, y serán la base para la simulación de control de este trabajo.

En simulación encontramos tres modelos para simular: "ARDroneBaseSim", "ARDroneHoverSim" y "ARDRoneWpTrackingSim". En cada uno se ejecuta una maniobra general, una maniobra que empieza en una posición estática en altura, o utiliza una función para crear puntos de paso para el sistema del dron. Como no se utilizará en este caso el control descrito en el modelo, tan solo se describirá el modelo del dron y su observador virtual.

En la ilustración 2 tenemos el modelo "ARDroneBaseSim", este modelo básicamente simula el dron con el sistema de control empotrado de primer nivel del que ya dispone. El bloque gris es un

controlador proporcional estándar que utiliza el modelo de Matlab para conseguir un control de segundo nivel. Este controlador será sustituido por nuestro sistema de control predictivo. Adicionalmente, se simula el funcionamiento del dron de tal forma que las salidas de los sensores acoplados en el dron no nos dan directamente la posición de este, sino que proporciona la velocidad en cada dirección y un sistema observador transforma estas magnitudes en posiciones integrando. En el caso de los ángulos, los ángulos son traducidos de ángulos de Euler a notación roll, pitch, yaw.

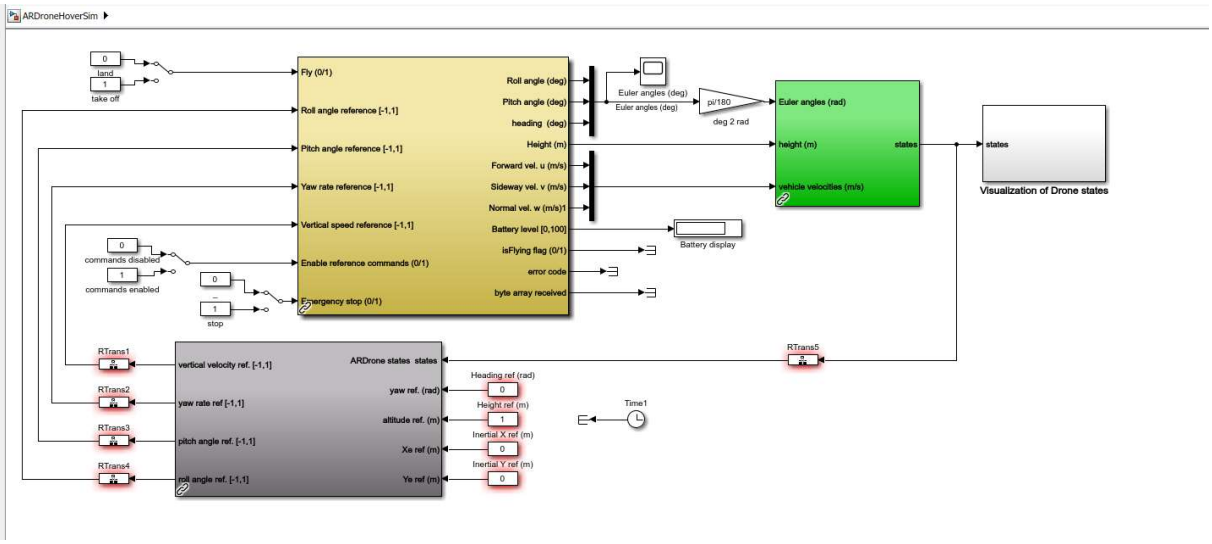


Ilustración 3. Modelo ARDroneBaseSim, donde el bloque Amarillo es el modelo del dron, el verde un observador de las salidas del dron a los inputs que necesita el controlador, y el bloque gris es el control que sustituiremos más adelante. Fuente: AR Drone Simulink Development-Kit V1.1 (5).

El sistema de simulación del dron es relativamente simple de exponer: Primero satura las entradas para asegurar que no hay ninguna entrada excesiva en los modelos lineales; y después se transmiten las señales a modelos en espacio de estados conseguidos gracias a la identificación experimental. Otras señales de salida; utilizadas para indicar errores o en la comunicación, están unidos a una constante ya que en simulación no son necesarias. Como simula también el control de primer nivel, las entradas son el roll, pitch yaw y velocidad vertical deseadas normalizadas entre -1 y 1.

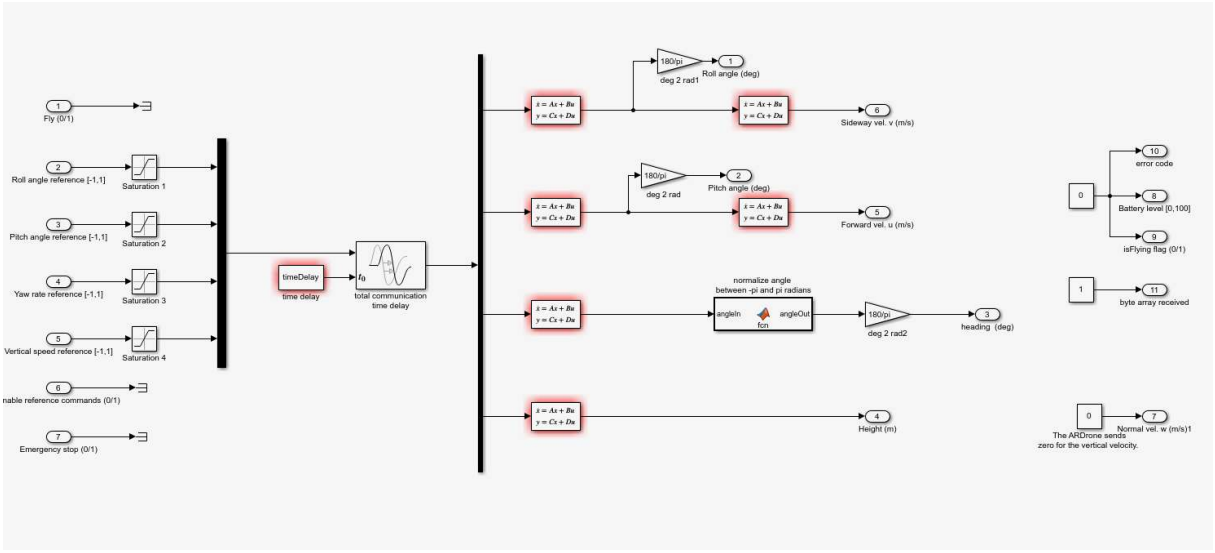


Ilustración 4. Interior del bloque del modelo del dron. Fuente: Captura propia.

2.5. MARCO TEÓRICO DEL CONTROL PREDICTIVO

En esta aplicación de control, se utilizará un sistema de control predictivo en espacio de estados, es decir, se utilizará un modelo diferencial para caracterizar al modelo del dron y se utilizará este modelo en la predicción de los estados del sistema en estados posteriores al actual, que influyen en el control del sistema. Para conseguir una predicción del sistema, el control predictivo utiliza los siguientes elementos:

- Un **modelo** identificado del sistema, basado en la excitación de las entradas del sistema en el rango de interés.
- Una **predicción** de los instantes siguientes basada en el estado futuro del sistema y las acciones de control anteriores.
- El **error** entre la predicción en el instante actual del sistema y la salida real , o error general de predicción.
- Un **índice de costes** que permita la determinación de las acciones de control en los instantes de control siguientes.
- Un cálculo de un número determinado de **acciones de control** donde solo se utiliza la primera y el resto se descartan para recalcular en instantes posteriores.

Para explicar el funcionamiento del control predictivo, partiremos de la fórmula final del cálculo:

$$y_{future} = y_{free\ px1} + G_{p\ x\ c} \cdot \Delta u_{c\ x\ 1}$$

Ecuación 4. Expresión de la predicción de la salida futura los "p" instantes posteriores al cálculo, conociendo los c siguientes cambios de u realizados. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

Lo cual nos indica que la predicción de la respuesta futura del sistema se basa en el uso de la respuesta libre; no forzada por las entradas, más el efecto de los c siguientes valores de la variación de la entrada. Si comenzamos el desarrollo de las ecuaciones con el sistema en espacio de estados sin acoplamiento directo:

$$x(k+1) = Ax(k) + Bu(k)$$
$$y(k) = C \cdot x(k)$$

Ecuación 5. Modelo estándar en espacio de estados discreto, donde el estado x y las entradas u definen el estado en la siguiente iteración, relacionadas con la salida mediante la matriz C. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

Observamos que para predecir el estado en k+1, podemos utilizar las entradas en los instantes futuros para calcular x en instantes, k+2, a esta predicción la denotamos **x(k+1|k)**, o el estado predicho en k+1 utilizando la información de k; y desarrollando para "p" predicciones futuras, tenemos:

$$x(k+1|k) = Ax(k) + Bu(k)$$
$$x(k+2|k) = Ax(k+1) + Bu(k+1)$$

$$\begin{bmatrix} x(k+1|k) \\ x(k+2|k) \\ \vdots \\ x(k+p|k) \end{bmatrix} = \begin{bmatrix} Ax(k) + Bu(k|k) \\ A^2x(k) + ABu(k|k) + Bu(k+1|k) \\ \vdots \\ A^px(k) + A^{p-1}Bu(k|k) + \cdots + ABu(k+p-2|k) + Bu(k+p-1|k) \end{bmatrix}$$

Ecuación 6.Modelo matricial de predicción donde; extrapolando la ecuación 5; se calculan los p instantes siguientes del estado a partir del primero. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

Lo cual, reestructurando y reagrupando, nos lleva a la siguiente formulación de x futura:

$$x_{future} = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^p \end{bmatrix} \cdot x(k) + \begin{bmatrix} B & 0 & \dots & 0 \\ AB & B & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{p-1}B & A^{p-2}B & \dots & B \end{bmatrix} \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+p-1k) \end{bmatrix}$$

Ecuación 6.Modelo matricial de predicción del estado agrupado. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

Sin embargo, recordemos que aquí solo se predice el estado interno del sistema, que en el modelo en espacio de estados es la variable o variables derivada en las ecuaciones diferenciales que definen el modelo. Para obtener la salida de este sistema; se multiplica cada matriz por la matriz C.

$$\begin{bmatrix} y(k+1|k) \\ y(k+2|k) \\ \vdots \\ y(k+p|k) \end{bmatrix} = \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^p \end{bmatrix} \cdot x(k) + \begin{bmatrix} CB & 0 & \dots & 0 \\ CAB & CB & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{p-1}B & CA^{p-2}B & \dots & CB \end{bmatrix} \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+p-1k) \end{bmatrix}$$

Ecuación 6.Modelo matricial de predicción de la salida. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

Sim embargo, una vez tenemos este conjunto de matrices, es beneficioso para el cálculo de la acción de control por índices de optimización, expresar estas matrices en incrementos de u o Δu; y para eso se hace un procedimiento algo engorroso pero que, omitiendo los pasos intermedios resulta en:

$$\begin{bmatrix} y(k+1|k) \\ y(k+2|k) \\ \vdots \\ y(k+p|k) \end{bmatrix} = \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^p \end{bmatrix} \cdot x(k) + \begin{bmatrix} CB \\ CAB + CB \\ \vdots \\ \sum_{i=0}^{p-1} CA^iB \end{bmatrix} \cdot u(k-1) + \begin{bmatrix} CB & 0 & \dots & 0 \\ CAB + CB & CB & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{p-1} CA^iB & \sum_{i=0}^{p-2} CA^iB & \dots & CB \end{bmatrix} \cdot \begin{bmatrix} \Delta u(k|k) \\ \Delta u(k+1|k) \\ \vdots \\ \Delta u(k+p-1|k) \end{bmatrix}$$

Ecuación 7. Transformación del modelo de predicción a incrementos de la entrada u. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

Y si llamamos P, Q y G a las matrices anteriores que multiplican a x(k), u (k-1) y Δu respectivamente, y se define la respuesta libre como:

$$y_{free} = P * x(k) + Q * u(k-1)$$

Ecuación 8. Ecuación de la respuesta libre, sin considerar variaciones de la entrada; antes de definir el error. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

Tenemos la ecuación 4 casi en su totalidad, pero debemos incluir el bias(k) o error de predicción para tener una estimación tolerante a errores de modelado. También ayuda a corregir errores del estimador de los estados del sistema que se requiere para tener información de los estados reales del sistema. Se considera constante en todos los instantes de predicción y se obtiene con las siguientes ecuaciones:

$$bias(k) = y_{measured}(k) - C * x(k)$$

$$y_{free} = P * x(k) + Q * u(k-1) + 1 * bias(k)$$

Ecuación 9. Ecuación de la respuesta libre, sin considerar variaciones de la entrada; incluyendo el "bias" o error constante en la predicción de "yfree" Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

Y consideramos la matriz que multiplica a Δu G; llegamos a la ecuación 4:

$$y_{future} = y_{free\,px} + \begin{bmatrix} CB & 0 & \dots & 0 \\ CAB + CB & CB & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{p-1} CA^iB & \sum_{i=0}^{p-2} CA^iB & \dots & CB \end{bmatrix} \cdot \Delta u_{c\,x\,1}$$

Ecuación 10. Ecuación 4 descrita desarrollando la parte que multiplica a Δu. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

2.6. MARCO TEÓRICO DE LAS RESTRICIONES EN CONTROL PREDICTIVO

Para formular el índice de costes cuadrático que optimiza los siguientes c movimientos de control para el sistema, es necesario formular el índice de costes, las matrices que lo acompañan, y la matriz de restricciones necesaria para acotar el problema como se desee. Para este problema, se plantea un índice de coste cuadrático con restricciones no estrictas o, en otras palabras, que tiene un margen de error a no cumplirse cuando el problema se optimiza. Este error se contabiliza en la variable ε y se minimiza en la función objetivo. Antes de considerar el error; la expresión de problema de optimización tiene la expresión:

$$min_{\Delta u} \frac{1}{2} \Delta u^T H \Delta u + c^T \Delta u$$
$$s.t.\ A\Delta u \leq b$$

Ecuación 11. Índice cuadrático de la resolución del control óptimo, definido por las matrices "H" y c, y sujeto a las restricciones definidas en "A" y "b". Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

Para obtener las matrices "H" y c, se define "H" como la matriz Hessiana que formula el índice de costes del problema de optimización lineal. "alfa" y "lambda" son el peso dado en la optimización al error de la salida y el valor de los movimientos de u respectivamente. El valor de ρ es el peso del error en el cumplimiento de ciertas restricciones, en este caso el valor de la salida. La definición de c es el vector gradiente del problema y se define como muestra la ecuación 12; donde "e" es el error de posición de la salida.

$$J = \frac{1}{2}\begin{bmatrix}\Delta u \\ \varepsilon\end{bmatrix}^T \begin{bmatrix}(G^T \alpha G + \lambda) & 0 \\ 0 & \rho\end{bmatrix}\begin{bmatrix}\Delta u \\ \varepsilon\end{bmatrix} + \begin{bmatrix}(-G^T \alpha e) \\ 0\end{bmatrix}^T \begin{bmatrix}\Delta u \\ \varepsilon\end{bmatrix} = \frac{1}{2} x^T H x + c^T x$$

Ecuación 12. Índice cuadrático de la resolución del control óptimo considerando restricciones no estrictas o blandas. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

Las restricciones que definen las matrices "A" y el vector "b", contabilizando el error, tienen la arquitectura de la ecuación 13, donde además de contabilizar en las restricciones la variación de la entrada u; se considera una variable ε, que se debe minimizar, pero puede ser positiva mayor que 0, permitiendo que las restricciones a la salida no se cumplan estrictamente. Se presentarán las restricciones y se definirá su función atendiendo al uso de cada fila de "A" o la matriz izquierda que multiplica a Δu y ε:

$$\begin{bmatrix} I & 0 \\ -I & 0 \\ I_L & 0 \\ -I_L & 0 \\ G & -1 \\ -G & -1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \Delta u \\ \varepsilon \end{bmatrix} \leq \begin{bmatrix} \Delta u_{max_1} \\ \Delta u_{max_2} \\ \vdots \\ \Delta u_{\max_c} \\ -\Delta u_{min_1} \\ -\Delta u_{min_2} \\ \vdots \\ -\Delta u_{min_c} \\ u_{\max_1} - u(k-1) \\ u_{\max_2} - u(k-1) \\ \vdots \\ u_{\max_c} - u(k-1) \\ -u_{\min_1} + u(k-1) \\ -u_{\min_2} + u(k-1) \\ \vdots \\ -u_{\min_c} + u(k-1) \\ y_{max} - yfree \\ -y_{min} + yfree \\ 0 \end{bmatrix}$$

Ecuación 13. Restricciones definidas en el problema de optimización. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

- La primera fila de restricciones se centra en limitar la variación de u o Δu; para ello tan solo es necesario multiplicar por una matriz identidad la primera variable; que es Δu. La matrices identidad son de tamaño número de entradas X horizonte de control (c). La restricción está repetida en la segunda fila para límites inferiores
- Para entender por qué la tercera fila se multiplica por una matriz triangular inferior, hemos de entender que Δu, como incógnita, es un vector de c variaciones de control, y según las siguientes restricciones esos c movimientos, en incremento total Δu sumado, no deben superar el incremento necesario para superar la u máxima; es decir la u máxima menos la u anterior. La siguiente fila es la misma condición para la u mínima, con las variables negadas.

$$\Delta u(k+c-1) + \cdots + \Delta u(k) \leq u_{max_c} - \mathrm{u(k\ -1)}$$

Ecuación 14. Desarrollo de las condiciones de "u" máxima. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

- La restricción de la fila 5 es que G por Δu, que definen la respuesta forzada, no deben ser mayores que la diferencia entre la respuesta libre y la salida máxima del sistema, para no superar el límite máximo impuesto a la salida. Sin embargo; esta restricción suma el error en negativo, y por tanto puede romperse haciendo ε mayor que 0 si cualquier otra opción es peor como alternativa, según el peso dado al error. La sexta fila es la misma restricción en negativo para los límites inferiores
- La última restricción es equivalente a exigir una ε siempre mayor o igual que 0.

Una vez se han definido las restricciones del índice a minimizar, se puede conseguir la solución óptima de u según los parámetros introducidos, y controlar el sistema. Finalmente, y aunque en este texto no se utilizará para la determinación de los movimientos de la acción de control, Δu se puede calcular utilizando el enfoque de mínimos cuadrados para el índice de coste cuadrático, que resulta en el siguiente índice de coste y la siguiente formulación de "H":

$$J = [y_{future} - sp]^T \, \alpha[y_{future} - sp] + \, \Delta u^T \lambda \Delta u$$

$$\Delta u = (G^T \alpha G + \lambda)^{-1} G^T \alpha \cdot \left(sp - y_{free}\right) = H \cdot (\left(sp - y_{free}\right)$$

Ecuación 15. Optimización del índice cuadrático y resolución por mínimos cuadrados. Fuente: Elaboración propia a partir del texto de J. Sanchís (2)

3. IMPLEMENTACIÓN DEL CONTROL DE SEGUNDO NIVEL.

3.1. IDENTIFICACIÓN DEL SISTEMA

Antes de crear ningún control predictivo que podamos aplicar a nuestro sistema; es necesario conseguir un modelo de como afectarán las entradas al dron con el controlador primario empotrado a las salidas, ya convertidas en unidades de posición absolutas y giro en sistema de referencia roll, pitch, yaw. Para conseguir las funciones de transferencia, se pueden utilizar diferentes tipos de entrada: Pseudoaleatoria, Escalones, impulsos…En este caso nos hemos decidido por impulsos en las entradas, ya que al ser un sistema con integradores a la salida en la función de transferencia; la respuesta a un escalón es una rampa al infinito, y un impulso permite estabilizarse a la señal de forma correcta. El sistema diseñado utiliza la Matlab Identification Toolbox, que tiene herramientas de identificación diseñadas para esta clase de necesidades.

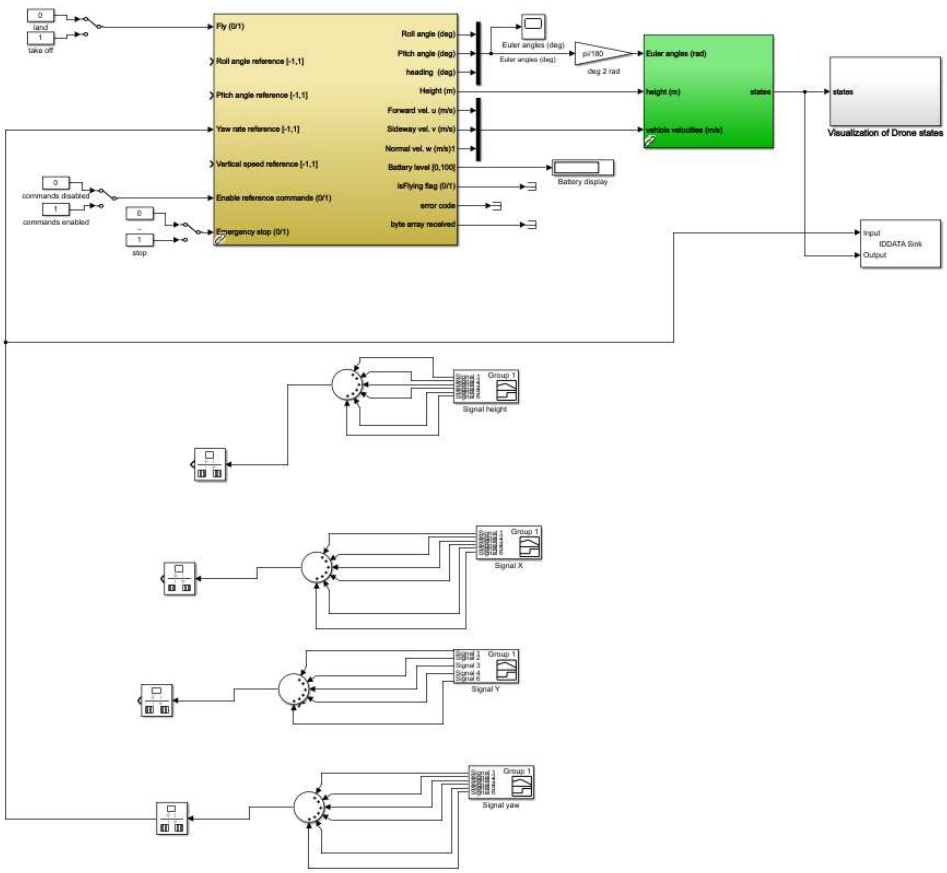


Ilustración 5. Sistema Simulink para la identificación de las señales del dron. Fuente: Elaboración Propia.

Para cada señal de entrada diseñaremos su propia señal de impulsos a diferente tiempo e intensidad entre -1 y 1, el máximo valor que admiten todas las entradas del sistema, y se determinará con un sistema de identificación una función de transferencia; que se validará con datos adicionales a los utilizados para identificar.

Aunque se identifican funciones de transferencia sin estados internos; estas funciones de transferencia se convertirán en sistemas en espacio de estado discretos cuando se realice el controlador. Estudios preliminares del comportamiento del sistema determinan que el funcionamiento del sistema, la matriz de funciones de transferencia es:

$$\begin{bmatrix} Pos\ X \\ Pos\ Y \\ Heading \\ Altura \end{bmatrix} = \begin{matrix} G_{xp} & 0 & 0 & 0 \\ 0 & G_{yr} & 0 & 0 \\ 0 & 0 & G_{hy} & 0 \\ 0 & 0 & 0 & G_{hv} \end{matrix} * \begin{bmatrix} Pitch\ rate \\ Roll\ rate \\ Yaw\ rate \\ Vertical\ speed\ rate \end{bmatrix}$$

3.1.1. Función de transferencia de la altura respecto de la velocidad vertical.

Comenzamos por esta función, y aplicamos la siguiente señal de forma sumada, es decir, las señales de un solo pulso se verán como una señal pulsante en el tiempo, se simulan 100 segundos, tiempo suficiente para estabilizar el dron en varias ocasiones durante la simulación. Los resultados se recogen en un bloque de análisis, que se lleva al *Sistem Identification* de Matlab, donde se opera con los datos.

Ilustración 6. Señal utilizada para la identificación de Ghv. Las diferentes señales se aplican todas a la misma entrada en diferentes intervalos temporales, siendo la entrada real una sola señal. Fuente: Elaboración Propia

En esta imagen se pueden ver los rangos que se utilizarán para identificar y validar la señal, de longitud similar en su extensión en el tiempo, y además en los cambios que experimenta la señal en ese tiempo.

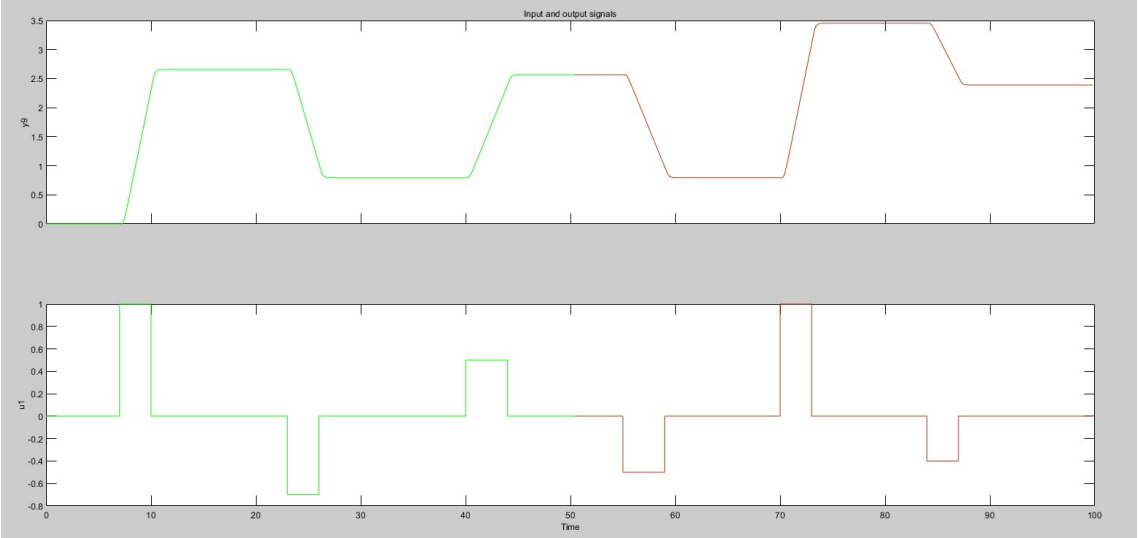


Ilustración 7. Secciones de la señal de entrada y salida utilizadas para identificación (izq.) y validación (der.) de Ghv. Fuente: Elaboración Propia.

La identificación del sistema resulta en la siguiente función, que consta de un polo distinto de 0, un integrador y ningún cero.

$$G_{hv} = \frac{0.8912}{\mathrm{s}(1 + 0.45096\mathrm{s})}\ m$$

Ecuación 16. Identificación de la función de transferencia Ghv. Fuente: Elaboración Propia.

Una vez creamos el modelo, conseguimos el siguiente resultado al superponer los datos de validación y la extrapolación del modelo según la entrada introducida. En azul, podemos ver el modelo en contraste con la señal gris, y la exactitud que Matlab considera que el modelo tiene según los datos de validación.

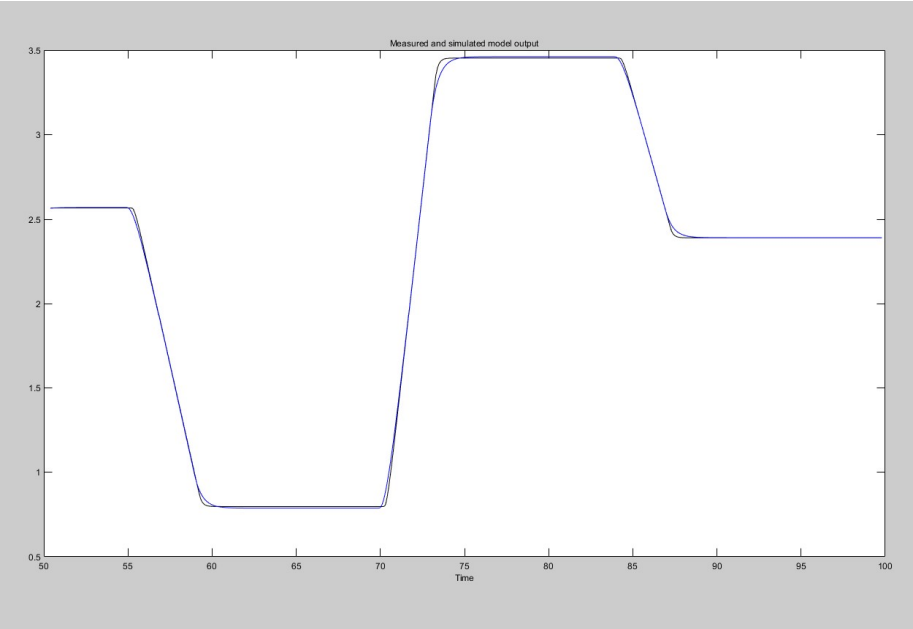


Ilustración 8. Validación del modelo Ghv de con los datos no utilizados en identificación. Fuente: Elaboración Propia.

3.1.2. Función de transferencia de la posición de X respecto del pitch rate.

Variamos la señal anterior para introducir aleatoriedad en la identificación y simulamos otros 100 segundos, tiempo suficiente para estabilizar la posición del dron en varias ocasiones durante la simulación. Los resultados se recogen en un bloque de análisis, que se lleva al Sistem Identification de Matlab, donde se opera con los datos.

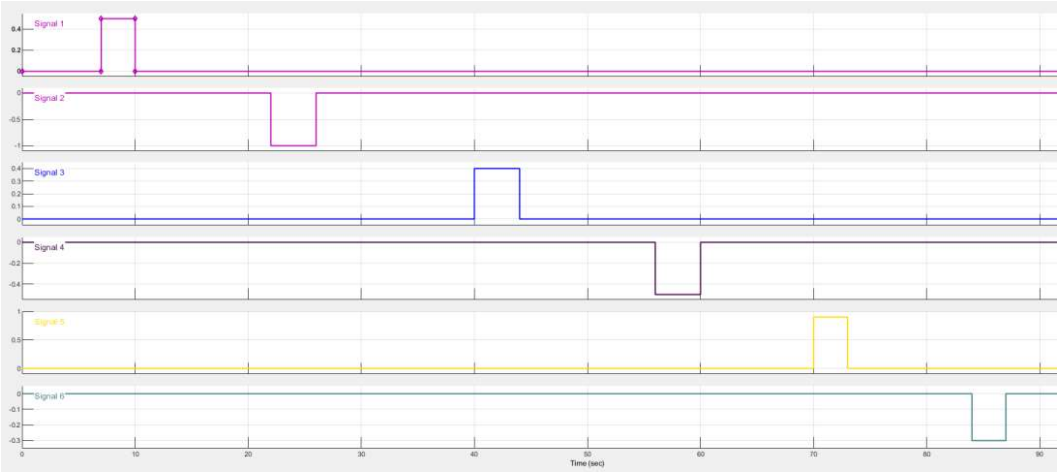


Ilustración 9. Señal utilizada para la identificación de Gxp. Las diferentes señales se aplican todas a la misma entrada en diferentes intervalos temporales, siendo la entrada real una sola señal. Fuente: Elaboración Propia

En esta imagen se pueden ver los rangos que se utilizarán para identificar y validar la señal, de longitud similar en su extensión en el tiempo, y además en los cambios que experimenta la señal en ese tiempo.

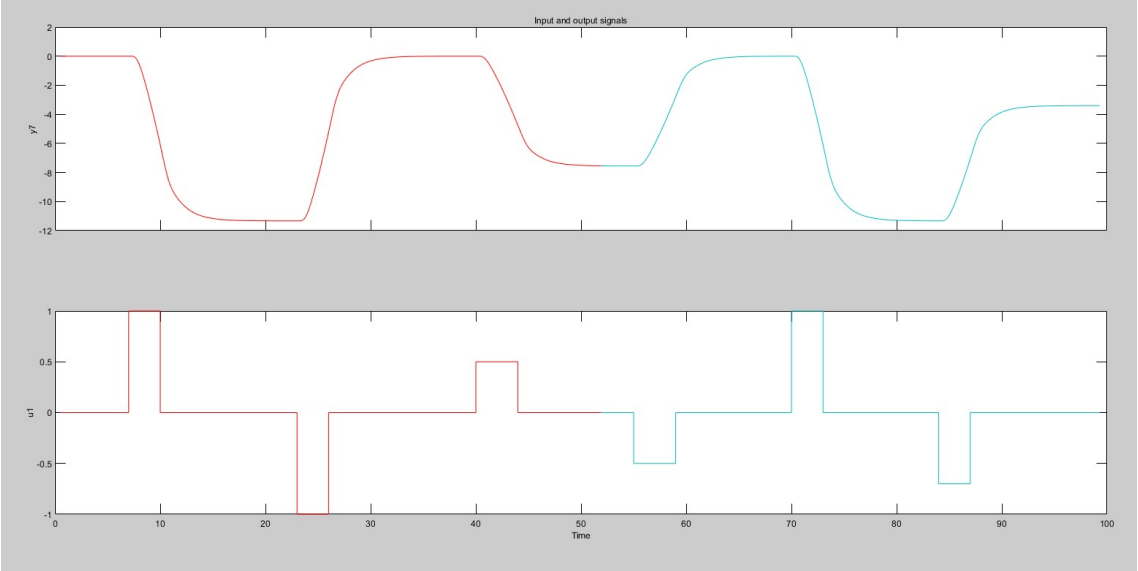


Ilustración 10. Secciones de la señal de entrada y salida utilizadas para identificación (izq.) y validación (der.) de Gxp. Fuente: Elaboración Propia.

La identificación del sistema resulta en la siguiente función:

$$G_{xp} = \frac{-3.8437}{\mathrm{s}(1 + 1.6284\mathrm{s})}\ m$$

Ecuación 17. Identificación de la función de transferencia Gxp. Fuente: Elaboración Propia.

Una vez creamos el modelo, conseguimos el siguiente resultado al superponer los datos de validación y la extrapolación del modelo según la entrada introducida. Como se observa fácilmente, es suficientemente precisa; igual que la anterior.

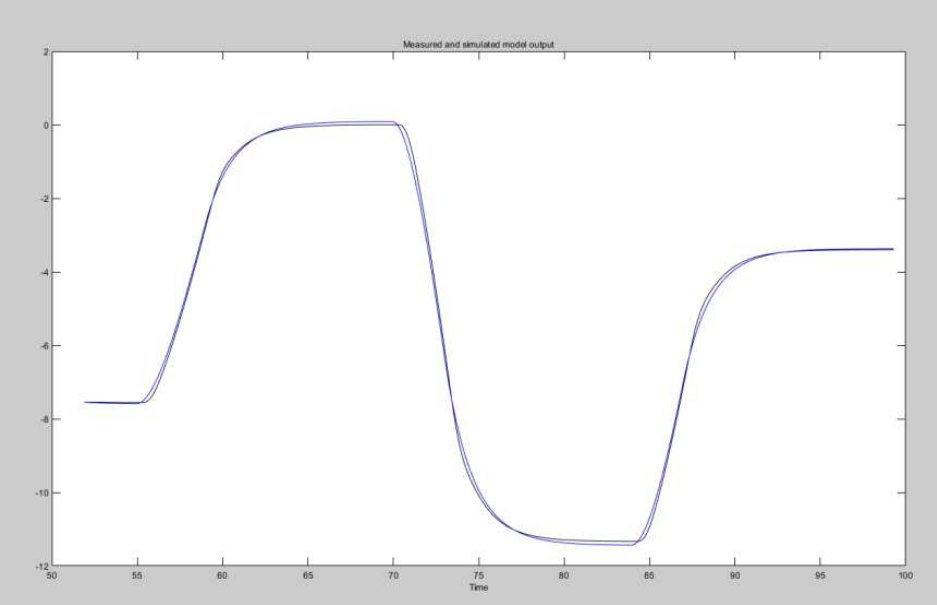


Ilustración 11. Validación del modelo Gxp de con los datos no utilizados en identificación. Fuente: Elaboración Propia.

3.1.3. Función de transferencia de la posición de Y respecto del roll rate.

Variamos otra vez la señal anterior eliminando un impulso para estabilizar la posición del dron en y, ya que en ensayos anteriores la posición no se había estabilizado al ser más lenta. Los resultados se recogen en un bloque de análisis, que se lleva al Sistem Identification de Matlab, donde se opera con los datos.

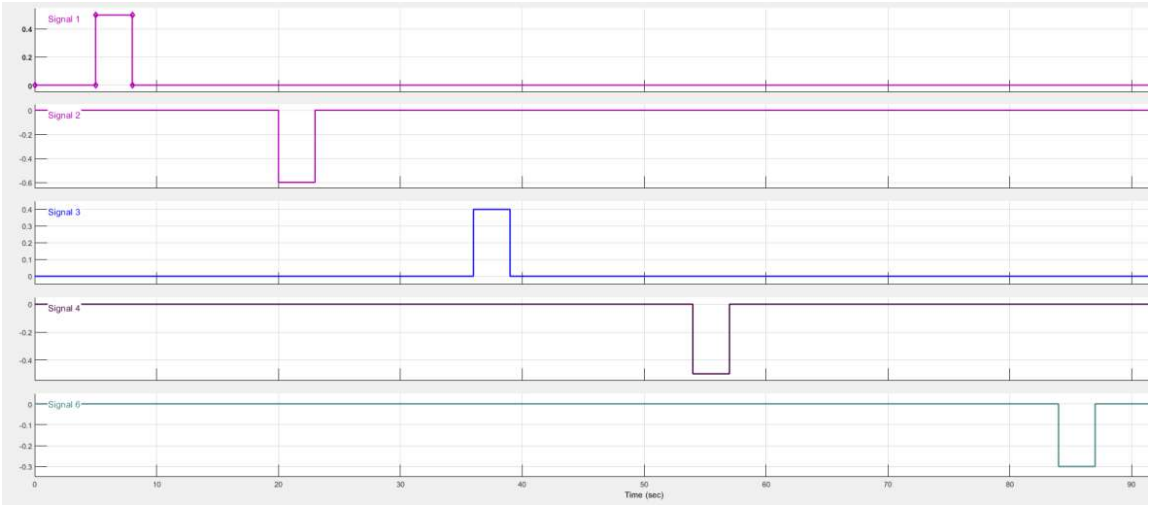


Ilustración 12. Señal utilizada para la identificación de Gyr. Las diferentes señales se aplican todas a la misma entrada en diferentes intervalos temporales, siendo la entrada real una sola señal. Fuente: Elaboración Propia

Aunque los rangos que se utilizarán tienen un impulso menos en esta ocasión, siguen siendo de longitud similar en su extensión en el tiempo, pero el rango de identificación dispone de un cambio más.

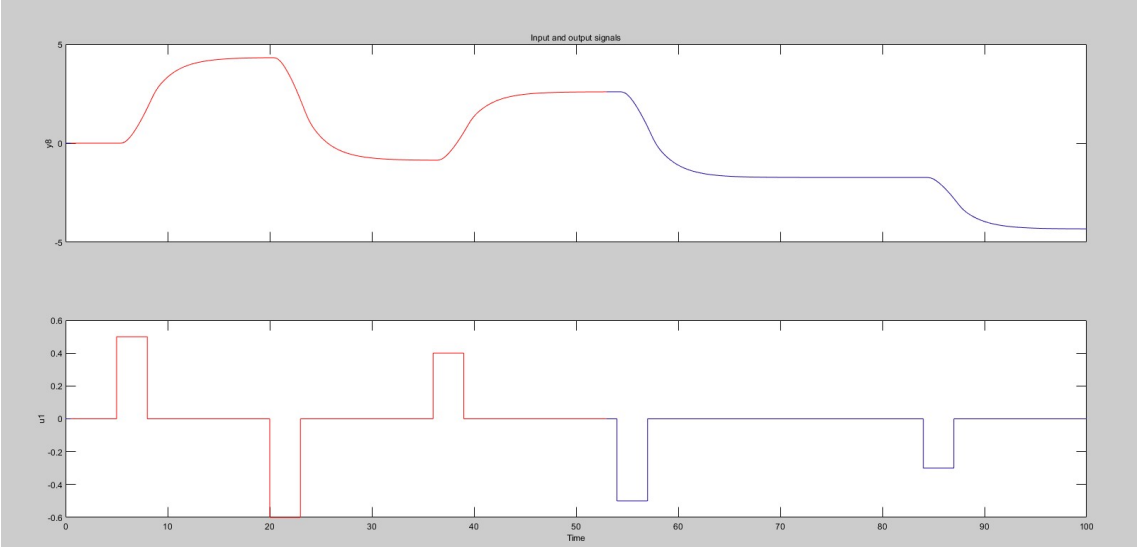


Ilustración 13. Secciones de la señal de entrada y salida utilizadas para identificación (izq.) y validación (der.) de Gyr. Fuente: Elaboración Propia.

La identificación del sistema resulta en la siguiente función:

$$G_{yr} = \frac{2.9747}{\mathrm{s}(1 + 2.4488\mathrm{s})}\ m$$

Ecuación 18. Identificación de la función de transferencia Gyr. Fuente: Elaboración Propia.

Una vez creamos el modelo, conseguimos el siguiente resultado al superponer los datos de validación y la extrapolación del modelo según la entrada introducida. Aunque la discrepancia es más alta al principio, una vez se estabiliza la señal en un nuevo impulso el modelo sigue siendo preciso.

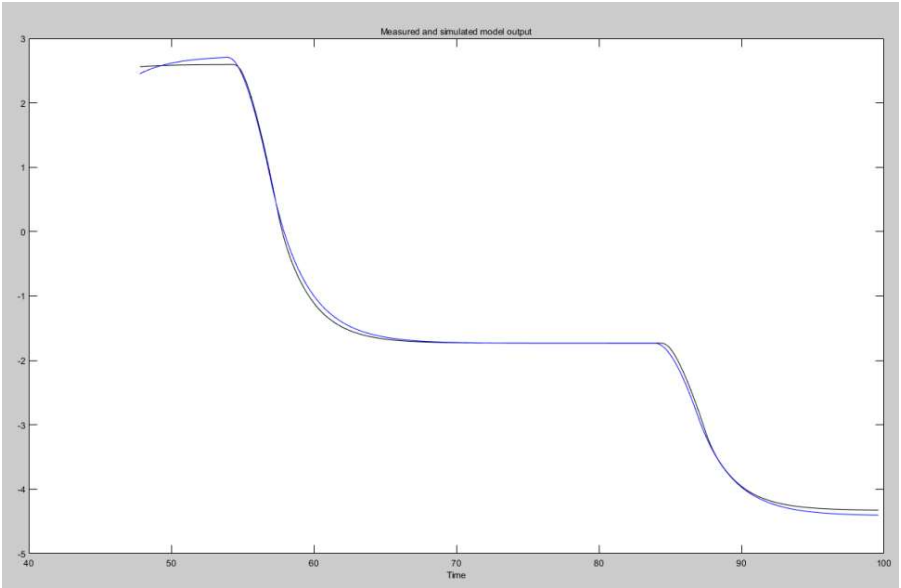


Ilustración 14. Validación del modelo Gyr de con los datos no utilizados en identificación. Fuente: Elaboración Propia.

3.1.4. Función de transferencia del heading respecto del yaw rate

Volvemos a añadir un impulso adicional ya que esta señal no requiere de tanto tiempo de estabilización en esta señal. Los resultados se recogen en un bloque de análisis, que se lleva al "Sistem Identification" de Matlab, donde se opera con los datos.

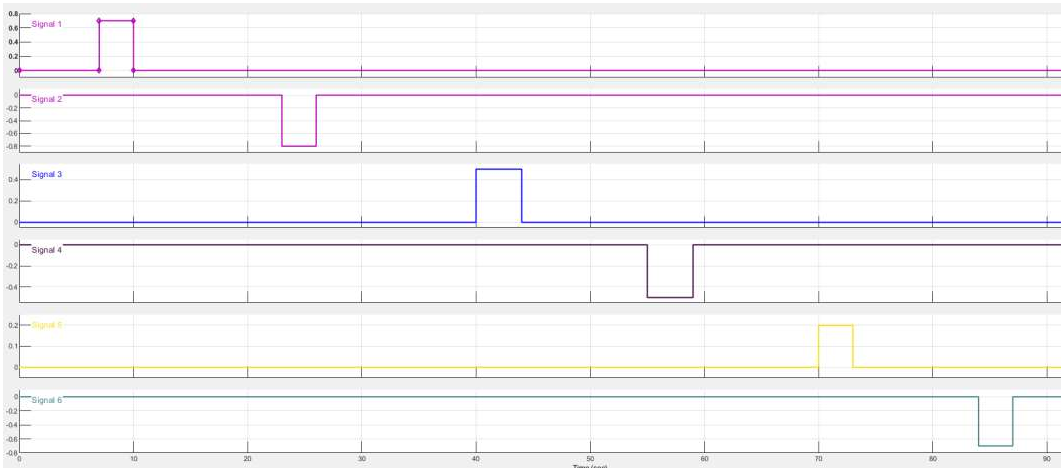


Ilustración 15. Señal utilizada para la identificación de Ghy. Las diferentes señales se aplican todas a la misma entrada en diferentes intervalos temporales, siendo la entrada real una sola señal. Fuente: Elaboración Propia

En esta imagen se pueden ver los rangos que se utilizarán para identificar y validar la señal. En este momento, ya se puede observar que la señal de yaw no se estabiliza en un valor cuando la entrada es 0; sino que desciende en valor ligeramente**.**

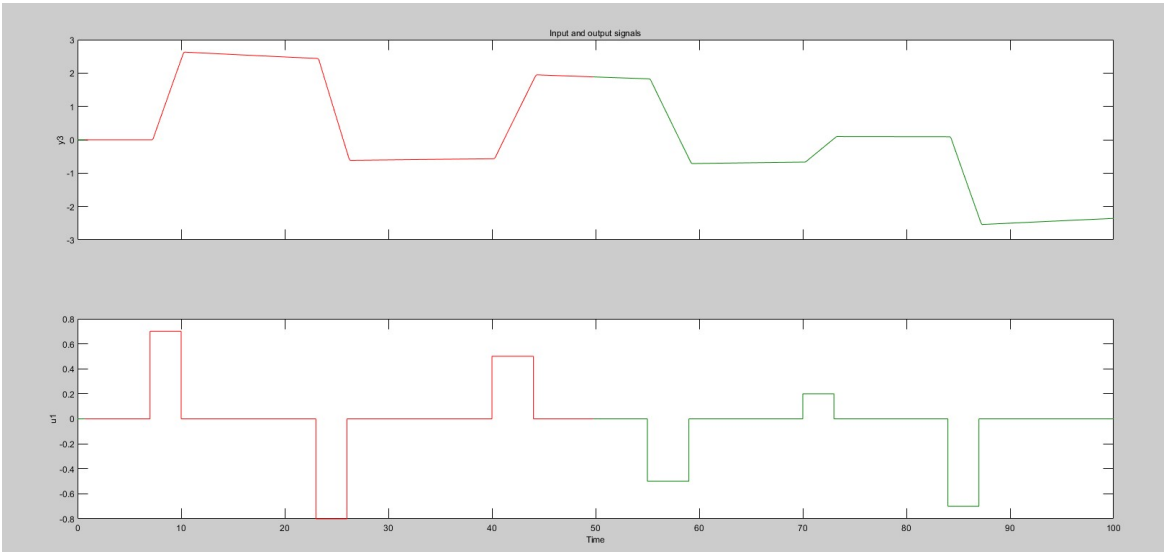


Ilustración 16. Secciones de la señal de entrada y salida utilizadas para identificación (izq.) y validación (der.) de Ghy. Fuente: Elaboración Propia.

La identificación del sistema resulta en la siguiente función, en esta ocasión ha sido necesario introducir un par de polos complejos para tener un modelo representativo:

$$G_{hy} = \frac{1.2625}{\mathrm{s}(1 + 2 * 0.55524 * 0.20151 * \mathrm{s} + (0.20151 * \mathrm{s})^2)}\ rad$$

Ecuación 19. Identificación de la función de transferencia Ghy. Fuente: Elaboración Propia.

Una vez creamos el modelo, conseguimos el siguiente resultado al superponer los datos de validación y la extrapolación del modelo según la entrada introducida. Aunque la discrepancia es la más alta de todas, el modelo aparenta ser válido.

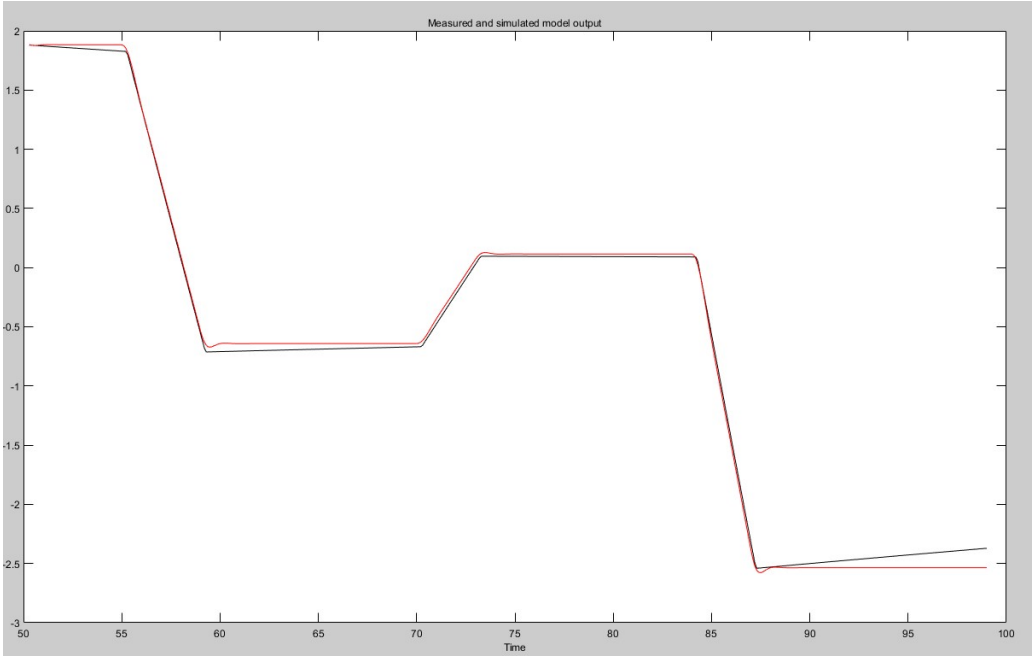


Ilustración 17. Validación del modelo Ghy de con los datos no utilizados en identificación. Fuente: Elaboración Propia.

3.1.5. Prueba preliminar del modelo identificado

Para la comparación de nuestros modelos en función de transferencia con el modelo del dron real, es necesaria una prueba en la cual las entradas a los dos sistemas sean iguales, y monitorizar las salidas dos a dos para poder hacer comparaciones del funcionamiento de forma visual y/o numérica. Para esta prueba utilizamos el esquema Simulink "SS_prueba", una modificación del modelo anterior para incluir una sola señal de entrada y tanto el modelo del dron nuestro como del Development kit para comparar respuestas:

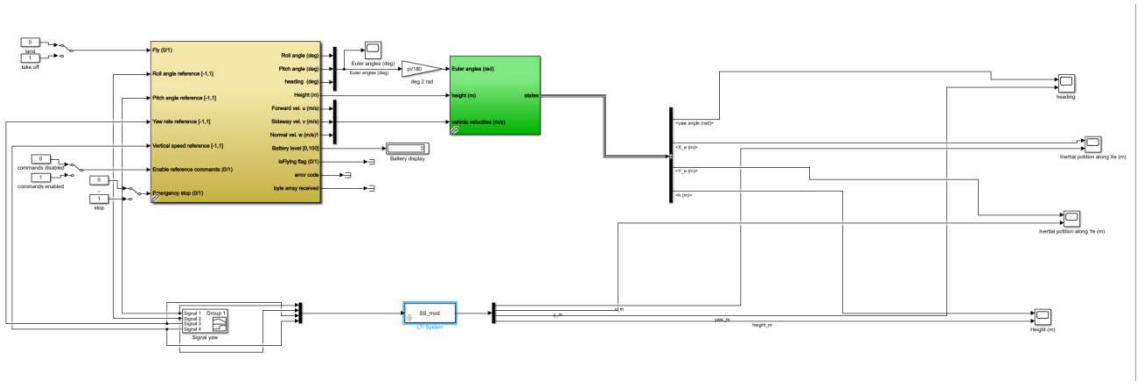
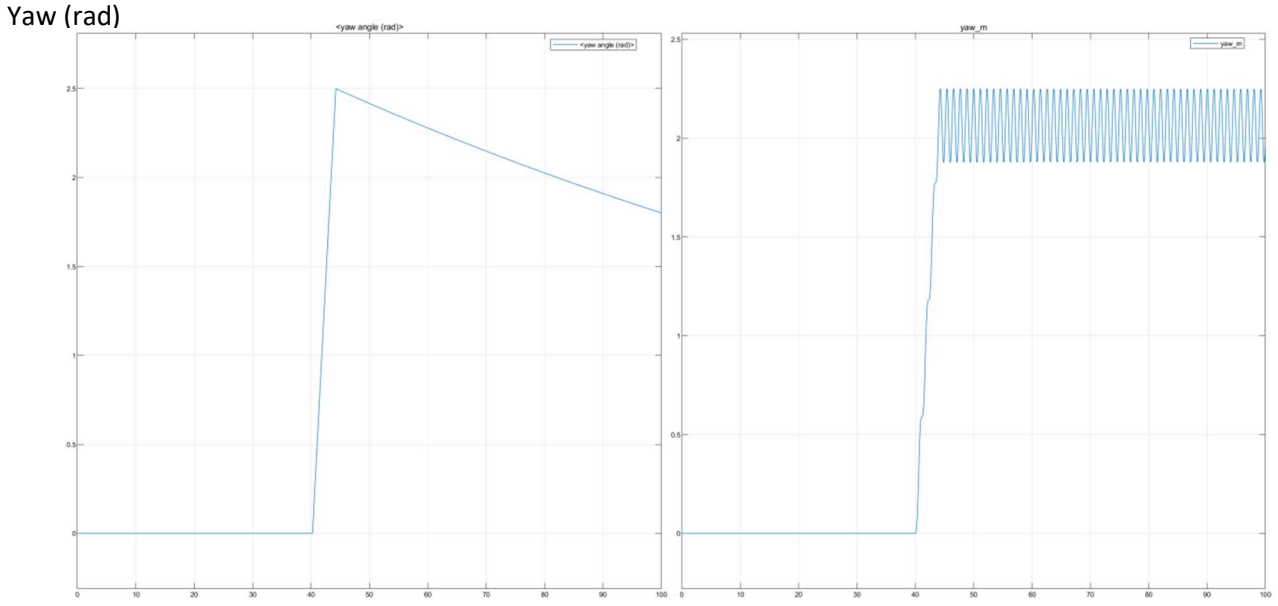


Ilustración 18. Modelo utilizado para la validación del modelo completo. Fuente: Elaboración Propia.

Sin embargo, a la hora de comparar la función de yaw, es claro que el modelo creado no se corresponde de forma aceptable con la salida real, y es necesario volver a identificar esta función.

Gráfica 1. Discrepancia de la función Ghy con el sistema (derecha). Fuente: Elaboración Propia.



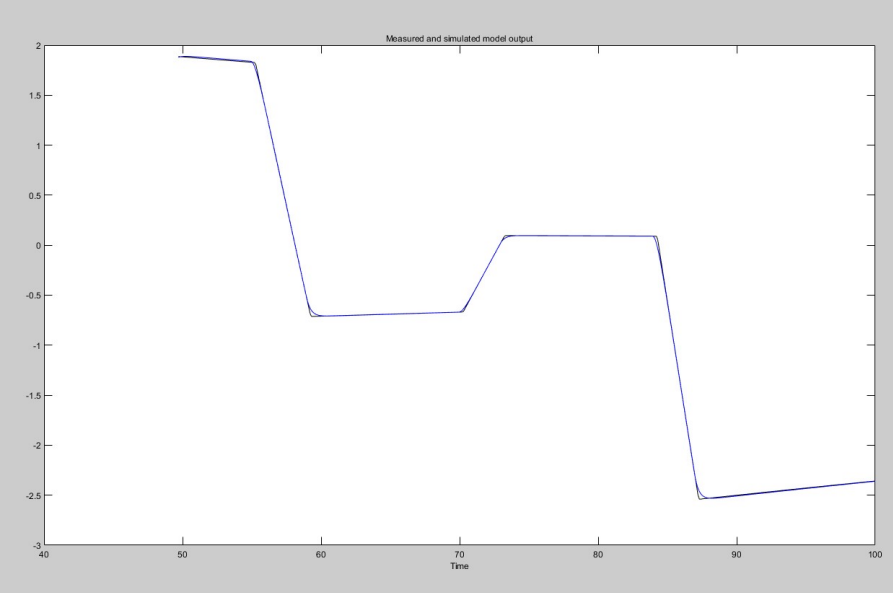


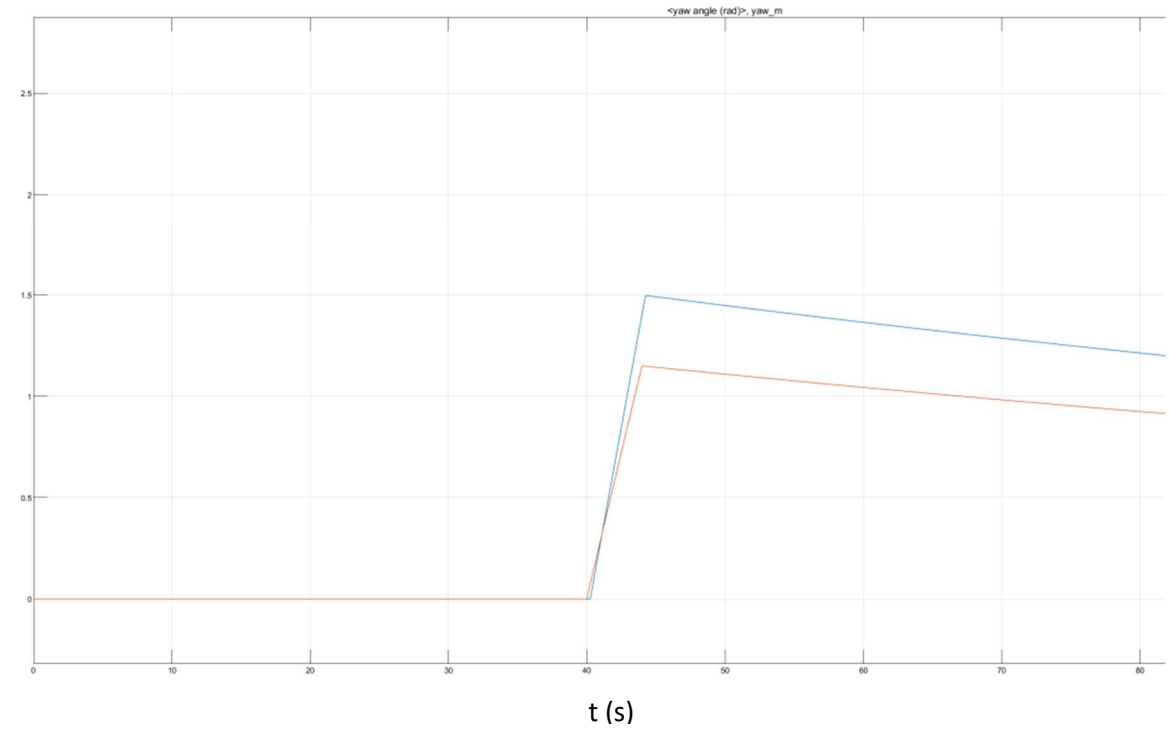
Ilustración 19.Identificación de Ghy como función sin integradores. Fuente: Elaboración Propia.

La identificación del sistema resulta ahora en la siguiente función; donde esta vez no se han buscado integradores en los polos.

$$G_{hy} = \frac{209.6}{(1 + 165.26s)(1 + 0.30732s)}\ rad$$

Ecuación 20. Corrección de la función de transferencia Ghy. Fuente: Elaboración Propia.

Gráfica 2. Nueva prueba para la función Ghy. Fuente: Elaboración Propia.



Hemos conseguido alcanzar una dinámica similar, pero debemos ajustar la ganancia del modelo para conseguir un mejor comportamiento en régimen permanente, **así que se eleva la ganancia a 275 de la función anterior, y resulta:**

$$G_{hy} = \frac{275}{(1 + 165.26s)(1 + 0.30732s)}\ rad$$

Ecuación 21. Forma final de la función de transferencia Ghy. Fuente: Elaboración Propia.

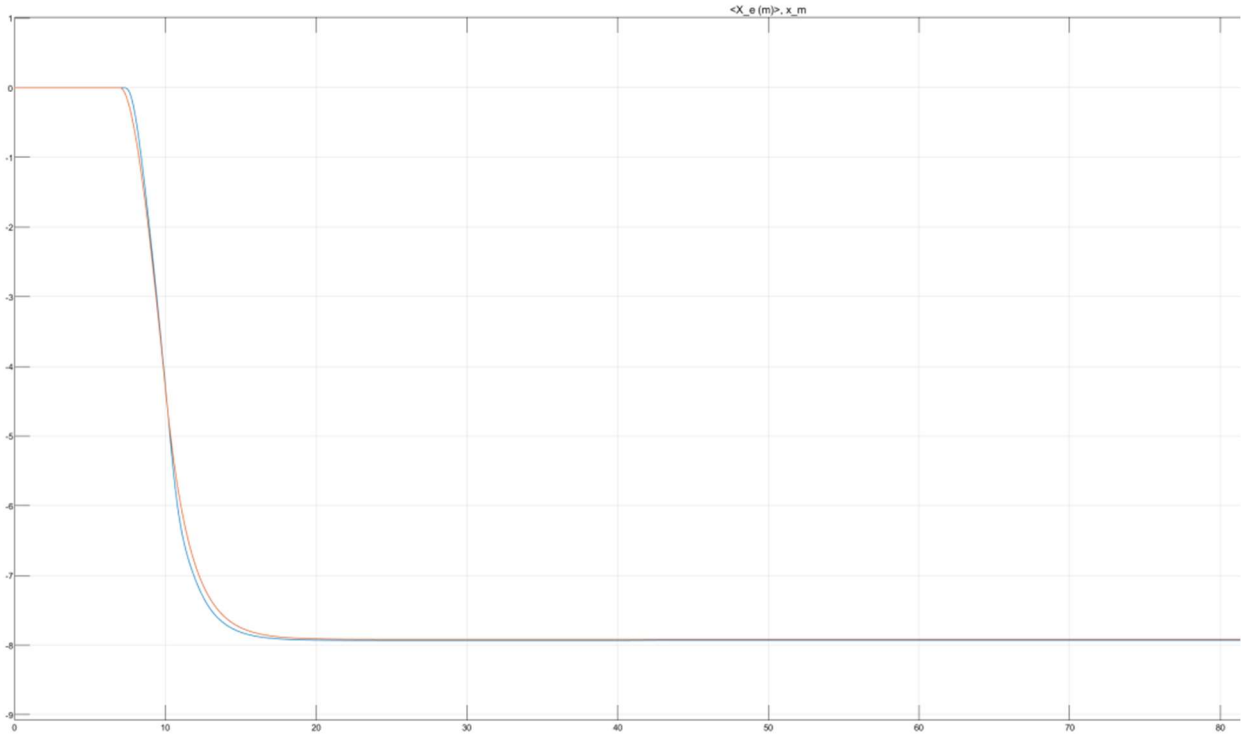
3.1.6. Prueba final del modelo completo

En este apartado se hace la prueba final de comparación entre el modelo y nuestro modelo identificado, y se muestran todos los resultados; además del modelo final declarado en Matlab, que es el siguiente:

```
s=tf('s');
Gxp=-3.77/(s*(1+1.6284*s));
Gyr=2.9/(s*(1+2.4488*s));
Ghy_bueno=275/((1+165.26*s)*(1+0.30732));
Ghv=0.8912/(s*(1+0.45096*s));
G_comp=blkdiag(Gxp,Gyr,Ghy_bueno,Ghv);
```

Gráfica 3. Prueba final de X para el modelo ajustado de forma final. La señal roja es el modelo identificado, y la azul el sistema original. Fuente: Elaboración Propia.

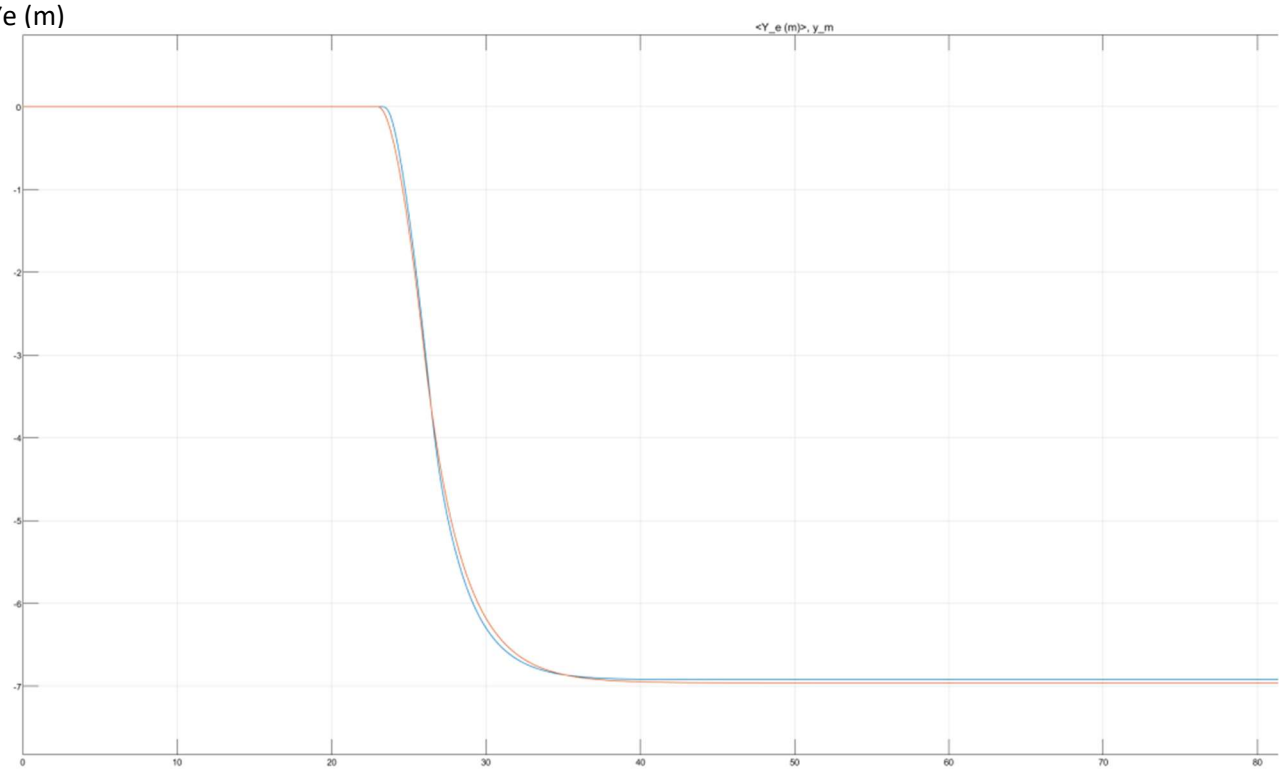
Xe(m)



t (s)

Gráfica 4. Prueba final de Y para el modelo ajustado de forma final. La señal roja es el modelo identificado, y la azul el sistema original Fuente: Elaboración Propia.

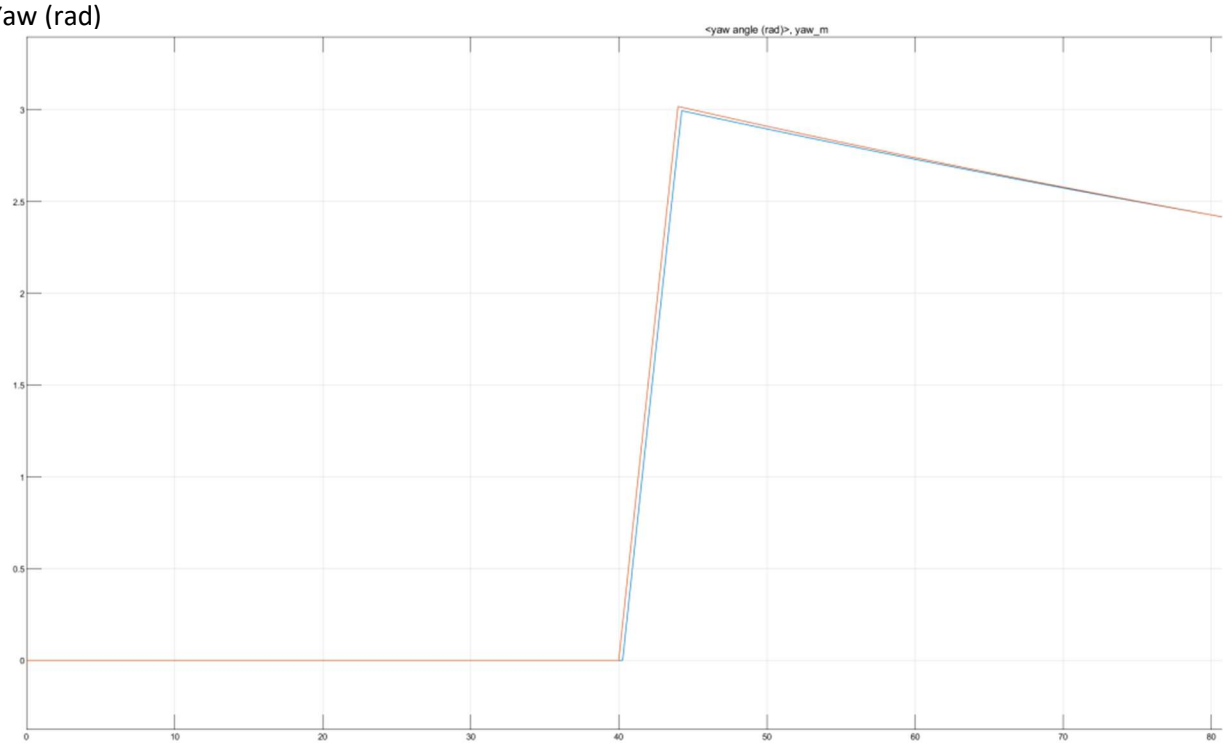
Ye (m)



t (s)

Gráfica 5. Prueba final del heading para el modelo ajustado de forma final. La señal roja es el modelo identificado, y la azul el sistema original. Fuente: Elaboración Propia.

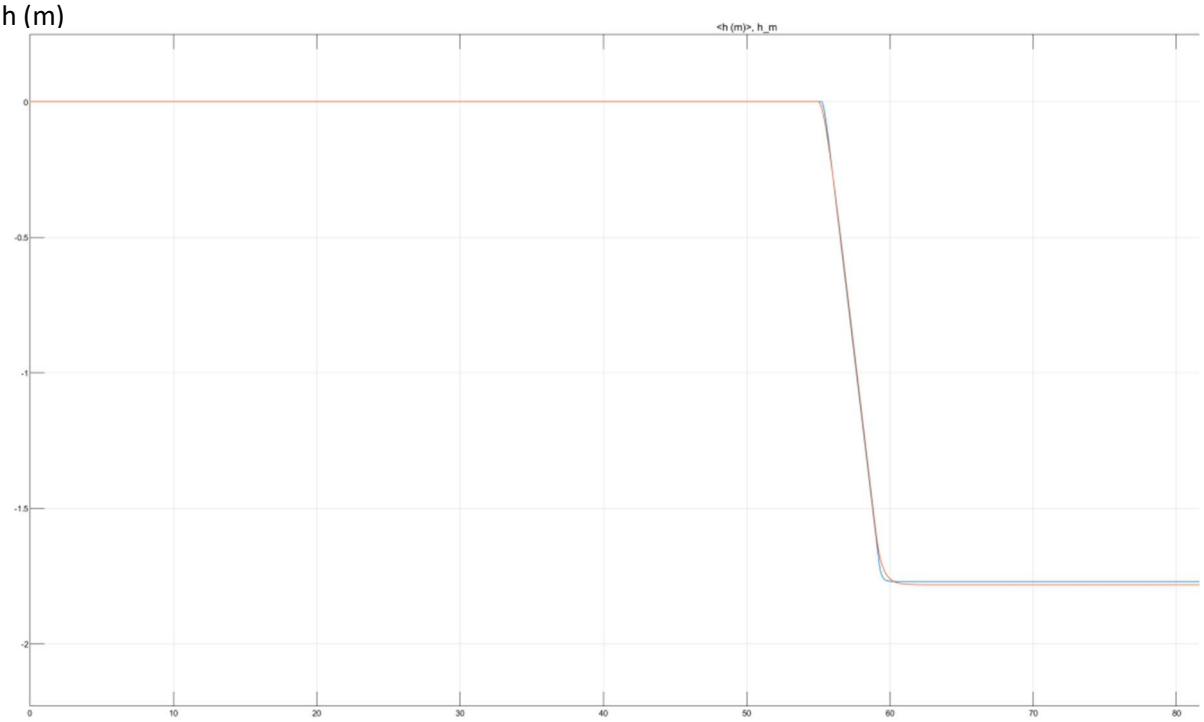
Yaw (rad)



t (s)

Gráfica 6. Prueba final de la altura para el modelo ajustado de forma final. La señal roja es el modelo identificado, y la azul el sistema original Fuente: Elaboración Propia.

h (m)



t (s)

Una vez han sido comprobadas todas las señales, es posible pasar a desarrollarlos prototipos del algoritmo de control y su aplicación final en Simulink.

3.2. ALGORITMOS DE CONTROL DESARROLLADOS E IMPLEMENTACIÓN.

Antes de la implementación total del control se crea un algoritmo preliminar; antes del algoritmo final, sin restricciones cuadráticas. Como el controlador que se va a diseñar necesita ser alimentado con un modelo en espacio de estados discreto; se procede a transformar las funciones de transferencia obtenidas a dicho formato y se implementa una simulación de control en espacio de estados con optimización por mínimos cuadrados. Como ejemplo de funcionamiento se utilizará el archivo que controla Gxp, pero todos son prácticamente iguales a excepción del modelo, y de que los parámetros "c", "p" "alfa" y "lambda" se pueden ajustar a gusto para comprobar su correcto funcionamiento.

3.2.1. Modelo e inicializaciones.

```
%% Modelo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
close all
clear all
s=tf('s');
Ghv=0.8912/(s*(1+0.45096*s));

SS_modb=ss(Ghv);
SS_mod=minreal(SS_modb);

Ts=0.05; %Periodo de muestreo
Gz=c2d(SS_mod,Ts); %Discretización
del modelo
Gzss=ss(Gz);
[A,B,C,D]=ssdata(Gz);

ny = size(C,1); %Número de salidas
nu = size(B,2); %Número de entradas
x=zeros(size(A,2),1);
```

En este apartado de inicialización se declara el modelo, se pasa a espacio de estados y se discretiza con un periodo de muestreo de 50ms; suficiente para el control del dron por la velocidad de su dinámica. Se extraen las 4 matrices que componen el sistema discreto en espacio de estados para los cálculos del controlador. Como el código del algoritmo es general, el número de entradas y salidas es variable, aunque en este caso es 1.

```
%% Inicializacion off-line
%%%%%%%%%%%%%%%%%%%%%%%%%
P=ssP(p,C,A);
Q=Qss(p,C,A,B);
G=Gss(Q,c);

alfaM=alfa1*eye(p);
lambdaM=lambda1*eye(c);

H=inv(G'*alfaM*G+lambdaM)*G'*alfaM;

%Valor inicial de las predicciones
y_pred=zeros(ny,1);
deltau_ant=zeros(nu,1);
deltau_k=zeros(c*nu,1); % No hemos
movido la VM

u_ant=zeros(nu,1);
u_k=zeros(nu,1); % Nuestro proceso
empieza con 0 en la VM
```

En este apartado de declaración de variables; declaramos los parámetros horizonte de predicción (p), que indica el número de instantes de 50ms a predecir; y el horizonte de control (c);que indica el número de instantes de 50ms que resolveremos para el control. También se da un valor a "lambda"; que es el peso o importancia de la variación de variable manipulada, y "alfa", que es el peso o importancia de la llegada a la referencia de la variable controlada.

Una vez se declaran estos parámetros, se inicializan matrices necesarias para el controlador, como son P,Q , G y "H"; definidas en el apartado 2.4 y 2.5. Estas dependen de las matrices del modelo en espacio de estados. Por último, se inicializa todos los vectores de y_pred deltau_ant, "deltau_k", "u_ant" y "u_k" al valor 0.

3.2.2. Bucle principal y resultados iniciales del control

```
instantes=1000; %Instantes de simulación
sp=[1]; %Set Point cualquiera
spv=[];

for j=1:p
      spv=[spv;sp];
end

u_plot=zeros(instantes,nu);
deltau_plot=zeros(instantes,nu);

%%%Simulamos%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
for i=1:instantes
    y_real=C*x; % Medir la salida



    bias=y_real-y_pred; %Calculamos el
error
    biasv=[];

    for j=1:p
      biasv=[biasv;bias];
%       biasv=[biasv;ones(p,1).*bias(j)];
    end



    deltau_k=H*(spv-P*x-Q*u_ant-
1.*biasv);



    deltau_k=deltau_k(1:nu);
    u_k=u_ant+deltau_k; %Cambio en la
entrada respecto u_ant

    x=A*x+B*u_k; %Calculamos el valor de
x para el siguiente instante
    y_pred=C*x;
    deltau_ant=deltau_k;
```

Figura 5. Bucle de control simulado sin el uso del "AR Development Kit" para la prueba del algoritmo. Fuente: Código propio.

En este apartado se expondrá el bucle principal donde se realiza una simulación sin el uso del "AR drone Development Kit"; que comprobará el funcionamiento correcto del control de segundo nivel de forma preliminar a la aplicación de este al modelo

Al inicializar se utiliza un bucle de simulación de 1000 instantes de 50ms (50 s), donde primero; fuera de este; se crea una referencia de llegada para el sistema.

Una vez en el bucle calculamos la y_real, **que en un controlador real sería una entrada al algoritmo**, pero aquí lo calculamos como igual que y_pred; ya que no tenemos con que comparar la señal del modelo; y por tanto la resta de los 2 es 0. Una vez el bias es un vector tamaño "p", podemos calcular "deltau_k" como solución de un problema de mínimos cuadrados, y por último calculamos "u_k" a partir de "deltau_k", la nueva x para la nueva "u_k" y actualizamos "deltau_ant" y "u_ant".

Durante la simulación, creamos variables especiales para crear una gráfica del resultado en Matlab y ser capaces de ver el funcionamiento del algoritmo de forma gráfica. Una vez se ha salido del bucle se crea una figura donde se puede observar la evolución de la variable manipulada, la variable controlada y las variaciones de la variable controlada en cada instante de simulación.

```
for j=1:nu
    y_plot(i,:)=y_real';
    deltau_plot(i,j)=deltau_k(j); % Guardamos las
variaciones de las acciones de control para graficar
    u_plot(i,j)=u_k(j); % Guardamos las acciones de
control para graficar
    end
end

%% Representacion grafica
figure
subplot(311)
plot(y_plot(:,1))
hold on
plot(sp(1,1).*ones(instantes,1),'-.')
title('Salida 1 del sistema')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(312)
plot(u_plot(:,1))
title('U1 del sistema')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(313)
plot(deltau_plot(:,1))
title('DeltaU 1 del sistema')
```

Figura 6. Creación de las gráficas de resultados. Fuente: Código propio.

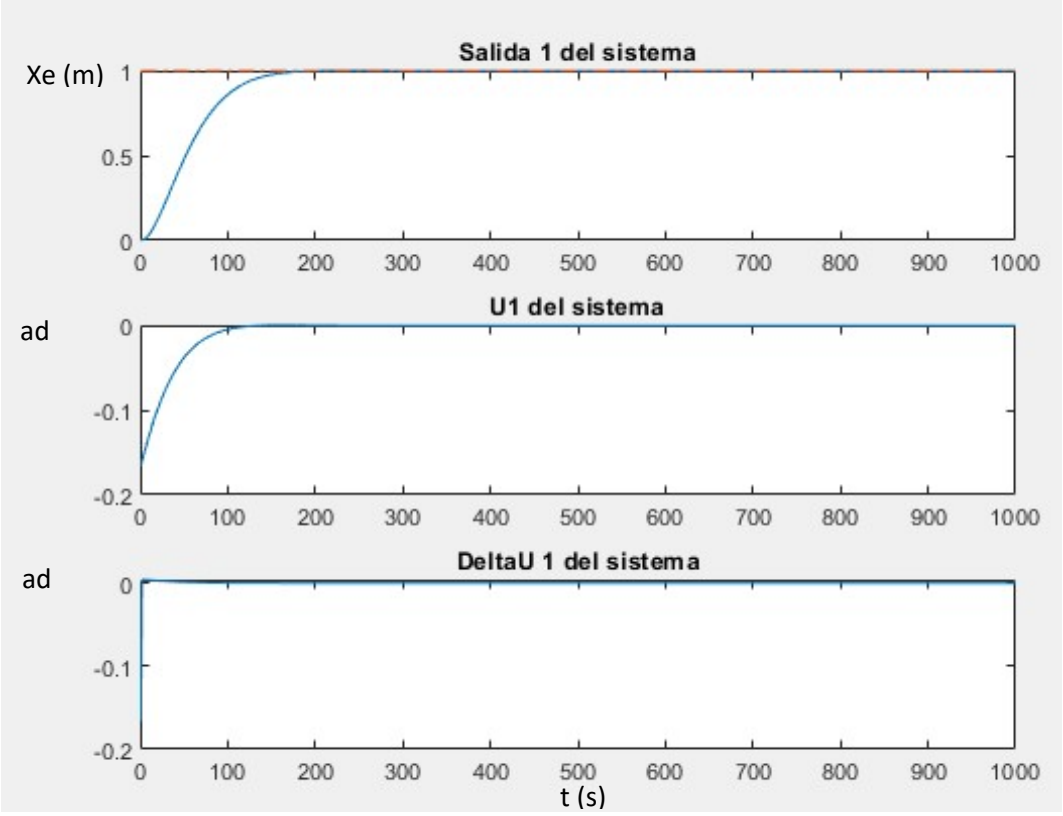


Ilustración 20. Resultados del primer algoritmo. Fuente: Elaboración Propia

3.2.3. Aplicación de restricciones cuadráticas

Se construyen las restricciones para el problema cuadrático de acuerdo con el concepto teórico visto en el apartado 2.6. El vector "b" se declara en 2 partes: 1 parte con todo aquello posible de saber antes de empezar el control, y otra parte que requiere de información adicional dentro del bucle de control.

La matriz I es la matriz identidad y IL es la matriz triangular inferior del tamaño de "c". La matriz G es la misma que construimos en el desarrollo teórico en el apartado.

En el código b está construido sin restar la "u" anterior ni la respuesta libre, que serán restados en el bucle, cuando las determinemos.

```
I=blkdiag(eye(c)); %Tantos bloques como nu para I e IL
IL=blkdiag(tril(ones(c)));
boff=[deltaumax*ones(c,1);-deltaumin*ones(c,1)];
boff2=[umax*ones(c,1);-umin*ones(c,1);ymax*ones(ny*p,1);-
ymin*ones(ny*p,1)];
Aq=[I zeros(nu*c,1);-I zeros(nu*c,1);IL zeros(nu*c,1);-IL zeros(nu*c,1);
G -1*ones(nu*p,1);-G -1*ones(nu*p,1)];
Aq=[Aq;zeros(1,nu*c), -1];
```

Figura 7. Declaración de las restricciones cuadráticas del algoritmo. Fuente: Código Propio.

En el bucle se calculan los parámetros necesarios para el índice de programación cuadrática, que son "cq", "H", "A" y "b". Sin embargo; tan solo hace falta calcular "cq" y actualizar "b"; ya que el resto son calculados antes del bucle. El resto del código se mantiene de forma idéntica al MPC no cuadrático. En el comando "quadprog" desactivamos las opciones de output de la ventana de comandos y **se modifica el display tan solo cuando hay una no factibilidad en la solución**.

```
y_free=P*x+Q*u_ant+1.*biasv;
    error=spv-y_free;
    cq=[-G'*alfaM*error;0];
    b=[boff;boff2-[u_ant(1)*ones(c,1);-u_ant(1)*ones(c,1);y_free;-
y_free];0];


    options = optimoptions('quadprog','Display','off');
    [deltau_k,fval,exitflag]
=quadprog(H,cq,Aq,b,[],[],[],[],[],options);
    if exitflag==-2
        disp('PROBLEMA NO FACTIBLE')
    end
    %deltau_k=quadprog(H,cq,Aq,b);

    deltau_k=deltau_k(1:nu);

    u_k=u_ant+deltau_k; %Cambio en la entrada respecto u_ant

    deltau_ant=deltau_k;
     x=A*x+B*u_k; %Calculamos el valor de x para el siguiente instante
     y_pred=C*x;
    u_ant=u_k;
```

Figura 8. Nuevo cálculo cuadrático de la solución. Fuente: Código Propio.

Una vez tenemos los límites impuestos, calculamos la simulación con los mismos instantes, y conseguimos la respuesta en gráfica del algoritmo. Podemos observar una mejor eficacia del sistema anterior respecto del MPC lineal ya que aquí **"c" se puede aumentar mucho más sin provocar desequilibrios o saturaciones de la acción de control**

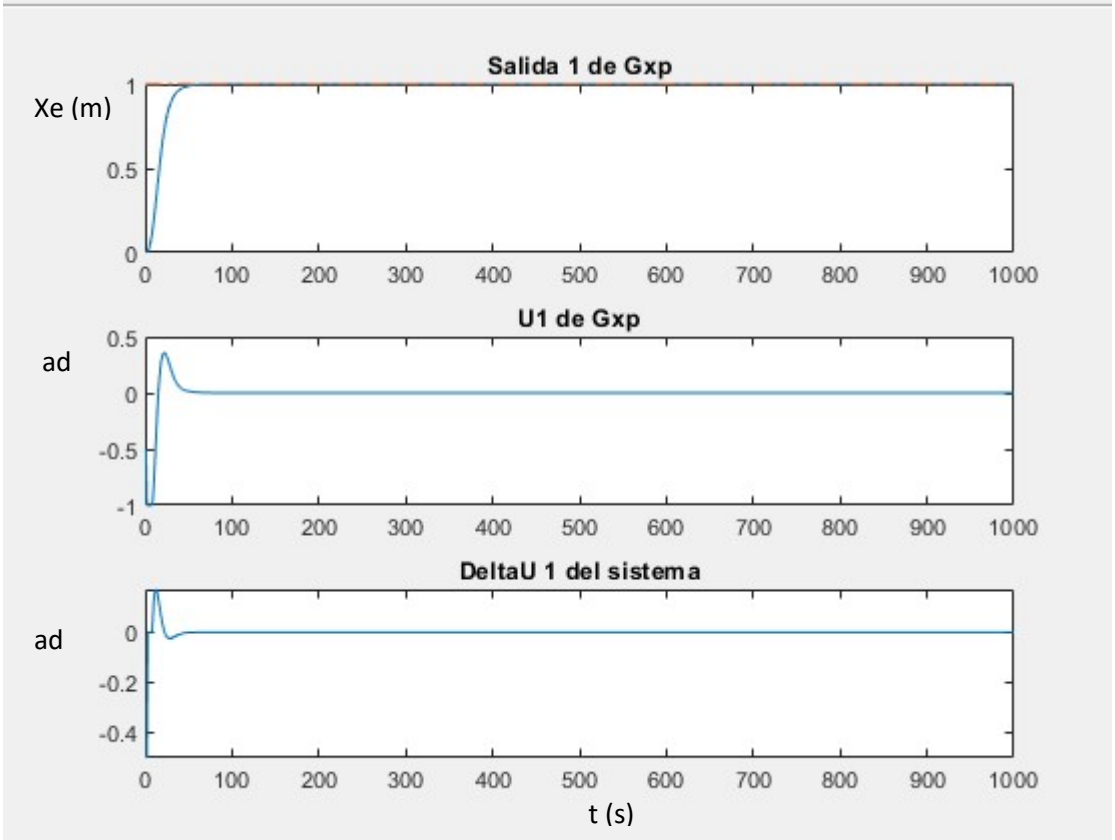


Ilustración 21. Gráficas de resultados con restricciones cuadráticas. Fuente: elaboración propia.

3.3. IMPLEMENTACIÓN DEL ALGORITMO EN SIMULINK

En este apartado utilizaremos el código probado y testeado con scripts de Matlab para probar con el modelo identificado de Simulink como el equivalente al dron real. Para ello necesitamos trasladar el funcionamiento del algoritmo a una función de Matlab que haga de controlador, y un script que inicialice todos los parámetros que pueda necesitar nuestro controlador, como las matrices "H", G y "A".

Por simplicidad y para conseguir una optimización del código, se ha decidido utilizar un solo script para que construya las matrices para los 4 controladores, y declare las variables a pasar a cada uno de los controladores como globales.

Este código es prácticamente una repetición cuádruple de los códigos anteriores de inicialización, cambiando nombres para que las variables no se sobrescriban, y se puedan ajustar completamente los 4 controladores de forma independiente. Esto tan solo es posible gracias a que la matriz de funciones de transferencia del sistema es diagonal, y no hay interacción entrada-salida no deseada.

```
%%Gxp
deltaumax1=1;
deltaumin1=-1;
%%Gyr
deltaumax2=1;
deltaumin2=-1;
%%Ghy
deltaumax3=1;
deltaumin3=-1;
%%Ghv
deltaumax4=1;
deltaumin4=-1;
%%%%%%%
%%Gxp
ymax1=30;
ymin1=-10;
%%Gyr
```

```
%Gxp
P1=ssP(p1,C1,A1);
Q1=Qss(p1,C1,A1,B1);
G1=Gss(Q1,c1);
%Gyr
P2=ssP(p2,C2,A2);
Q2=Qss(p2,C2,A2,B2);
G2=Gss(Q2,c2);
%Ghy
P3=ssP(p3,C3,A3);
Q3=Qss(p3,C3,A3,B3);
G3=Gss(Q3,c3);
%Ghv
P4=ssP(p4,C4,A4);
Q4=Qss(p4,C4,A4,B4);
G4=Gss(Q4,c4);
```

```
boff1=[deltaumax1*ones(c1,1);-deltaumin1*ones(c1,1)];
boff2_1=[umax1*ones(c1,1);-umin1*ones(c1,1);ymax1*ones(ny*p1,1);-
ymin1*ones(ny*p1,1)];
%%%%%%
boff2=[deltaumax2*ones(c2,1);-deltaumin2*ones(c2,1)];
boff2_2=[umax2*ones(c2,1);-umin2*ones(c2,1);ymax2*ones(ny*p2,1);-
ymin2*ones(ny*p2,1)];
%%%%%
boff3=[deltaumax3*ones(c3,1);-deltaumin3*ones(c3,1)];
boff2_3=[umax3*ones(c3,1);-umin3*ones(c3,1);ymax3*ones(ny*p3,1);-
ymin3*ones(ny*p3,1)];
%%%%%
boff4=[deltaumax4*ones(c4,1);-deltaumin4*ones(c4,1)];
boff2_4=[umax4*ones(c4,1);-umin4*ones(c4,1);ymax4*ones(ny*p4,1);-
ymin4*ones(ny*p4,1)];
```

Figura 9. Extractos (no completos) de la declaración de los parámetros iniciales de los cuatro controladores. Fuente: Código Propio

En Matlab, para conseguir que estas variables lleguen a la función deseada se utilizan variables globales. Las variables necesarias como globales son varias. Primero, el vector “b”, en 2 partes, una de ellas con la necesidad de actualizarse cada ciclo. La matriz “A” de restricciones, la matriz “alfa” con los pesos del índice a optimizar y las matrices “G”, “P” ,”Q” y “H”; necesarias en el algoritmo. La matriz C del sistema, para calcular la respuesta del modelo. El horizonte de predicción y control (“p” y “c”).Por último, la inicialización de la u(k-1), o “u_ant”.

Para la función de control de cada uno de los parámetros del dron, se declaran globales las mismas variables que en el script anterior para que las reciba correctamente. Una vez en la función; el programa calcula la resolución del problema cuadrático para cada instante de tiempo con los datos de salida real y del observador pertinente. Para el caso de la función de Gxp, queda así:

```
function u_k=qs_ssfunc1(input)
global Aq1 c1 boff1 boff2_1 alfaM1   H1 C1 nu P1 Q1  G1 p1
sp=input(1);
y_real=input(2);
x_obs=[input(3); input(4)];
spv=[];
persistent u_ant1;


if isempty( u_ant1)
    u_ant1=0;
end


for j=1:p1
      spv=[spv;sp];
end
biasv=[];
y_pred=C1*x_obs;
bias=y_real-y_pred; %Calculamos el error
for j=1:p1
      biasv=[biasv;bias];

end

    y_free=P1*x_obs+Q1*u_ant1+1.*biasv;




    error=spv-y_free;
    cq=[-G1'*alfaM1*error;0];
    b=[boff1;boff2_1[u_ant1(1)*ones(c1,1);u_ant1(1)*ones(c1,1);y_free;y_free];0];

   options = optimoptions('quadprog','Display','off');
    [deltau_k,~,exitflag] =quadprog(H1,cq,Aq1,b,[],[],[],[],[],options);
    if exitflag==-2
        disp('PROBLEMA NO FACTIBLE')
    end

    deltau_k=deltau_k(1:nu);

    u_k=u_ant1+deltau_k; %Cambio en la entrada respecto u_ant

    %deltau_ant=deltau_k;

    u_ant1=u_k;
end
```

Figura 10. . Función implementada en el controlador de la posición de X con el roll del sistema. . Fuente: Código Propio.

Para la implementación del control, se añade al código de inicialización la creación de observadores del estado. Para conseguir el valor de los estados reales del sistema a controlar, se utilizan observadores de orden completo, discretos. Sus polos son suficientemente rápidos para captar la dinámica del sistema que deben observar con rapidez suficiente para que el controlador no tenga problemas de aliasing. El funcionamiento del observador esta integrado en el controlador como se ve en la figura 1 de la memoria. Su formulación teórica y implementación son la siguiente:

$$A_{Ob} = A - L \cdot C$$
$$B_{Ob} = \begin{bmatrix} B & L \end{bmatrix}$$
$$C_{Ob} = I_{n \times n}$$
$$D_{Ob} = \begin{bmatrix} 0_{n \times m} & 0_{n \times p} \end{bmatrix}$$

```
polos=[0.9 0.92];
L=place(A1',C1',polos)';
Aob=A1-L*C1;
Bob=[B1 L];
Cob=eye(2);
Dob=[zeros(2,1) zeros(2,1)];
Ob1=ss(Aob,Bob,Cob,Dob,0.05);
```

Figura 11. Formulación teórica del observador de orden completo no actualizado. A,C y B son las matrices del sistema en espacio de estado a controlar, y L la corrección del error de estimación, parámetro de diseño. Fuente: Elaboración propia.

Para esta aplicación se ha decidido utilizar observadores discretos en espacio de estados completos no actualizados, ya que la diferencia entre 2 instantes de control es muy pequeña, de 50ms de diferencia. Además, se ha observado una dinámica aceptable con los observadores no actualizados, así que no es necesario hacer más compleja esta parte del diseño.

Para el modelo de la ilustración; en el que se prueba el control cuadrático sin la generación de trayectorias para probar el seguimiento de una referencia constante disponemos del modelo del dron en amarillo; del estimador en verde y la estructura de control en forma de bloques de función que cierran el bucle de control para las cuatro entradas, queda un sistema tal que así, para el bucle de la primera entrada y salida (Xe y pitch):

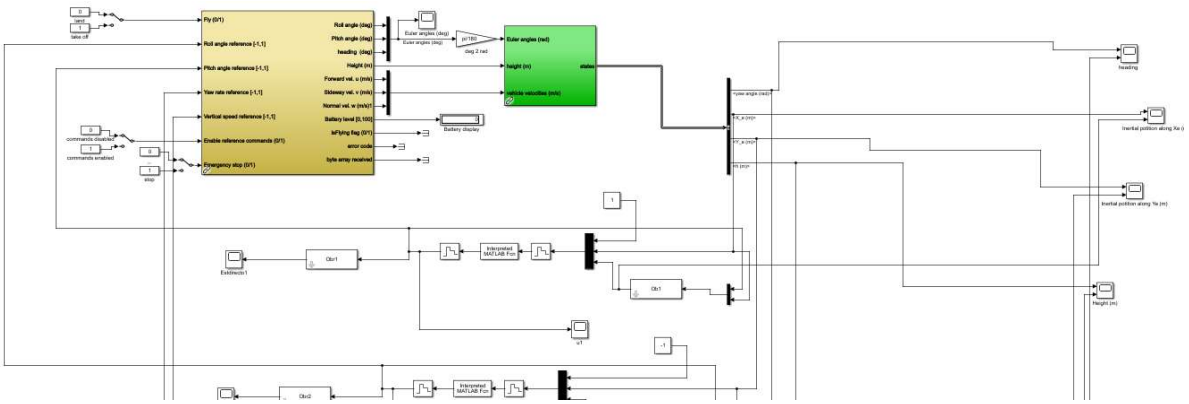


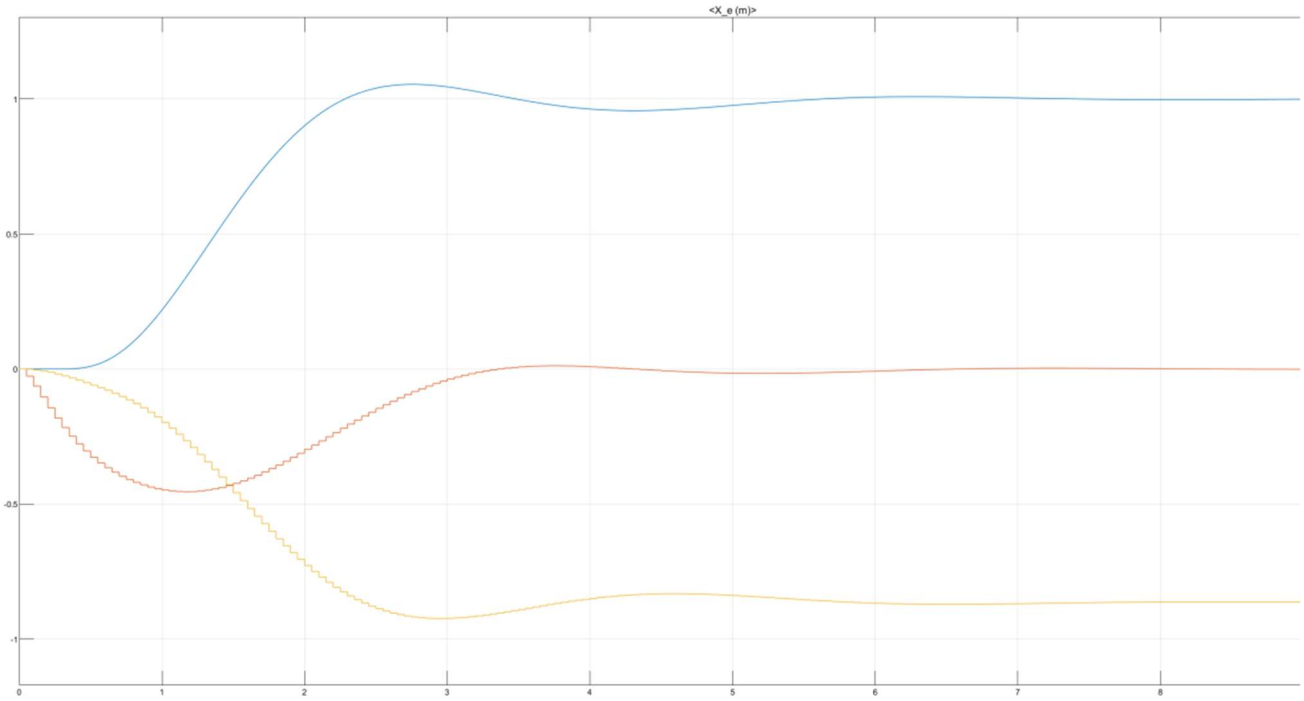
Ilustración 22. Imagen parcial de la prueba de controlador aplicado al sistema con el modelo en amarillo y el estimador en verde. Debajo de los bloques se observa el controlador de Gxp. Fuente: Elaboración propia.

En este diagrama Simulink, controlamos el modelo de dron que hemos identificado de con cuatro funciones de control predictivo cuadrático SISO. Están limitamos por bloques de discretización de orden cero a que tan solo den una respuesta nueva y reciban nuevas entradas cada 50ms de simulación; para discretizar su funcionamiento al período de muestreo. Estas funciones reciben la referencia de control, la salida del sistema controlado, y los estados estimados por el observador discreto, y dan una acción de control. Además, introducimos la misma entrada en otra función no controlada del modelo SISO con los estados incluidos como salidas; para comprobar la estimación realizada.

En esta simulación se miden las salidas reales del sistema controlado, los estados estimados de un modelo sin controlar (bucle abierto) y los estimados gracias a la salida del sistema controlado, y la acción de control dada por la función de control de segundo nivel. Aquí podemos ver un ejemplo de su funcionamiento, para referencia de 1 en X; donde la salida es la señal azul y sus estados proporcionales a esta o su derivada:

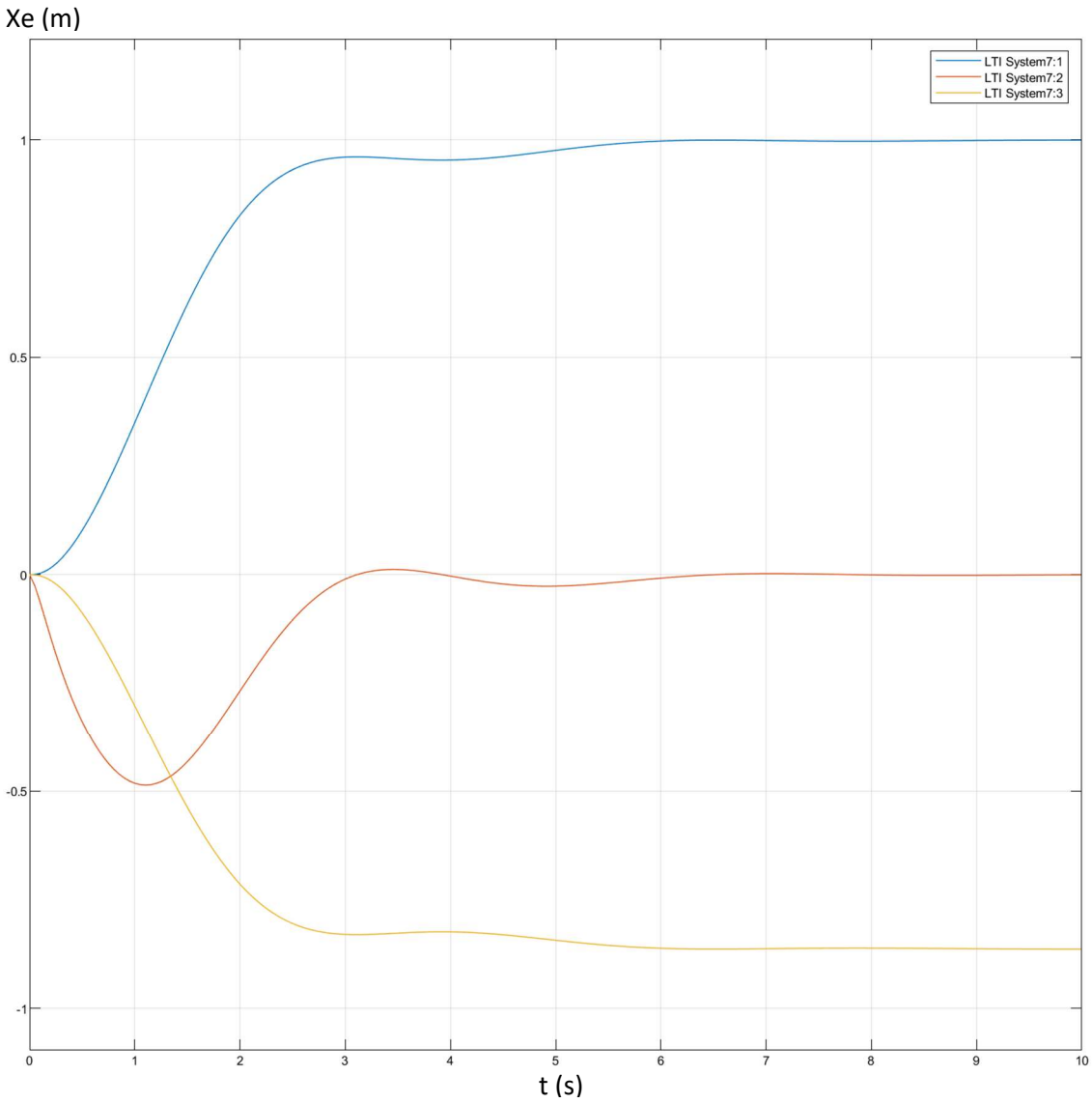
Gráfica 7. Salida (en azul) y estados observados (otras dos señales)del modelo controlado para X. Fuente: Elaboración propia.

Xe (m)



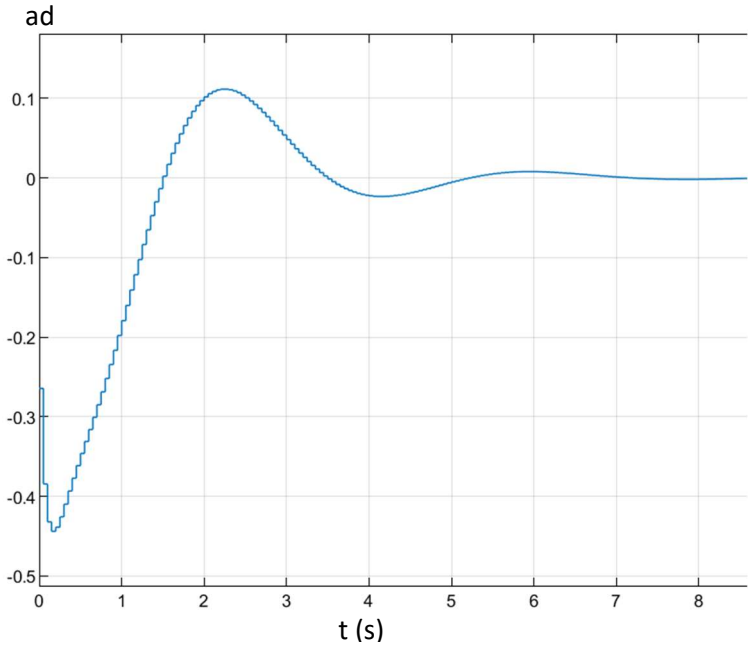
t (s)

Gráfica 8. Salida del modelo no controlado en espacio de estados de X (en azul), con los estados incluidos como salidas; que son las señales roja y amarilla. Fuente: Elaboración propia.



Se puede observar que las dos señales son muy similares en su valor final, aunque su valor en el tiempo difiera de forma notable. Esto se debe probablemente a discrepancias entre el modelo identificado y la señal del dron que serán compensadas por el factor de error constante desarrollado en el algoritmo de control.

Gráfica 9. Acción de control realizada por el controlador en pitch. Fuente: Elaboración propia.



3.4. AJUSTE DEL CONTROL

En este apartado se prueban diferentes parámetros internos del control predictivo con restricciones cuadráticas para la obtención de las respuestas mejores de cada una de las variables controladas de nuestro sistema.

Durante el ensayo se la salida siempre de forma suficientemente grande para evitar su saturación, u entre 1 y -1, y Δu a 1 y -1 para que, por ahora esas restricciones no influyan y podamos centrarnos en modificar otros parámetros. Estos parámetros cuadráticos se utilizarán para restringir la zona de vuelo del dron y otros factores.

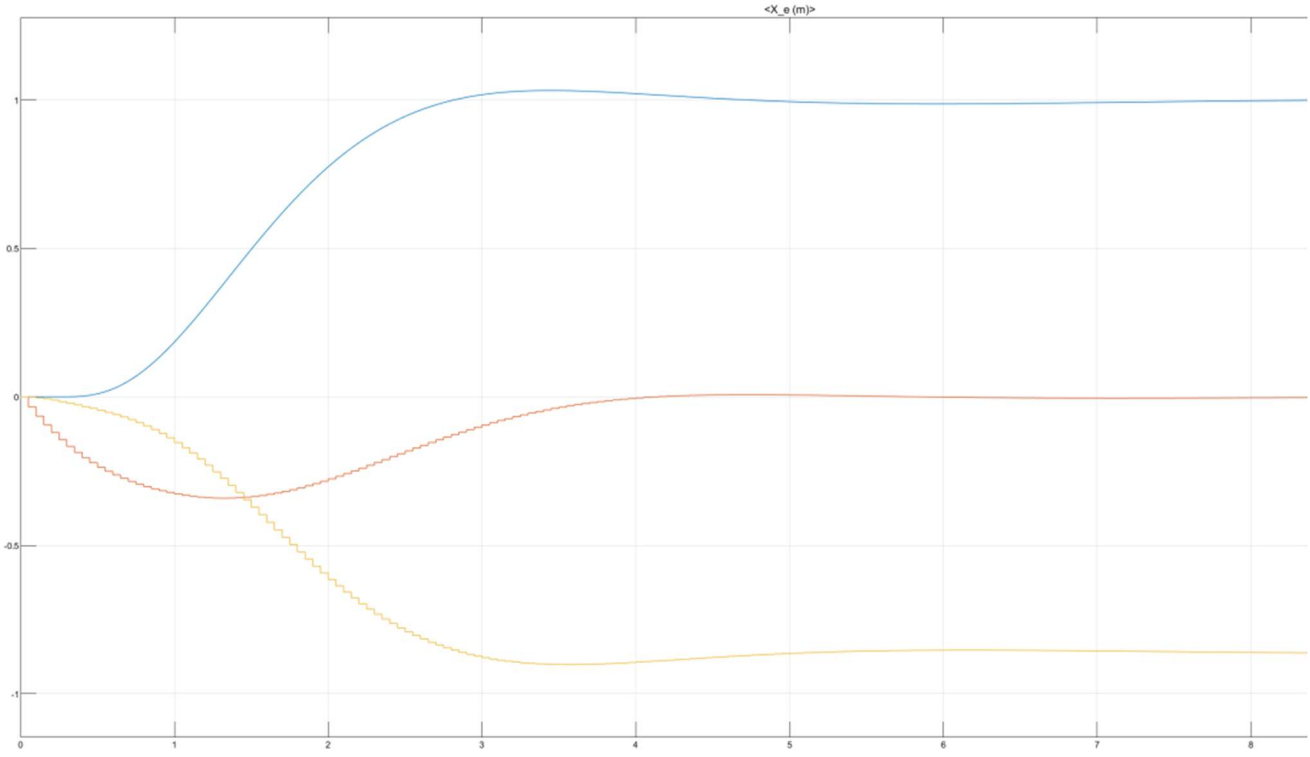
En las gráficas se representa la señal en azul y los estados estimados en otros colores, para seguir comprobando el funcionamiento de nuestros observadores.

3.4.1. Ajuste del controlador de posición de X

Empezamos el control con un horizonte de control “c” de 1 instante, horizonte de predicción “p” de 50 instantes de control, y “alfa” y “lambda” unidad. Gracias a que “c” es pequeña y “p” es alta, tenemos una respuesta estable pero algo lenta, pudiendo mejorarse su tiempo de subida aumentando “c”.

Gráfica 10. Posición en X (en azul) y sus estados estimados (en amarillo y naranja) para c=1, "alfa" y "lambda" 1,p=50. Fuente: Elaboración propia.

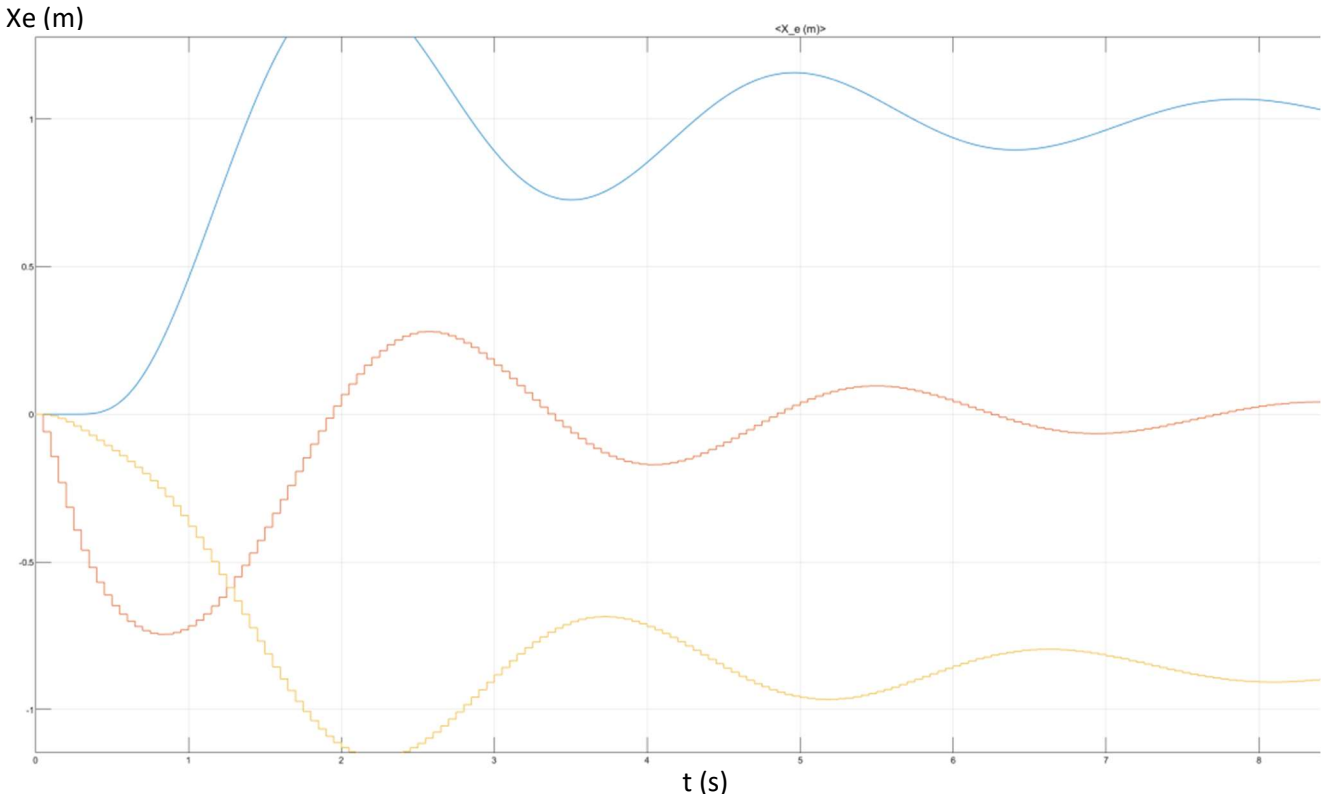
Xe (m)



t (s)

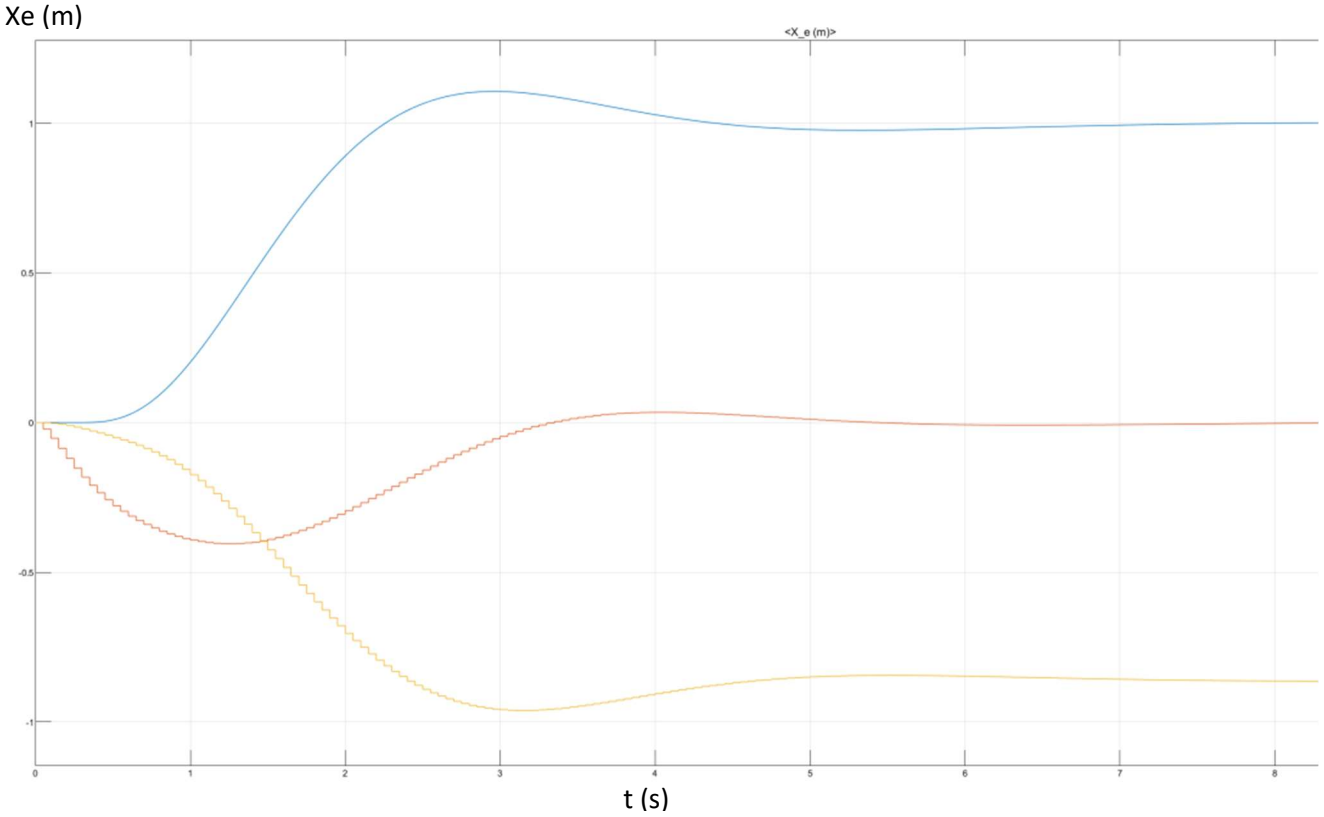
Al aumentar "c" a 2, se observa que el sistema se vuelve mucho más agresivo y oscilante, lo cual provoca severas sobreoscilaciones en la posición, nada deseables y muy agresivas. Esto puede provocar que, en un dron real, la poca robustez del control sea tal que se vuelva inestable.

Gráfica 11. X (en azul) y sus estados estimados (en amarillo y naranja)si aumentamos "c" a 2.Hace la señal casi inestable. Fuente: Elaboración propia.

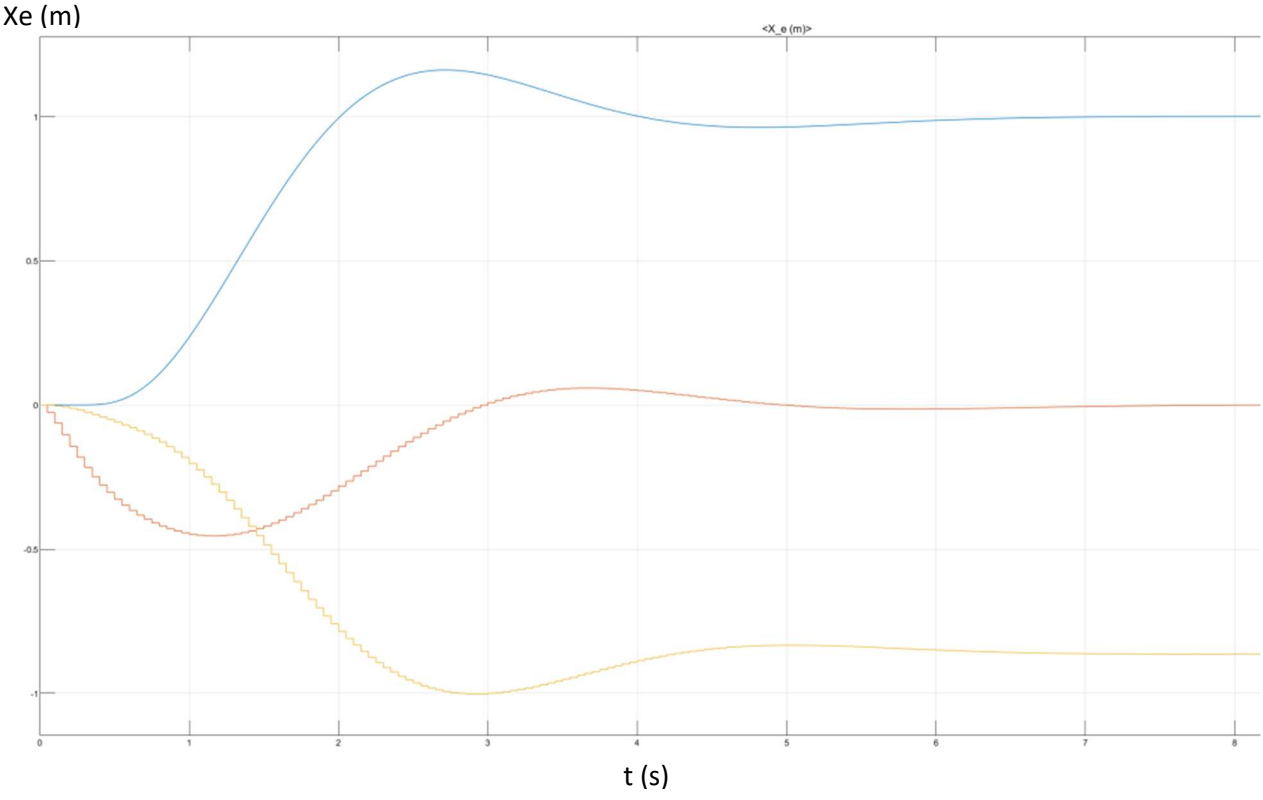


Se aumenta "lambda" a 10 para conseguir un control más robusto, y conseguimos bajar el tiempo de subida respecto del primer ensayo; alrededor de 3 segundos, a algo más de 2 segundos con una pequeña sobreoscilación. Ajustando para diferentes valores de "lambda", llegamos a un tiempo de subido de 2 segundos con ligeramente más sobreoscilación, un compromiso entre especificaciones y robustez del control aceptable para la aplicación. Un "c" mayor podría provocar tiempos de computación muy altos, porque el problema cuadrático es complejo, y un funcionamiento más agresivo; así que se conserva esta solución como aceptable para esta aplicación.

Gráfica 12. X (en azul) y sus estados estimados (en amarillo y naranja) cuando variamos "c" a 2 y "lambda" a 10.Menor mejoría del funcionamiento. Fuente: Elaboración propia.



Gráfica 13. X (en azul) y sus estados estimados (en amarillo y naranja) cuando bajamos "lambda" a 5. Fuente: Elaboración propia.



La acción de control no satura en el ensayo porque el ángulo pitch es un actuador muy generoso para avanzar un solo metro, así que no es un buen indicador de optimalidad.

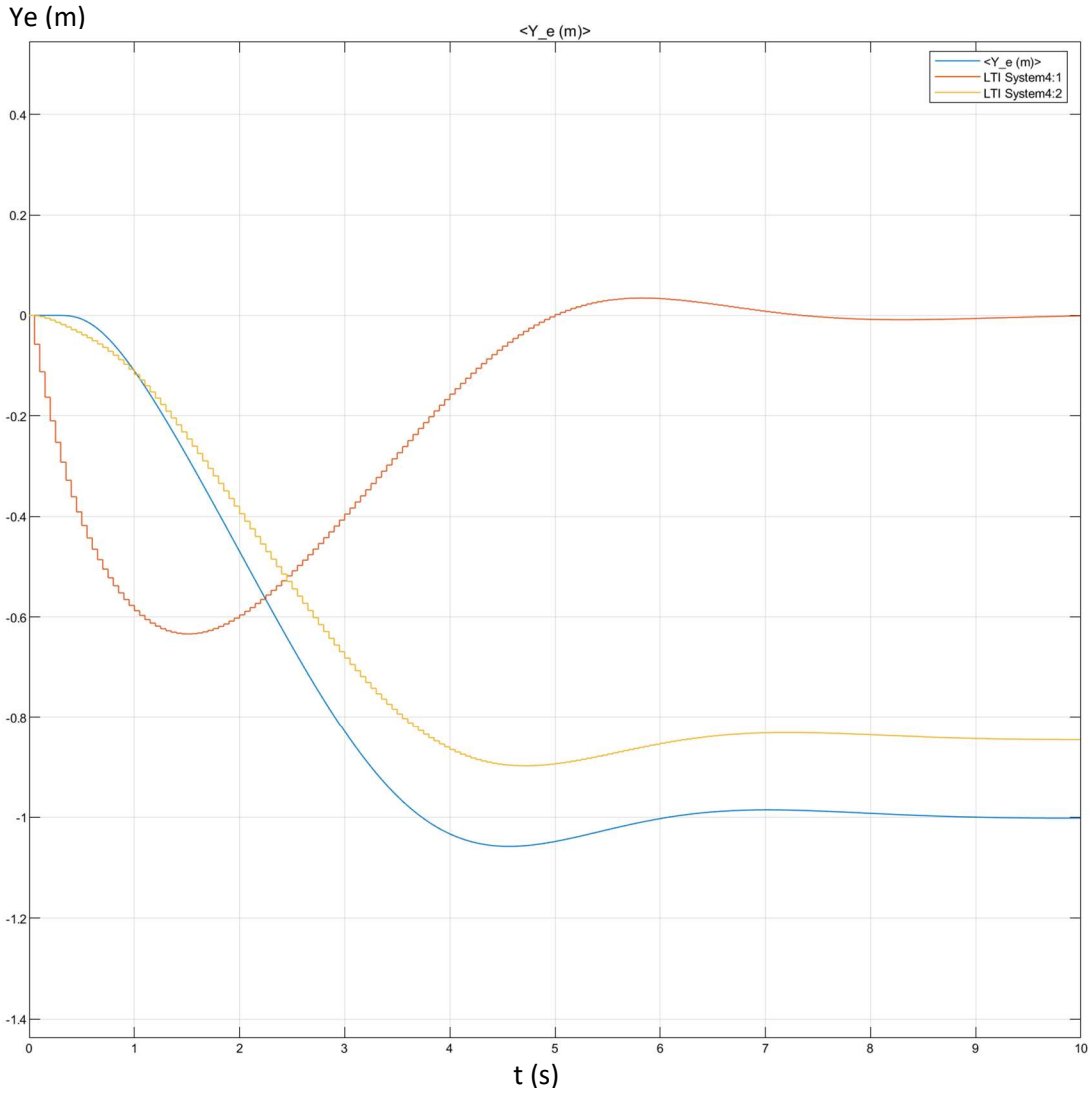
3.4.2. Ajuste del controlador de posición Y

Para la posición de y hacemos un análisis similar y obtenemos resultados muy similares a los anteriores; e intentamos una optimización similar, ya que ocurre como con el sistema anterior, donde c=2 consigue buenos tiempos de establecimiento, pero provoca sobreoscilaciones altas.

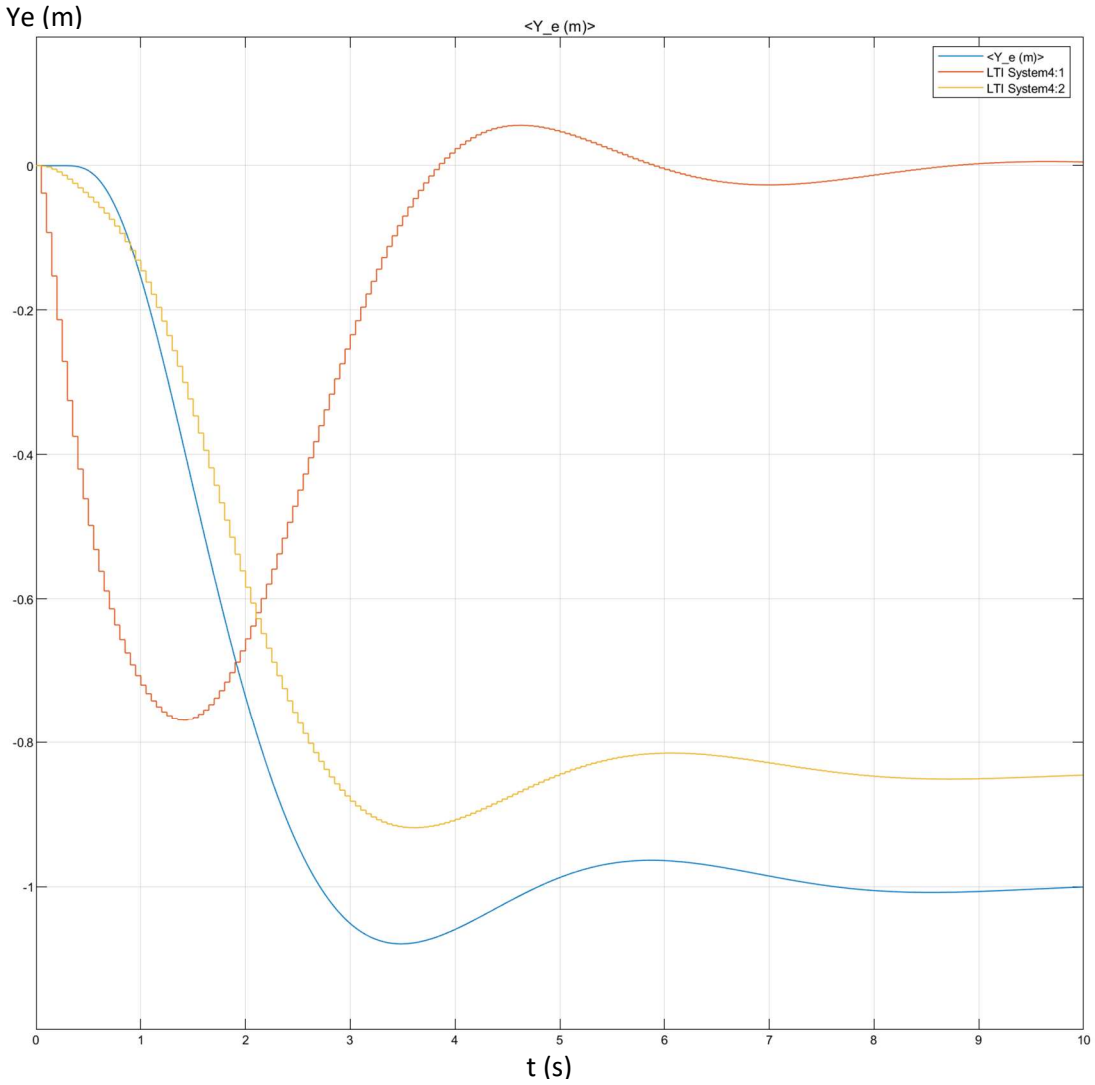
Al final nos decantamos por una "lambda" de 3 para conseguir un control 1 segundo más rápido con una oscilación aceptable.

En este caso la acción de control tiene un resultado similar a en x, donde el actuador, en este caso el ángulo roll es muy generoso y si saturara en la optimización sería tremendamente brusco.

Gráfica 14. Posición de y (en azul) y sus estados estimados (en amarillo y naranja) con "c" 1 y "alfa" y "lambda" 1, para referencia de -1. Fuente: Elaboración Propia



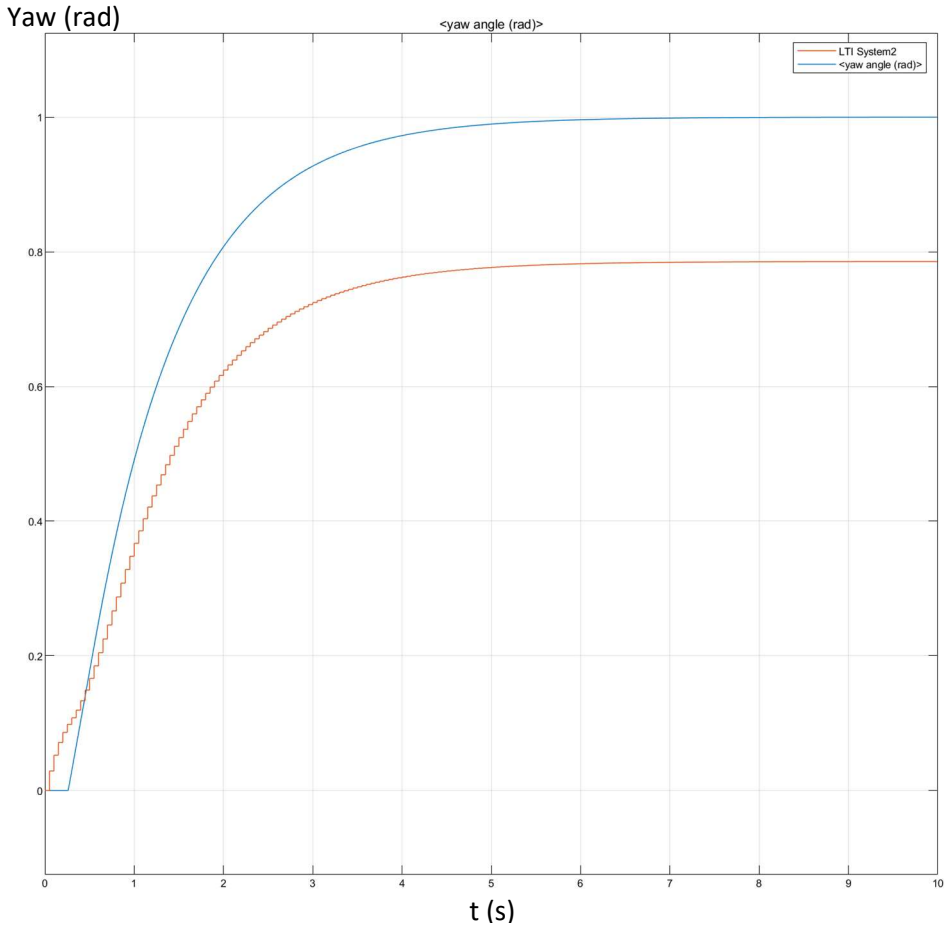
Gráfica 15. Optimización final de y (en azul) y sus estados estimados (en amarillo y naranja), con lambda=5, alfa=1 y c=2. Fuente: Elaboración Propia.



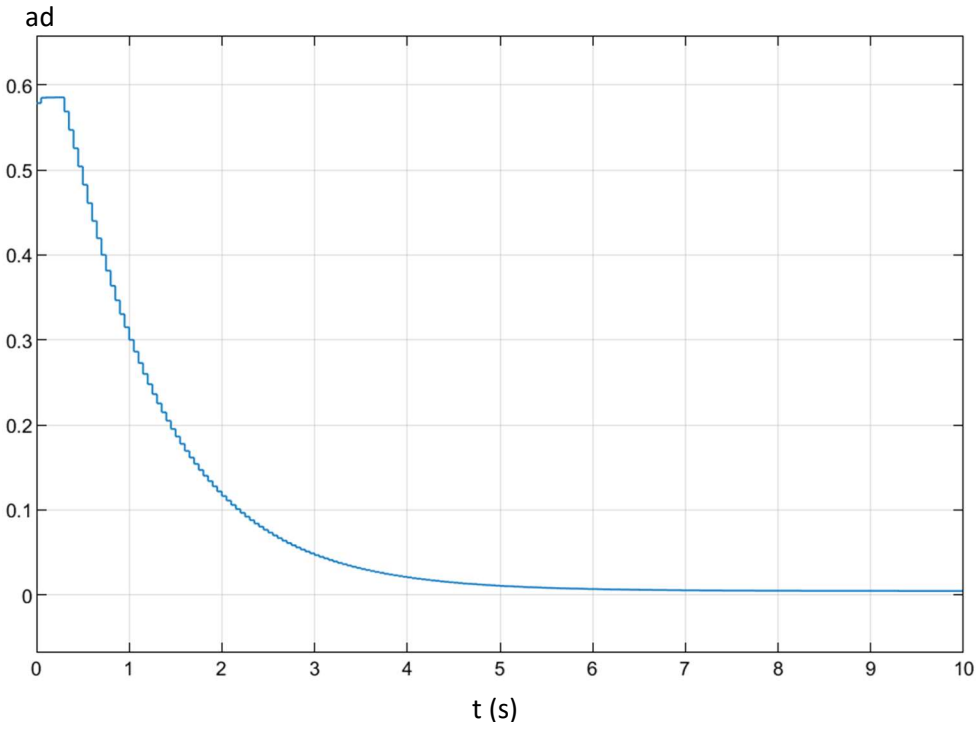
3.4.3. Ajuste del controlador del heading

En el caso del ángulo yaw, su optimización no es tan importante ya que en la trayectoria del dron el heading no es clave, aunque su control podría ser útil en el caso de que queramos hacer uso de elementos del dron como herramientas o cámaras. Aun así, se optimiza para conseguir la mayor robustez posible en el control, y tiempos de establecimiento razonables. Así que dejamos los parámetros c a 1 y "p" a 40, con alfa y "lambda" 1, para conseguir una robustez aceptable con poco tiempo de establecimiento, sin sobreoscilaciones y poco cálculo computacional. El tiempo de subida son 4 segundos, pero no tiene sentido intentar un control más rápido de algo no crítico.

Gráfica 16. Optimización final del heading(en azul) y su estado estimado (en rojo), con lambda=1, alfa=1 y c=1. Fuente: Elaboración Propia



Gráfica 17. Acción de control del ángulo yaw. Fuente: Elaboración propia

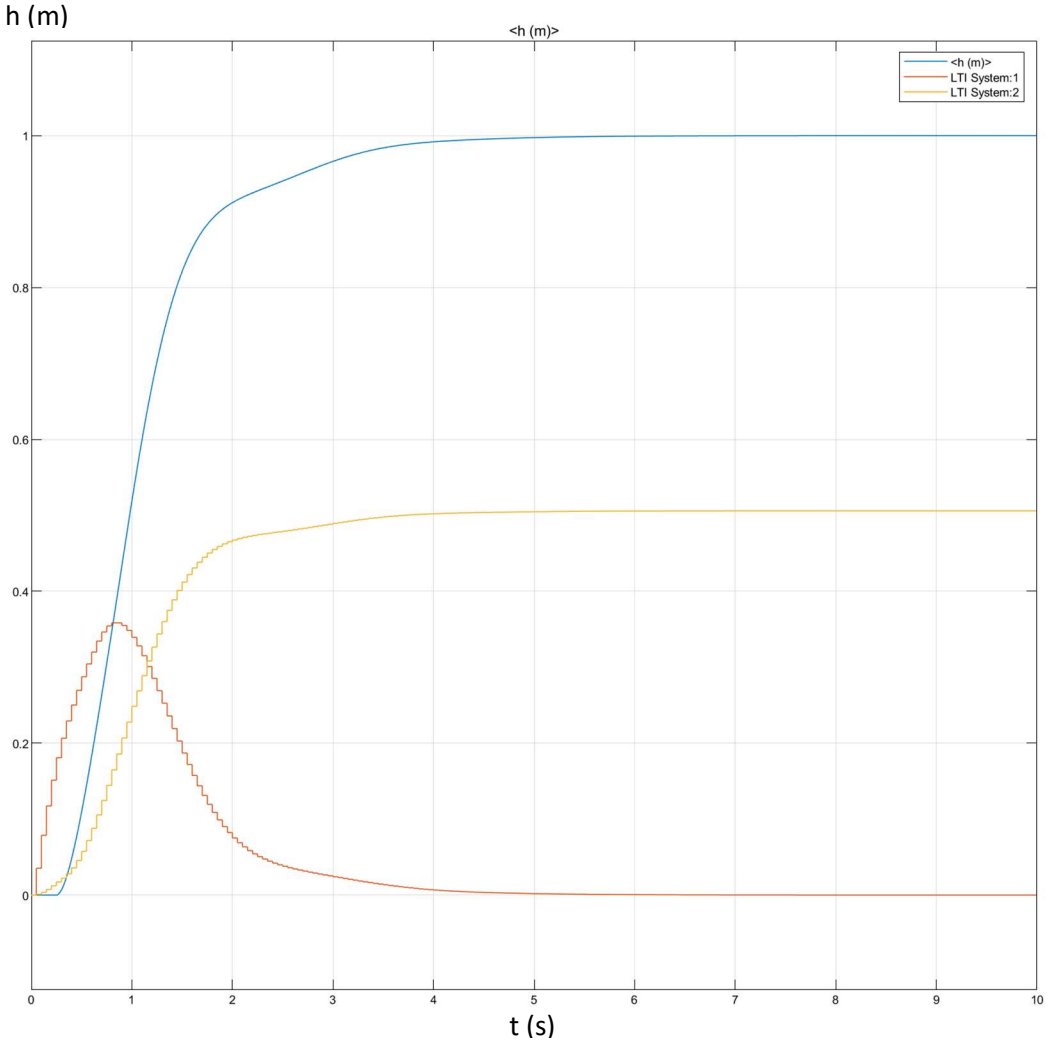


3.4.4. Ajuste del controlador de altitud

En este control sí es necesario optimizar la capacidad de respuesta del dron para cambiar la altura, así que es más crítico el tiempo de establecimiento que en el control anterior.

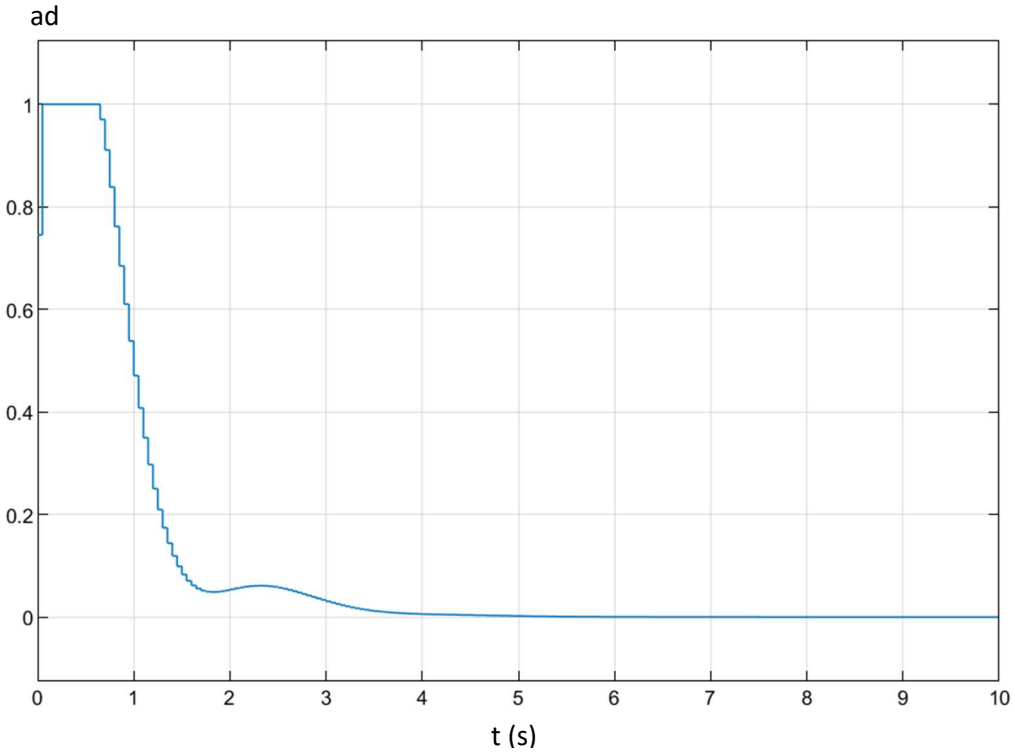
De forma parecida a el control de posición, se aumenta para mejorar el tiempo de subida con c=2, p=40, alfa=1 y lambda=1, y tenemos la siguiente respuesta:

Gráfica 18. Respuesta en altura (en azul) y sus estados estimados (en amarillo y naranja)con c=2, p=40, lambda y alfa=1. Fuente: Elaboración propia



Se observa que la llegada es bastante rápida, aunque no hay sobreoscilación de la respuesta. El problema es que la acción de control ya ha saturado, como se puede ver en la gráfica 19; esto quiere decir que hay poco margen de mejora del tiempo de subida. Aumentar "c" tiene un efecto minúsculo, y la acción de control es muy similar a la anterior. Al saturar el actuador le es imposible optimizar más la rapidez, así que no es capaz de sobreoscilar para llegar más rápido. Por tanto, este resultado es uno de los mejores posibles de obtener para esta función.

Gráfica 19. Acción de control de vertical "velocity rate"; saturando en 1.



4. GENERACIÓN DE TRAYECTORIAS Y ENTORNO GRÁFICO

4.1. GENERACIÓN DE TRAYECTORIAS

En este apartado se creará un generador de trayectorias en el que, introduciendo parámetros de entrada como el tipo de trayectoria deseada, y los puntos necesarios para que la trayectoria se complete, conseguimos que se genere una trayectoria con puntos intermedios para la llegada correcta del dron a la posición indicada.

Esta función crea las trayectorias, que se indican en una variable de tipo matriz de elementos de clases diferentes. En esta matriz se indica el tipo de trayectoria y los parámetros que se quieren introducir en el sistema. Los tres tipos de trayectorias posibles son : lineal, heading (rotación sobre sí mismo) y circular.

La trayectoria lineal es tan sencilla como su nombre indica, dando un punto final de destino, el sistema crea puntos intermedios de paso para mejorar la robustez del sistema. También dispone de la posibilidad de establecer un tiempo de espera en el punto final, si la aplicación lo requiere. Por último, el sistema también indica la precisión o tolerancia con la que se debe alcanzar cada punto antes de pasar a la siguiente de la trayectoria.

```
case "lineal"
        wp1=trayectorias{i,2};
        dist=norm(wp_ant-wp1);%Cálculo de las distancia
         deltaj=dist/ceil(dist);
       for j=deltaj:deltaj:ceil(dist)%al menos un waypoint  por cada
metro de distancia
           t=j/dist;
           waypoints{k,1}=[(1-t)*wp_ant+t*wp1 head_ant
trayectorias{i,3}*(t==1) 0.15*(t==1)+0.35*(t~=1)];%los waypoints son
creaciones intermedias lineales entre destino y origen
           k=k+1;
       end
```

Figura 12. Sección del código de funcionamiento de la creación de trayectorias lineales. Fuente: Código propio

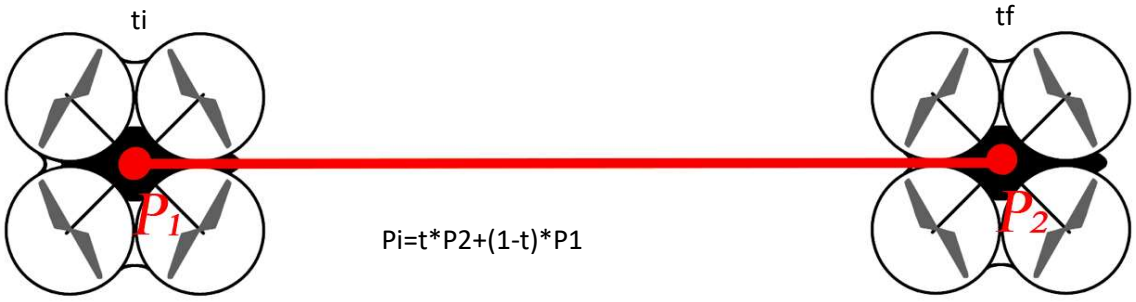


Ilustración 23. Representación de la trayectoria lineal. Fuente: Elaboración Propia.

La trayectoria de heading ejecuta una rotación sobre sí misma en el punto donde esta y vuelve a la posición original del dron para seguir volando, esto es necesario para tener un buen control de roll y pitch, ya que el heading desestabiliza el control del dron. La rotación es más interesante en el caso de que el dron tuviera una herramienta o sensor en una de sus partes, y requiriera de una posición concreta para su uso. Una vez en la rotación deseada, se espera el tiempo indicado en esa posición y vuelve a heading 0, para continuar la ruta.

```
case "heading"% muy similar a lineal pero con 1 solo parámetro
        head1=trayectorias{i,2};
        dist=abs(head_ant-head1); %Cálculo de las distancia en
radianes

        deltaj=dist/ceil(dist);%al menos un waypoint  por radian

        for j=deltaj:deltaj:ceil(dist)
           t=j/dist;
           waypoints{k,1}=[wp_ant (1-t)*head_ant+t*head1
trayectorias{i,3}*(t==1) 0.35];%El waypoint es una combiánción lineal
de 2 números, la espera en el ultimo punto.
           k=k+1;
        end
```

Figura 13. Sección del código de funcionamiento de la creación de trayectorias de heading. Fuente: Código propio.

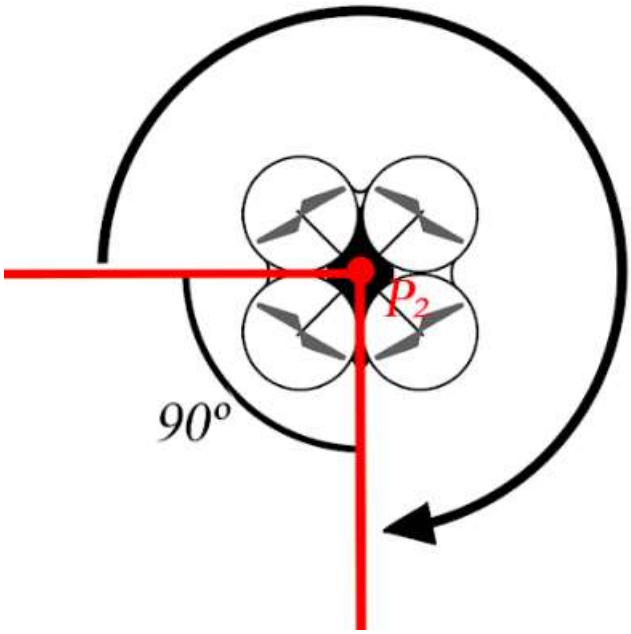


Ilustración 24. Representación gráfica de la trayectoria de cambio de heading. Fuente: Elaboración Propia.

La trayectoria circular es la más compleja de todas, en esta trayectoria indicamos 3 puntos a la misma altura por los que el dron debe pasar y el programa automáticamente calcula el centro, radio y ángulos a recorrer gracias a la función auxiliar creada para este script "par_circulo". El programa automáticamente divide la trayectoria en diferentes puntos por los que pasará el dron en su camino. Estos puntos son variables en número, a aproximadamente 2 por radian de ángulo que hace el dron.

```
case "circular"
        vec_aux=trayectorias{i,2};
        wp1=vec_aux(1:2);
        wp2=vec_aux(3:4);
        wp3=vec_aux(5:6);
        [D,F,R,alfa1,alfa2]=par_circulo(wp1,wp2,wp3);%sacamos el centro radio
y ángulo del circulo
        npuntos=(abs(alfa1)+abs(alfa2))/0.5;%el número de puntos de la
circumferencia, 1 por cada 0.5 radianes
        deltaj=(abs(alfa1-alfa2)/ceil(npuntos))*sign(alfa2);%dividimos el arco
en incremetnos deltaj
         for j=alfa1:deltaj:alfa2
            waypoints{k,1}=[R*cos(j)+D R*sin(j)+F h_ant head_ant
trayectorias{i,3} 0.15*(j==alfa2)+0.35*(j~=alfa2)]; %Creación del arco en
waypoints
            k=k+1;
         end
```

Figura 14. Sección del código de funcionamiento de la determinación de los parámetros del círculo. Fuente: Código propio.

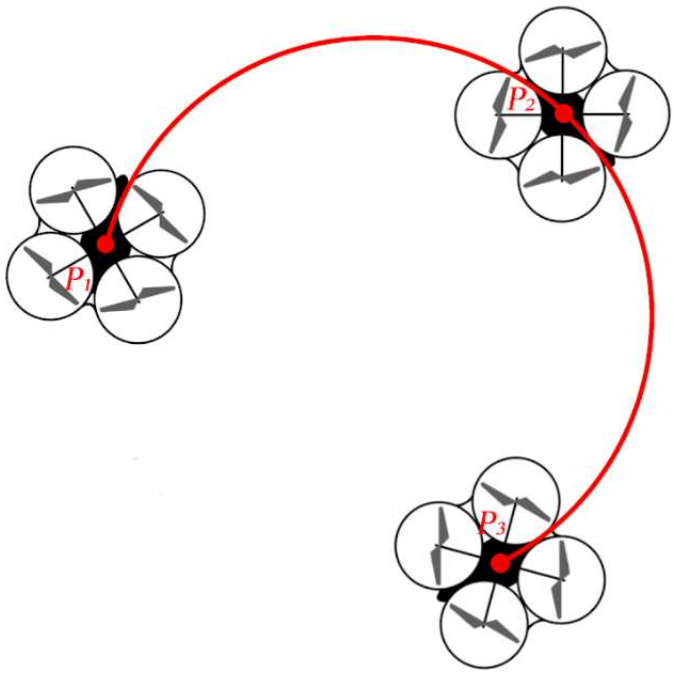


Ilustración 25. Representación Gráfica de la trayectoria circular. Fuente: Elaboración Propia.

Cada uno de los casos genera un número de puntos de paso que después llegan a un gestor de trayectorias; donde se pasa como referencia el primer punto; y si se cumple un cierto criterio de proximidad y ha pasado un tiempo indicado por cada trayectoria, se pasa al siguiente punto, excepto si es el último punto de la matriz. En la siguiente imagen se puede observar el bloque de generación de trayectorias en Simulink y sus entradas y salidas

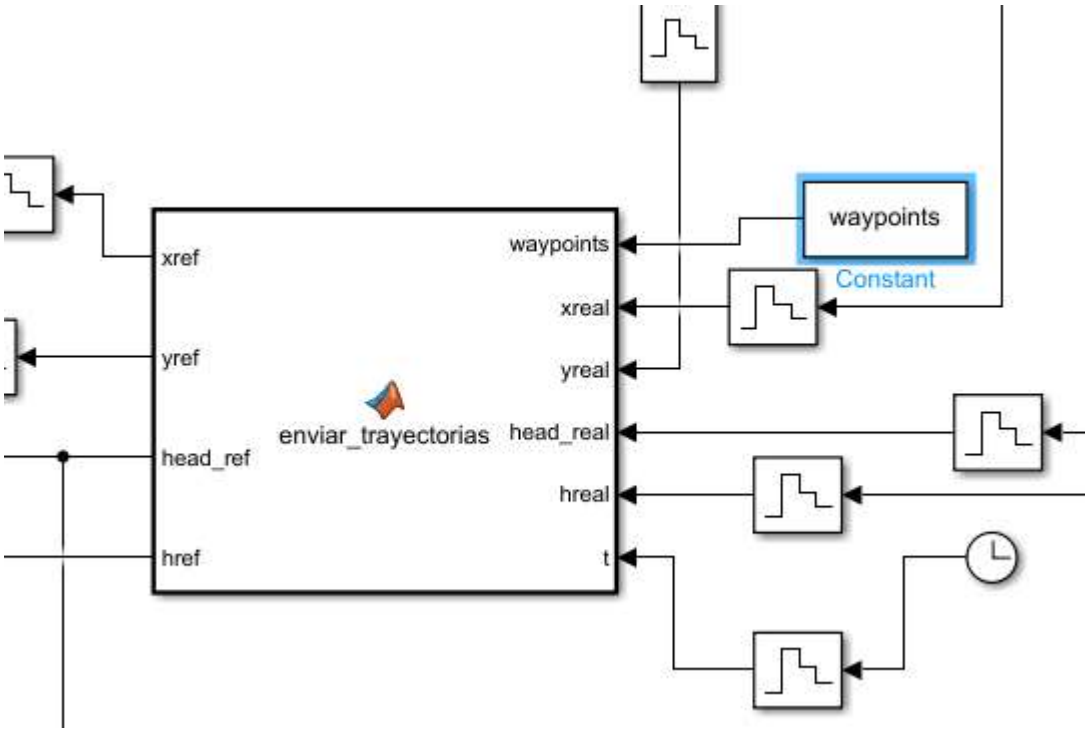


Ilustración 26. Enviar trayectorias en el diagrama Simulink, con sus entradas y salidas a la función. Fuente: Elaboración propia

El criterio de paso al siguiente punto de la trayectoria es variable entre 0.15 y 0.35 según sea punto de paso o final, y de 4 grados en el caso del heading. Cuando esto ocurre entre el punto actual y la posición real, pasamos al siguiente punto, a no ser que debamos mantener la posición durante un determinado tiempo, en ese caso se empieza a contar y se pasará de punto cuando el reloj indique que ha pasado el incremento de tiempo requerido.

```
if ( norm([xreal yreal] - waypoints( wpointcount,1:2)) < waypoints(
wpointcount,6) && ...
        norm(hreal - waypoints( wpointcount,3)) < 0.45 && norm(head_real -
waypoints(wpointcount,4)) < 4*pi/180)%Comrpobamos la distancia al waypoint

    if(wpointcount ~= nPoints)

        if (t0 ==0)
            t0 = t;
        else
            if (t - t0 > waypoints( wpointcount,5)) %Comprobamos el tiempo
de espera

                if(wpointcount < nPoints)
                   wpointcount = wpointcount+ 1;
                   t0 = 0;
                end
```

Figura 15. Criterio de llegada al punto según la función de gestión de las trayectorias. Fuente: Código Propio

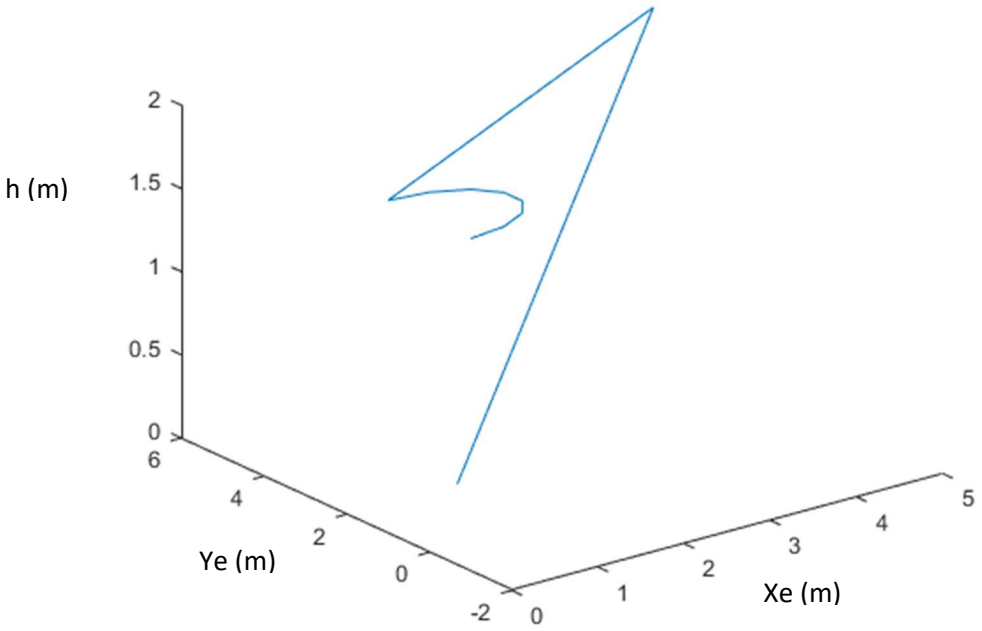
4.2. RESULTADOS INICIALES

Para la comprobación del generador y gestor de trayectorias, se crea una secuencia de trayectorias de diferente tipo y longitud; y se simula el controlador para las referencias que va introduciendo el sistema de forma gradual, y se comparan los resultados de la trayectoria original y la realizada

```
trayectorias={"lineal" [5 5 2] 0; %indicamos la x,y y altura y el tiempo de espera
    "heading" 2 1%indicamos el nuevo heading, y el tiempo de espera
    "circular" [ 0 1  1 0  0 -1] 0};%indicamos 3 puntos a la misma altura y el tiempo de espera
```

Ilustración 27. Trayectoria declarada en el programa con la sintaxis correspondiente. Fuente: Elaboración propia

Gráfica 20. Trayectoria generada representada gráficamente. Fuente: Elaboración propia



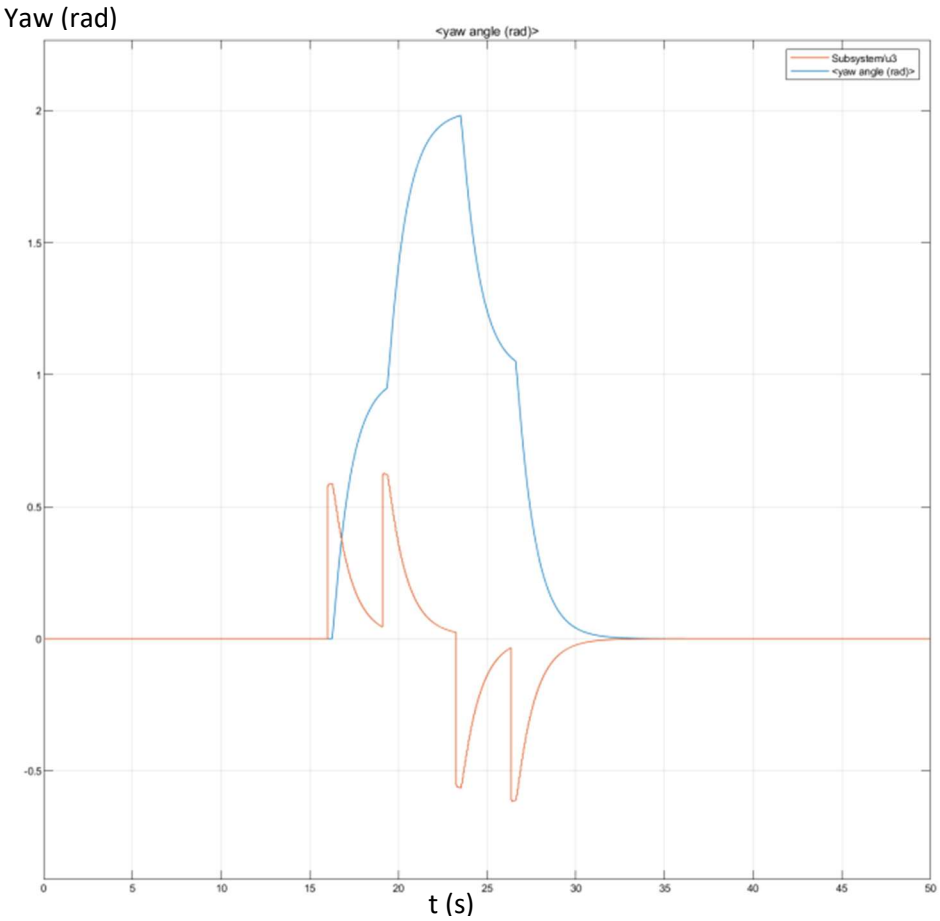
La generación de trayectorias inicial no se hace al mismo tiempo que el control, sino antes. Por tanto, el programa genera automáticamente una matriz a partir de la trayectoria indicada donde se encuentran los siguientes puntos de paso. En la matriz tenemos las cuatro salidas del sistema a las que queremos llegar (3 dimensiones más heading) y el tiempo de espera en ese punto:

	1	2	3	4	5
1	0.6250	0.6250	0.2500	0	0
2	1.2500	1.2500	0.5000	0	0
3	1.8750	1.8750	0.7500	0	0
4	2.5000	2.5000	1	0	0
5	3.1250	3.1250	1.2500	0	0
6	3.7500	3.7500	1.5000	0	0
7	4.3750	4.3750	1.7500	0	0
8	5	5	2	0	0
9	5	5	2	1	0
10	5	5	2	2	1
11	5	5	2	1	0
12	5	5	2	0	0
13	6.1232e-17	1	2	0	0
14	0.4339	0.9010	2	0	0
15	0.7818	0.6235	2	0	0
16	0.9749	0.2225	2	0	0
17	0.9749	-0.2225	2	0	0
18	0.7818	-0.6235	2	0	0
19	0.4339	-0.9010	2	0	0
20	6.1232e-17	-1	2	0	0

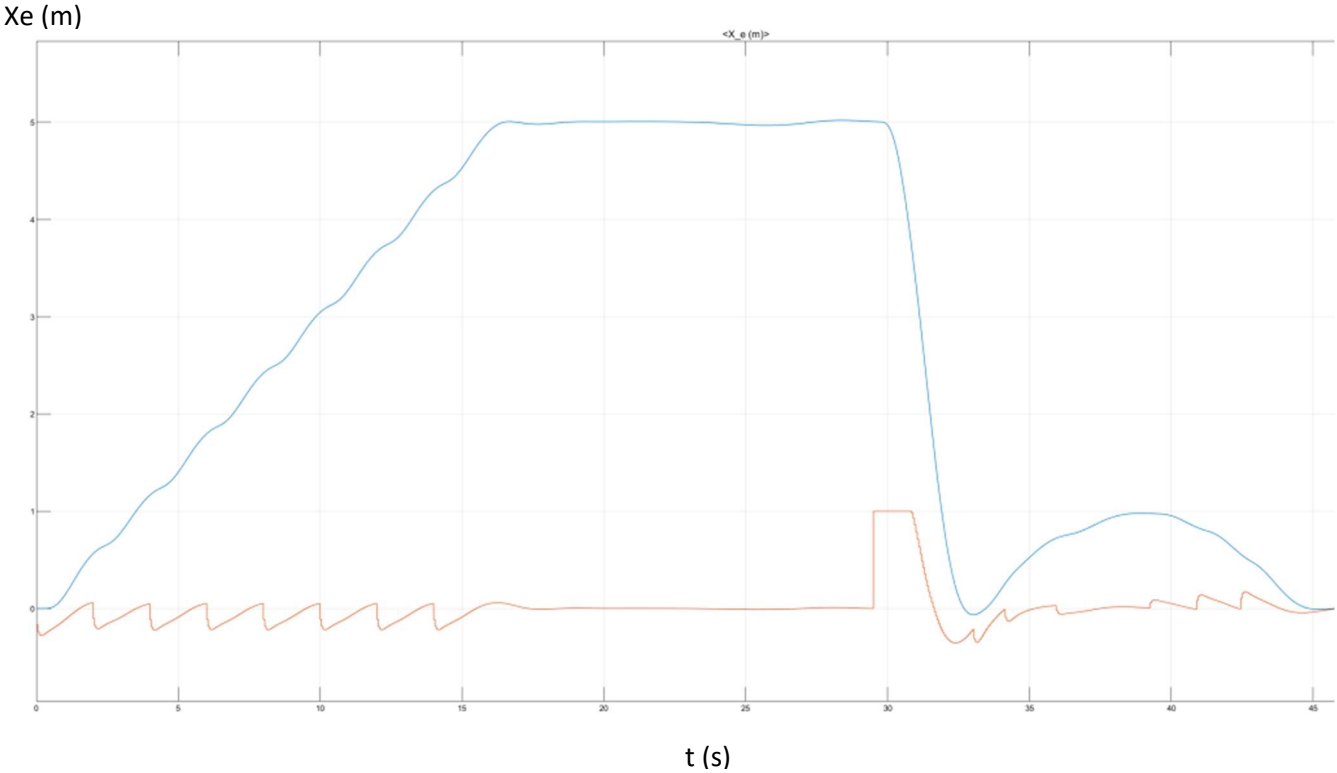
Ilustración 28. Trayectoria generada representada por las coordenadas de los puntos y el tiempo de espera, sin mostrar la precisión. Fuente: Elaboración propia

Si tan solo observamos la salida de cada una de las señales de forma independiente y no representadas en una trayectoria; se puede observar mejor como ha funcionado el sistema de control de cada una de las señales y el movimiento en heading, que no se ve en una trayectoria en el espacio.

Gráfica 21. Trayectoria realizada, vista desde la señal del yaw(en azul) y su acción de control(en rojo)

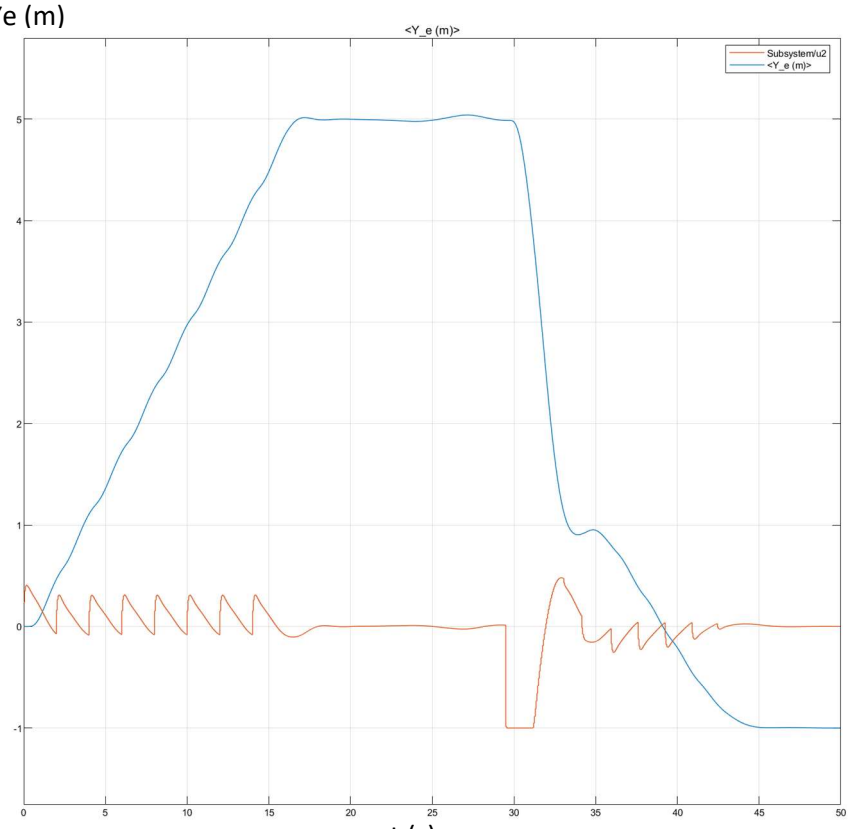


Gráfica 22. Trayectoria realizada, vista la señal X (en azul) y su acción de control(en rojo)



Gráfica 23. Trayectoria realizada, vista la señal Y(en azul) y su acción de control (en rojo)

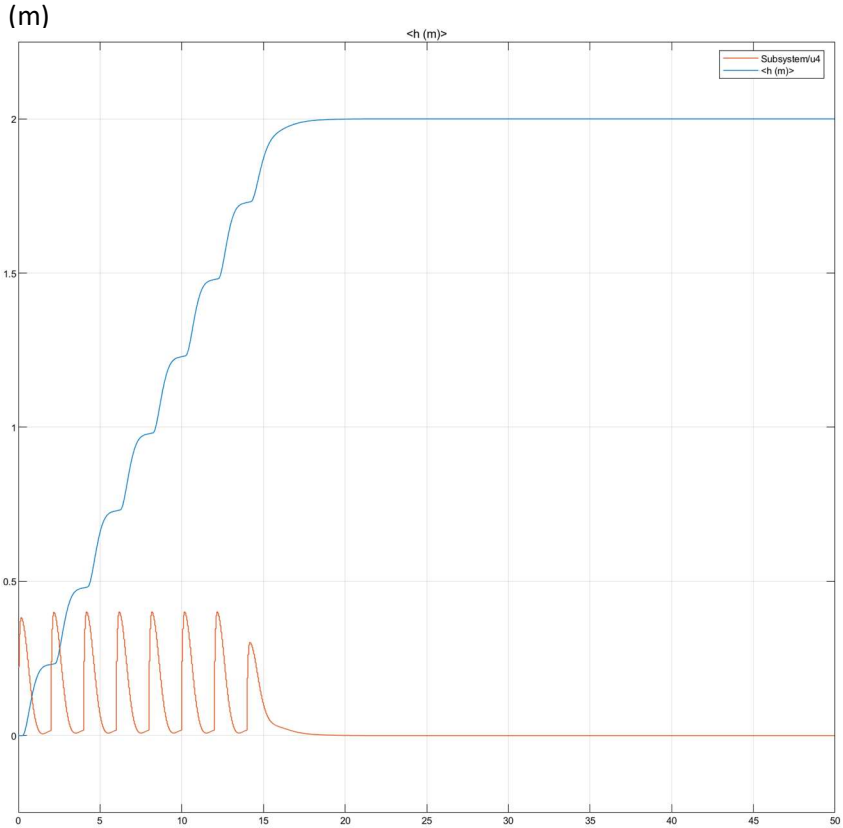
Ye (m)



t (s)

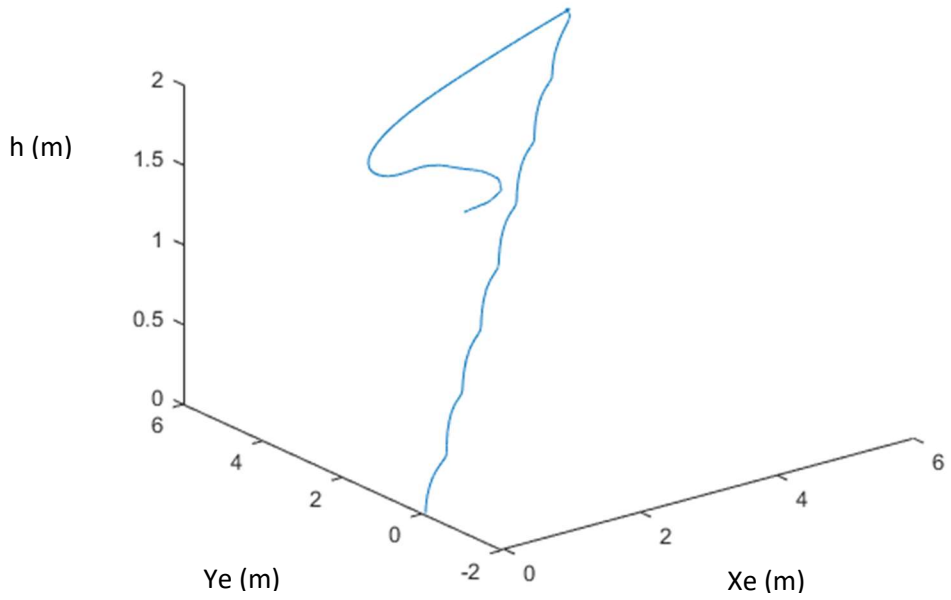
Gráfica 24. Trayectoria realizada, vista la señal de la altura(en azul) y su acción de control(en rojo)

h (m)



t (s)

Gráfica 25. Trayectoria realizada por el dron en simulación. Claramente experimenta cambios demasiado bruscos de referencia. Fuente: Elaboración propia



4.3. MEJORA DEL GENERADOR DE TRAYECTORIAS

El generador de trayectorias que hemos utilizado da las referencias de los puntos de paso antes del control del dron y sin tener en cuenta un tiempo de llegada determinado; aunque es una forma válida de dar referencias, para esta clase de trayectorias la falta del control de velocidad en tiempo real provoca brusquedades y cambios de trayectoria demasiado fuertes para el dron; así que probaremos de mejorar este sistema con un **interpolador en tiempo real con control del tiempo de llegada**. Esta modificación se parece más al algoritmo original utilizado en la bibliografía(3) que el anterior, ya que genera los puntos intermedios en tiempo real.

Este interpolador es la función “enviar_trayectorias2”. Tenemos una función donde las trayectorias se interpolan entre el punto inicial y final en tiempo real atendiendo al tipo de trayectoria, el tiempo de espera; y una capacidad única de este interpolador es el **control de la velocidad** a la que las referencias llegan, es decir; del tiempo para realizar la trayectoria.

En este nuevo sistema las trayectorias se declaran de forma muy parecida, pero se añade el tiempo de maniobra y las trayectorias circulares pasan a declararse **dando el centro y el ángulo con signo;** siendo positivo el sentido antihorario o levógiro. El tiempo de espera se mantiene como en el anterior generador de trayectorias.

Una vez se tienen las trayectorias declaradas, la estructura es muy parecida excepto por que los puntos se generan en tiempo real basándose en el tiempo en el que se quiere llegar; aunque si se da un valor muy pequeño de tiempo **se prioriza la llegada a hacer una trayectoria perfecta**, dando el punto final de la trayectoria directamente al controlador y omitiendo el resto de los puntos de paso.

```
if (t < tfinal)
            tn= (t-tinicial)/(tfinal-tinicial);
            r=[(1-tn)*wp_ant+tn*wp1 head_ant];%los waypoints son creaciones
intermedias lineales entre destino y origen

         else
            r=[wp1 head_ant];
             if (( norm([xreal yreal] - wp1(1:2)) < 0.1 && norm(hreal - wp1(3)) <
0.1 )) && (traycount < ntray) && (llegada==0)
                 tesp=t+trayectorias{traycount,3};
                 llegada=1;
             end
             if (t >= tesp )  && (llegada==1)
              traycount=traycount+1;
              wp_ant=wp1;
              h_ant=wp1(3);
              flag=0;
              tesp=0;
              llegada=0;
```

Figura 16. Algoritmo de la trayectoria lineal; que es interpolada mientras no se supere al tiempo de llegada. Fuente: Código propio

Como se puede ver, la interpolación de los puntos en una trayectoria lineal se interrumpe si el tiempo de llegada se agota; entonces el sistema intentará llegar lo más rápido posible al punto final, dando la última referencia al controlador. En el caso de trayectorias circulares, se crean puntos espaciados un grado sexagesimal entre el punto inicial y final. Una vez determinado el punto final que indican el centro y ángulo, se crea la lista de puntos intermedios y se elige puntos gradualmente más avanzados en la trayectoria a medida que avanza el tiempo de llegada.

```
vec_aux=trayectorias{traycount,2};
         centro=vec_aux(1:2);
         angulo=vec_aux(3);
         angulor=atan2(wp_ant(2)-centro(2),wp_ant(1)-centro(1));
         Radio=norm(wp_ant(1:2)-centro);
         if angulor >= 0
         angulor=angulor*180/pi;
         else
             angulor=angulor*180/pi*-1+180;
         end
         wp1=[Radio*cosd(angulor+angulo)+centro(1)
Radio*sind(angulor+angulo)+centro(2) h_ant];
         if (flag==0)
             flag=1;
             deltat=trayectorias{traycount,4};
             tinicial=t;
             tfinal=tinicial+deltat;
             tesp=0;
         end
         if (t < tfinal)
            ang1=angulor:sign(angulo):angulor+angulo;
            wp=[Radio*cosd(ang1)'+centro(1),
Radio*sind(ang1)'+centro(2),h_ant*ones(size(ang1))'];
            tn= (t-tinicial)/(tfinal-tinicial);
            tn=ceil(tn*size(wp,1)+0.01*(tn==0));
            r=[wp(tn,1) wp(tn,2) wp(tn,3) head_ant];
```

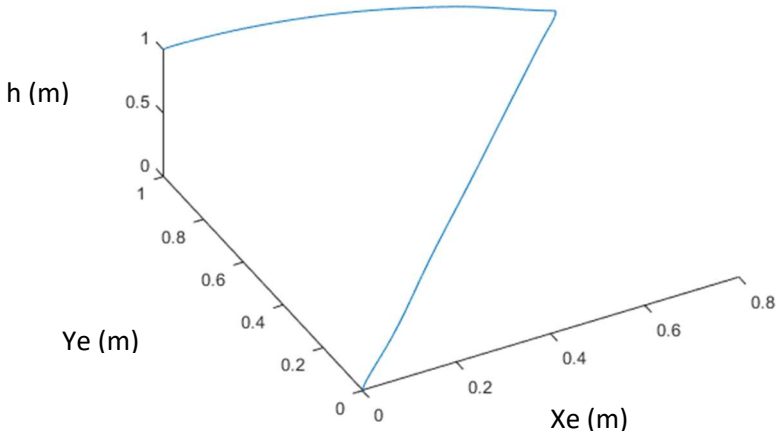
Figura 17. Algoritmo de la trayectoria circular, donde se determinan el punto de llegada, se obtienen los puntos y se envían al controlador. Fuente: Código propio

Si vemos las gráficas conseguidas gracias al nuevo sistema, podemos observar como el funcionamiento es mucho más suave y además con un control de tiempo mucho claro para el usuario, con una interpolación casi inapreciable en el funcionamiento de las trayectorias. Aun así, si se dan tiempo muy ajustados puede provocar una pequeña brusquedad a la llegada que hace que la trayectoria no sea correcta del todo.

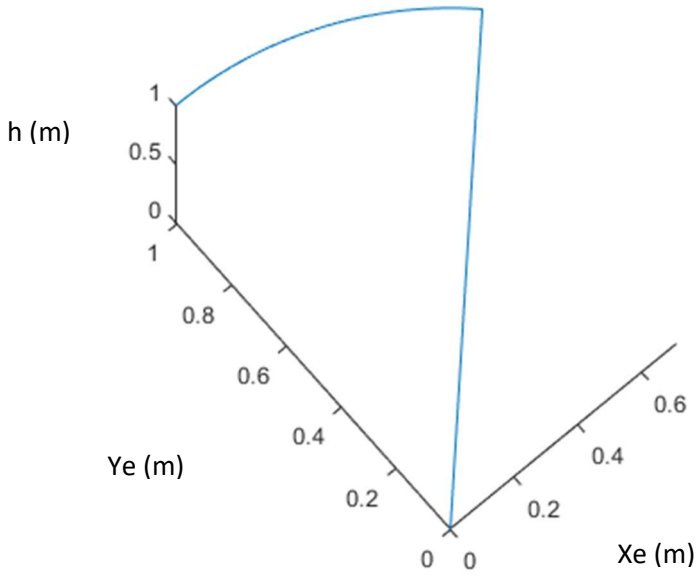
```
trayectorias={"lineal" [0.71 0.71 1] 1 10; %indicamos la x,y y altura , el tiempo de espera y el tiempo de llegada
     "heading" 1 1 10;%indicamos el nuevo heading, , el tiempo de espera y el tiempo de llegada
     "heading" 0 1 10;
       "circular" [0 0 45] 0 15};%indicamos el centro , el ángulo , el tiempo de espera y el tiempo de llegada
```

Ilustración 29. Nueva trayectoria indicada, con la sintaxis correspondiente. Fuente: Elaboración propia

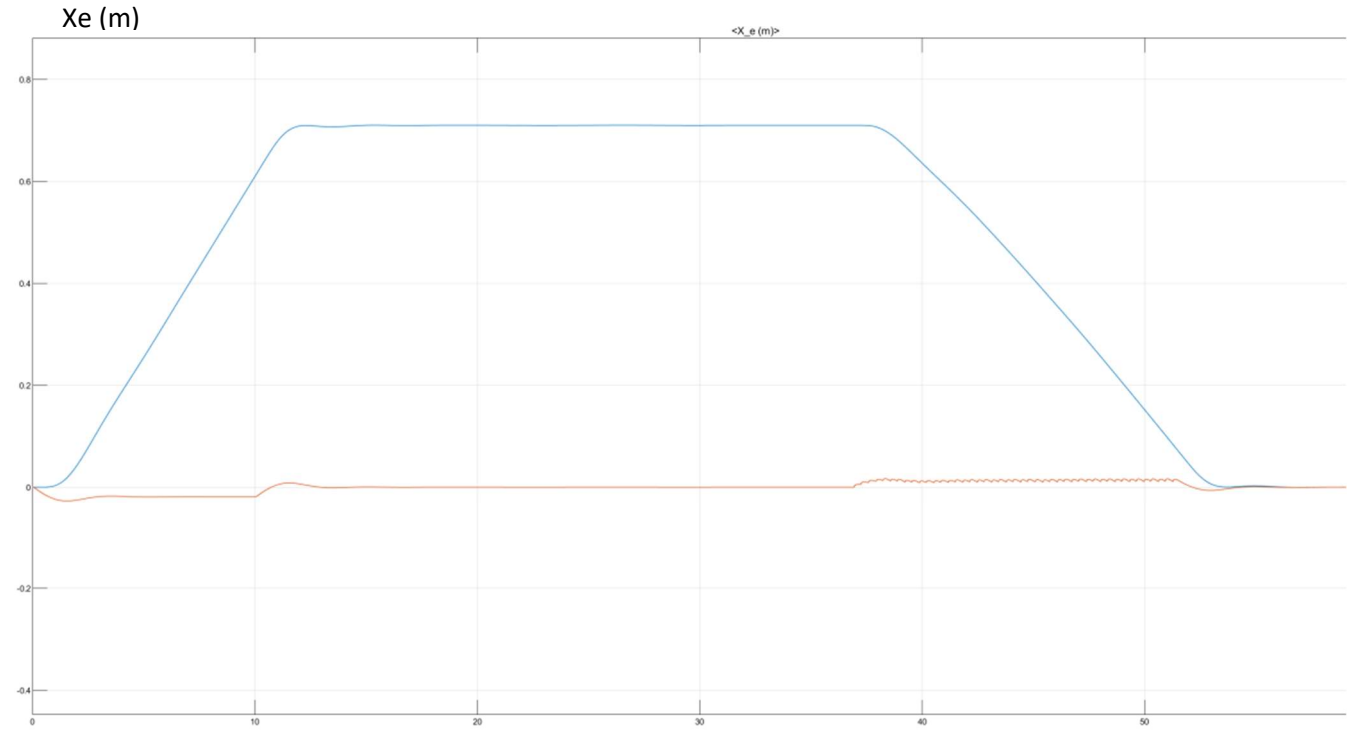
Gráfica 26. Trayectoria realizada por el dron. Nótese la suavidad de la trayectoria Fuente: Elaboración propia



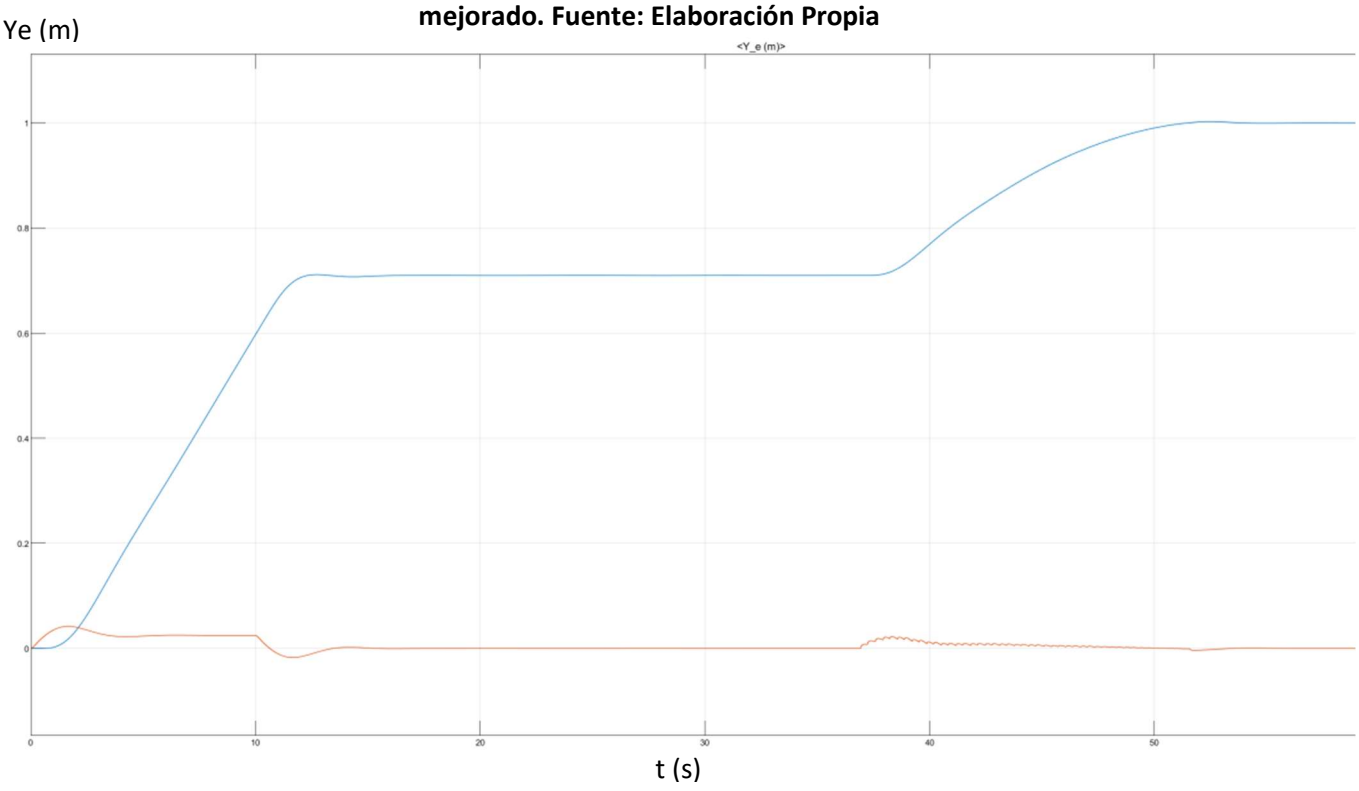
Gráfica 27. Referencias en X Y, h representadas en 3D. Fuente: Elaboración propia



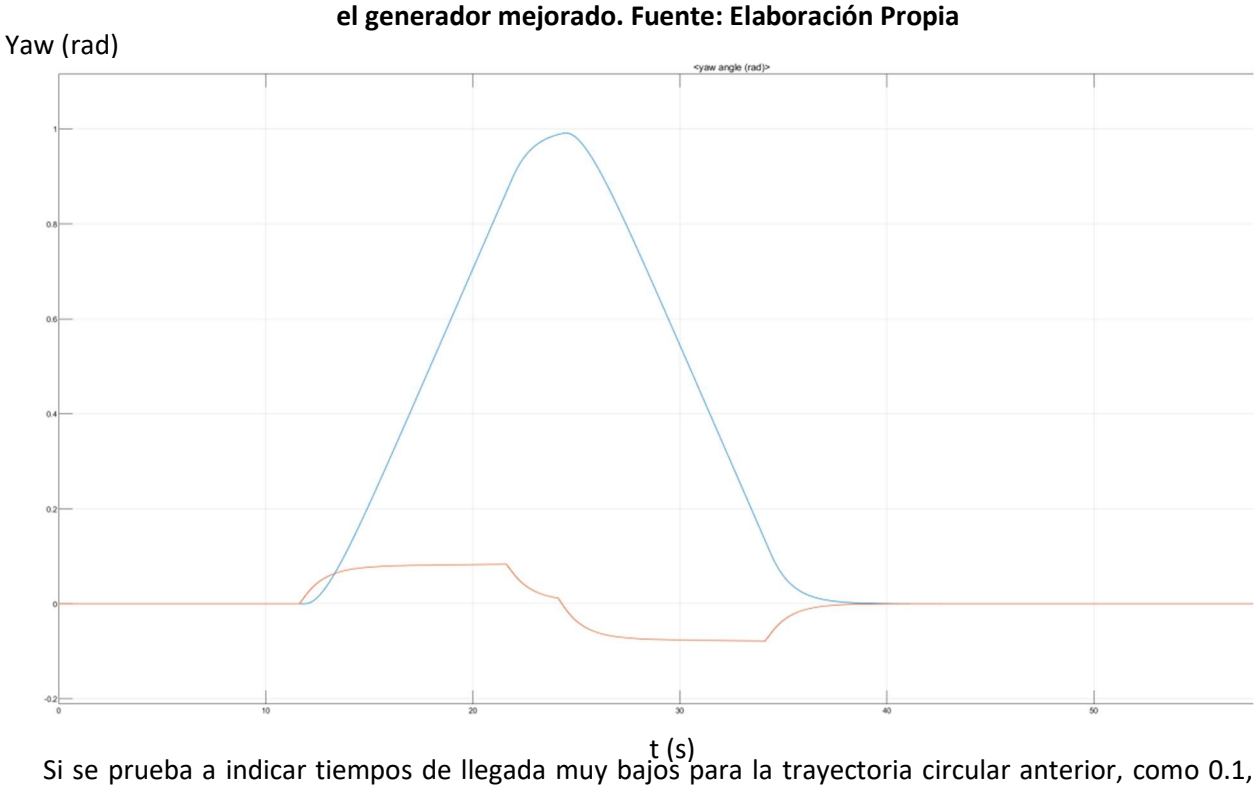
Gráfica 28. Señal de X (en azul) y su acción de control (en rojo) al realizar la trayectoria con el generador mejorado. Fuente: Elaboración Propia



Gráfica 29. Señal de Y(en azul)y su acción de control (en rojo) al realizar la trayectoria con el generador mejorado. Fuente: Elaboración Propia

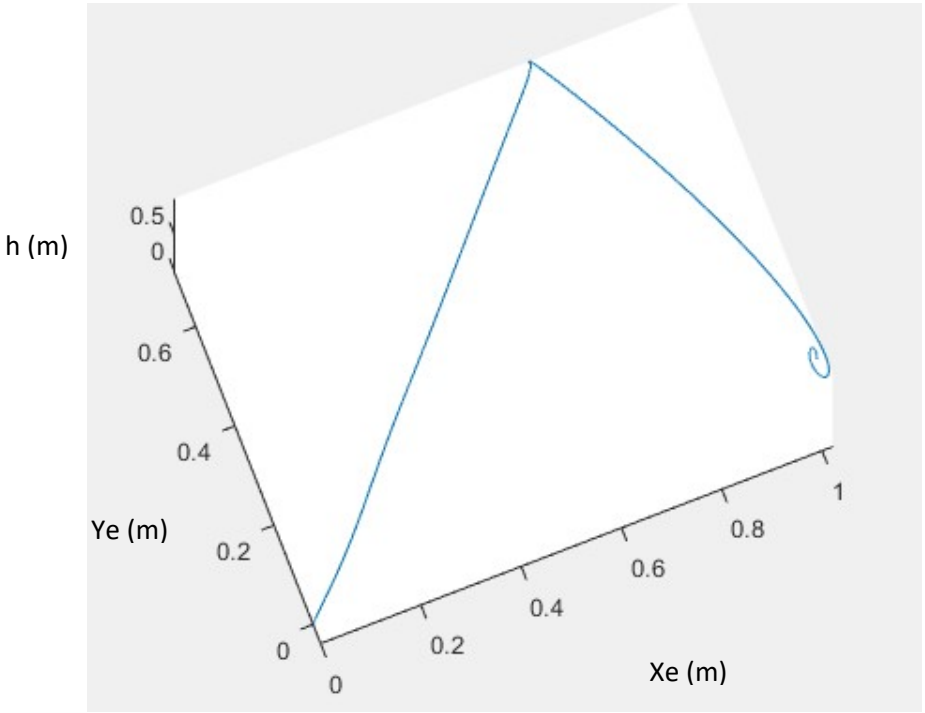


Gráfica 30. Señal de heading (en azul) y su acción de control (en rojo) al realizar la trayectoria con el generador mejorado. Fuente: Elaboración Propia

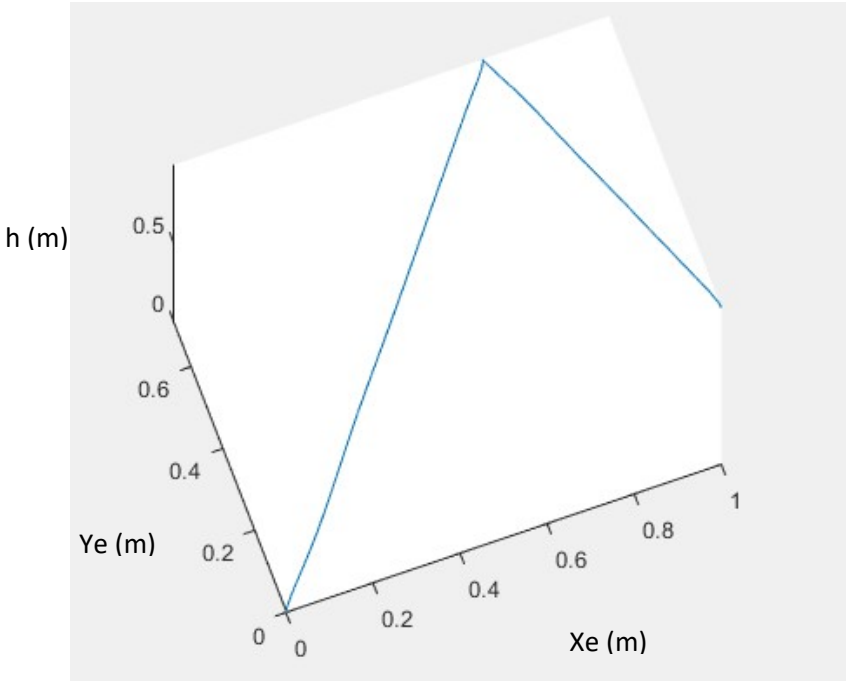


Si se prueba a indicar tiempos de llegada muy bajos para la trayectoria circular anterior, como 0.1, podemos observar que la trayectoria es deformada, haciéndose similar a una diagonal; y provocando una sobreoscilación final que es la espiral cerca del punto de llegada que se puede observar en la gráfica siguiente.

Gráfica 31. Misma trayectoria realizada con un tiempo de llegada demasiado bajo para el dron. Fuente: Elaboración Propia



Gráfica 32. Trayectoria sustituyendo el comando de trayectoria circular por uno lineal con el mismo punto final. Fuente: Elaboración Propia



4.4. SIMULACIÓN UTILIZANDO LA INTERFAZ GRÁFICA FLIGHTGEAR

4.4.1. Envío de parámetros

Flightgear es un programa pensado principalmente para la simulación realista del pilotaje de aeronaves recreativas en entornos reales de vuelo, como aeropuertos y su geografía alrededor. El programa está diseñado como programa de código abierto gratuito; por lo que los usuarios pueden modelar sus propios escenarios y terrenos; o crear sus propias aeronaves para pilotar o controlar de forma automática. En el caso de este trabajo, lo único que se utilizará es una visualización que recibirá datos directamente de Simulink sobre el estado de la aeronave; los representará en el escenario y la aeronave creadas en el programa para esta simulación. De esta forma se observa el estado de la aeronave de forma sencilla y visual.

En este apartado; se utilizará el programa de simulación de aeronaves Flightgear para la creación de una interfaz gráfica donde se observe la trayectoria de vuelo del dron; y se pueda atestiguar de un solo golpe de vista el funcionamiento del dron sin necesidad de interpretar las gráficas de resultados de Simulink.

Para el envío del estado del dron al programa de Flightgear; se puede utilizar el bloque de Simulink "FlightGearPreconfigured 6DoF Animation", que permite la conexión simplificada entre estos dos programas para la visualización de sistemas. El bloque requiere de las coordenadas de la aeronave en latitud y longitud, además de la altitud y los 3 ángulos de rotación de la aeronave: roll, pitch y yaw. La única configuración necesaria es la versión de Flightgear, el puerto de entrada de comunicaciones, la dirección IP de envío y el período de muestreo de la imagen.

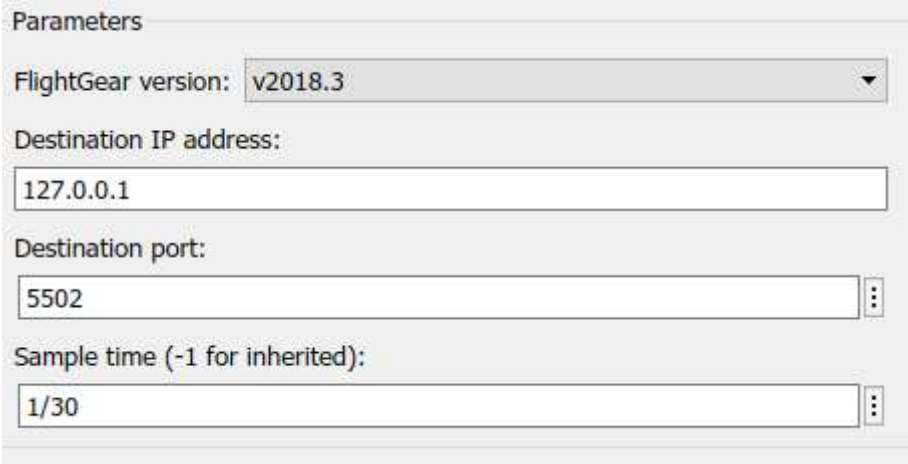


Ilustración 30. Opciones posibles en el bloque "FlightGear Preconfigured 6DoF Animation". Fuente: Elaboración Propia.

Es necesaria una configuración adicional a la hora de simular el modelo del dron para evitar problemas gráficos; que consisten en una preconfiguración y en la eliminación de ciertas mejoras gráficas al terreno y ambiente que provocan errores de visualización y cierran forzosamente el programa. Esta configuración es el siguiente código; que se escribe en la ventana de configuración de Flightgear:

--fdm=network,localhost,5501,5502,5503 --fog-fastest --disable-clouds --start-date-lat=2004:06:01:09:00:00 --disable-sound --enable-freeze --altitude=0 --heading=0 --offset-distance=0 --offset-azimuth=0 --enable-terrasync --prop:/sim/rendering/shaders/quality-level=0

4.4.2. Transformación de coordenadas planas a latitud y longitud

Primeramente, es necesario crear una simulación preconfigurada para que la simulación siempre utilice los mismos parámetros. En este caso, se utiliza el bloque "Generate Run script", encontrado en las herramientas de Matlab de simulación de aeronaves; para configurar el sistema con un script preconfigurado que crea Matlab de forma simplificada al pasar diferentes parámetros al programa.

En esta preconfiguración elegimos el aeropuerto Ellis Agricultural Field, un aeropuerto no comercial de Florida. Y comenzamos el vuelo en la pista 14, única disponible. Utilizamos como aeronave el modelo instalado QAV250; obtenido de ().

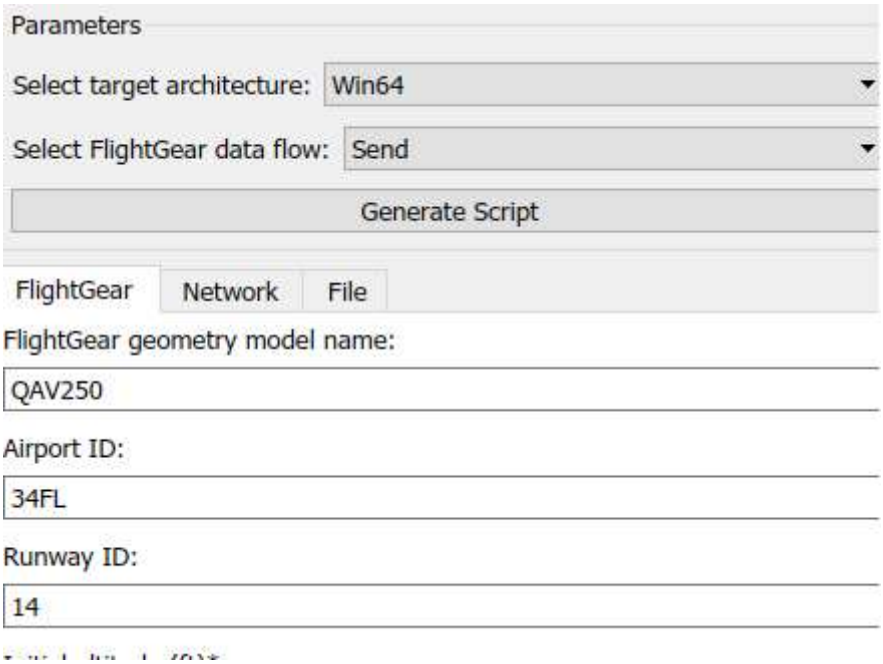


Ilustración 31. Configuración del programa para el lanzamiento del modelo de Flightgear creado con "Generate Run Script", con el código del aeropuerto, pista y nombre del modelo utilizado. Fuente: Elaboración Propia

Una vez creada esta interfaz, es necesario transformar la posición del dron que utilizamos como plano bidimensional en latitud y longitud; que son las magnitudes preferidas por Flightgear. Para ello nos basamos en la implementación que hace el Dr. Christopher Lum, de la universidad de Washington, para su canal de Youtube (4), donde explica la teoría básica para conseguir el cambio de coordenadas. Para simular, es necesario definir diferentes valores y conceptos geográficos.

Se utiliza el sistema ECEF (Earth-centered, Earth-fixed) y la convención NED (North, East, Down) para definir planos y direcciones en el globo. También se considera, cuando nos referimos a longitud; la longitud terrestre, y en latitud a la latitud geodésica, que es la más común; y se obtiene como el ángulo creado por la normal a la superficie de la tierra; que pasa por el objeto de interés; al cruzarse con el plano central de latitud 0.

Para conseguir la transformación de coordenadas, calculamos el radio de curvatura máximo y mínimo que pasan por el punto sobre la superficie de la tierra. Estos radios son el radio Meridiano (M) y el radio Normal (N), que típicamente son planos situados en la dirección N-S y E-O respectivamente. Sus expresiones son:

$$M = \frac{a(1-e^2)}{(1-e^2\sin^2\varphi)^{3/2}}$$

Ecuación 22.Radio del arco de mayor curvatura o meridiano. Fuente: Elaboración Propia a partir de del video del C. Lim (4).

$$\boldsymbol{N = \frac{a}{(1 - e^2 \sin^2 \varphi)^{1/2}}}$$

Ecuación 23 .Radio del arco de menor curvatura o normal. Fuente: Elaboración Propia a partir de del video del C. Lim (4).

Donde *e* es la excentricidad de la tierra y *a* el semieje mayor de la tierra; que según el estándar WGS-84(6), tienen el valor de a=6378137 y e=0.081819190842622. Una vez tenemos estos parámetros, podemos relacionar las velocidades Norte Este y abajo de nuestro dron a la latitud y longitud del sistema:

$$\dot{\varphi} = \frac{V_N}{(M + h)}$$

Ecuación 24.Radio del arco de menor curvatura o normal. Fuente: Elaboración Propia a partir de del video del C. Lim (4).

$$\dot{\lambda} = \frac{V_E}{(N + h)\cos(\varphi)}$$

Ecuación 25.Radio del arco de menor curvatura o normal. Fuente: Elaboración Propia a partir de del video del C. Lim (4).

$$\dot{h} = -V_D$$

Ecuación 26.Radio del arco de menor curvatura o normal. Fuente: Elaboración Propia a partir de del video del C. Lim (4).

Port tanto, es posible crear un subsistema en Simulink que con las velocidades y la altura del dron compute las coordenadas de latitud y longitud en diferencia, pero con una integración y un valor inicial del sistema se puede transformar este diferencial en la magnitud de latitud y longitud deseada. Una vez tenemos las funciones implementadas; se debe pasar la magnitud a grados y saturar el sistema para evitar integraciones a infinito; aunque en el caso de las pruebas de corta distancia no es crítico.

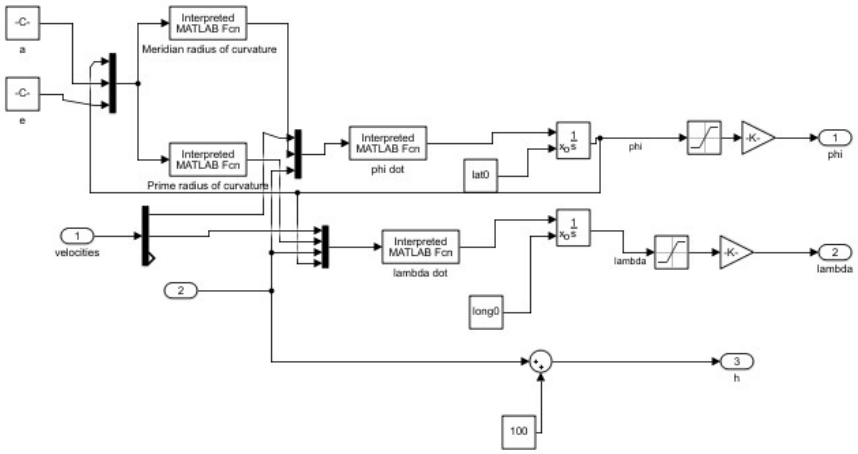


Ilustración 32. Implementación del cambio de coordenadas dentro de un subsistema en Simulink. Fuente: Elaboración Propia a partir de del video del C. Lim

Una vez se ha implementado este sistema de transformación que nos permite el uso de las coordenadas correctas durante la visualización, podemos arrancar el programa creado por "Generate Run Script" y la simulación para ver su visualización y el funcionamiento del dron en tiempo real. Aunque la visualización es aceptable, la escala de los escenarios del programa claramente se pensó para aeronaves de gran tamaño, lo cual provoca la sensación de que el movimiento del dron es muy pequeño en comparación con la pista o los campos colindantes al aeropuerto.

El programa permite cambios en la hora del día y en versiones más nuevas que la utilizada; cambio de condiciones climáticas; aunque el dron real no sería capaz de soportar vientos moderados por su baja potencia y pequeño tamaño. También permite la visualización del estado del dron gracias a un HUD incorporado en el programa que indica las coordenadas, altitud y ángulos de rotación de la aeronave.

Ilustración 33. Captura del dron durante un vuelo simulado. Fuente: Elaboración Propia

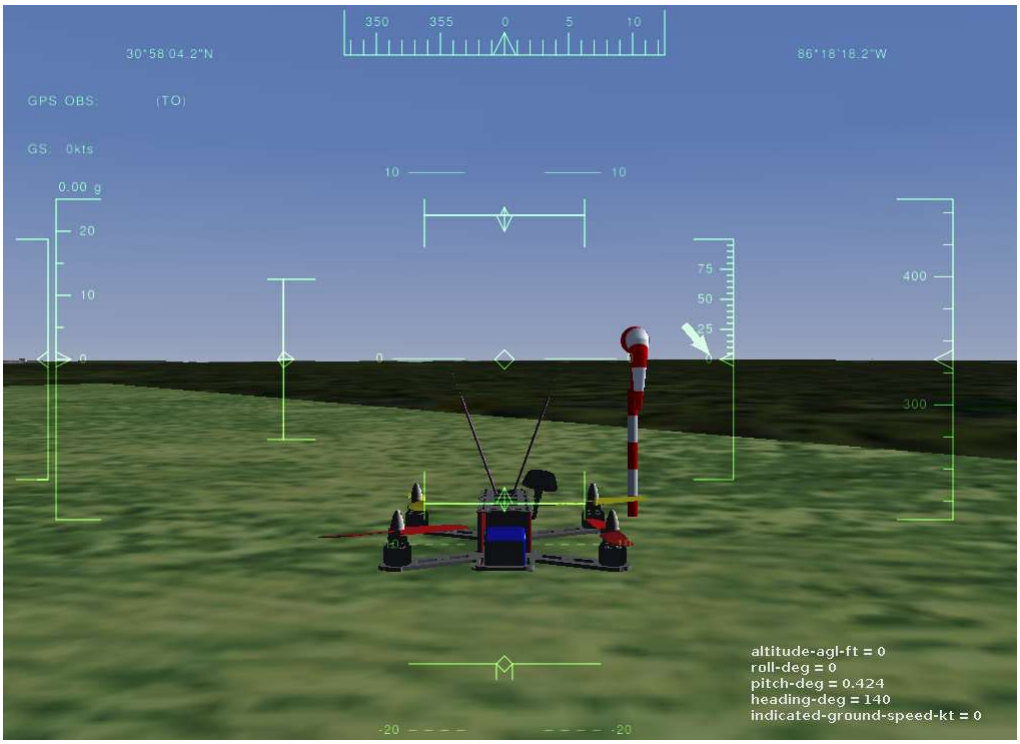


Ilustración 34.Captura del dron durante un vuelo con el HUD indicando la altura, rotación y coordenadas del vuelo. Fuente: Elaboración Propia

5. CONCLUSIONES

Una vez finalizada la simulación gráfica del sistema, conviene revisar los objetivos de este TFM para la validación de su cumplimiento o su no cumplimiento:

- Se ha realizado el estudio de las maniobras del dron y de la estructura de control a implementar para ser capaces de analizar, identificar y aplicar la segunda capa de control de forma eficaz
- Se han creado los modelos lineales en espacio de estados necesarios; resultado de la identificación del modelo del dron, y se han discretizado al periodo de muestreo suficientemente pequeño. Aunque los modelos lineales no suelen ser completamente precisos en rangos fuera de los de su identificación; solo se podrían haber conseguido mejores con un dron real; ya que el dron simulado es lineal.
- Se ha conseguido una mejora significativa del funcionamiento del dron durante el ajuste de los parámetros, y se ha demostrado la capacidad del generador de trayectorias para realizar trayectorias de diversos tipos.
- Se ha enviado con éxito nuestros parámetros de simulación a un entorno gráfico para conseguir la representación gráfica del control en vuelo.

  Una ampliación evidente que se le podría hacer a este TFM sería la prueba del control en un sistema real. Sin embargo; por razones de fuerza mayor no se ha podido acceder a la zona de vuelo de la universidad durante la realización de este trabajo y ha sido imposible la comparación con un dron real. Otra posible ampliación sería la ampliación de los tipos de trayectorias que es posible crear, como espirales o curvas tridimensionales. Sin embargo; una se generan las trayectorias básicas estas trayectorias son sencillas de añadir si se deseara funcionamientos más complejos.

  Aún con estas posibles excepciones, se ha conseguido la realización de los objetivos iniciales de forma exitosa.

6. BILBIOGRAFÍA

1) González, Fabio & Cristancho, M. & López, E. (2018). Modelamiento y simulación de un quadrotor mediante la integración de Simulink y SolidWorks. MASKAY. 9. 15. 10.24133/maskay.v9i1.1043.
2) Transparencias de PoliformaT de la asignatura "Control Industrial Avanzado" J. Sanchís Saez.
3) Transparencias de la asignatura "Robótica Industrial" de A. Valera Fernández.
4) Video tutoriales de Christopher Lim, profesor de la universidad de Washington. https://www.youtube.com/c/ChristopherLum/about (Enlaces funcionales en agosto de 2020) https://www.youtube.com/watch?v=f8tdTiuj5lo&t=1086s (Enlaces funcionales en agosto de 2020.)
5) Modelo del dron para su control y simulación (Enlaces funcionales en agosto de 2020) https://es.mathworks.com/matlabcentral/fileexchange/43719-ar-drone-simulink-development-kit-v1-1
6) https://en.wikipedia.org/wiki/World_Geodetic_System#WGS84 (Enlaces funcionales en agosto de 2020)
7) Modelo QAV 250 https://www.lumenier.com/products/legacy/qav250
8) Página oficial Parrot https://www.parrot.com/en/

Presupuesto

Donde se monetiza el trabajo de control creado para conseguir una aproximación de su coste como proyecto de ingeniería.

7. PRESUPUESTO

7.1. MOTIVACIÓN Y CONTENIDO

La necesidad del presupuesto se presenta en la necesidad de valorar económicamente el coste o viabilidad de este de forma rigurosa. Esta valoración orientativa permite la consideración económica de la realización de esta clase de control para un dron como una herramienta de inspección y vigilancia de zonas agrarias o industrias de planta extensa. Para este estudio se ha considerado la adquisición del dron para la implementación del sistema en un dron real, aunque en este estudio no se ha realizado por la dificultad de acceder a los drones disponibles en el laboratorio durante la realización de este trabajo. Sin embargo; este trabajo solo describe la implementación del control del dron como prototipo, y para el uso a largas distancias sería necesario un sistema de comunicación diferente al utilizado en el modelo, que es conexión Wi-Fi.

El presupuesto se compondrá de las siguientes tablas: Una tabla de precios de materiales, una tabla de precios de mano de obra; y un listado detallado de cada unidad de obra del proyecto , y un agregado final del presupuesto donde se resume y añaden diferentes cargos adicionales necesarios para completar el presupuesto. Para los cálculos de salarios por hora, se ha considerado un total de **244 días laborables al año, con jornadas de 8 horas cada uno de estos**

7.2. TABLA DE PRECIOS DE MANO DE OBRA.

Tabla 1. Precios de la mano de obra

TRABAJADOR	SALARIO ANUAL	COSTE POR HORA
INGENIERO INDUSTRIAL	25.527,97 €	13.08€/h

7.3. TABLA DE PRECIOS DE MATERIALES

Tabla 2. Precios de materiales

MATERIALES/SOFTWARE	PRECIO POR UNIDAD	AMORITZACIÓN	COSTE/HORA
LICENCIA SOFTWARE MATLAB	800€/año	Lineal de 1 año	0.41€/h
DRONE PARROT AR ELITE EDITION	147.39 €/unidad	Lineal de 1 años	0.08€/h
TORRE CON CPU	951.46€	Lineal de 3 años	
RATON Y TECLADO	16.98€	Lineal de 3 años	
MONITOR	124 €	Lineal de 3 años	
LICENCIA WINDOWS 10	158.99€	Lineal de 3 años	
TOTAL,ORDENADOR WORKSTATION ACTUAL	951.46+16.98+124+158.99=1251.43 €/unidad (aprox.)	Lineal de 3 años	0.64€/h

7.4. PRECIOS DESCOMPUESTOS

Tabla 3. Desarrollo de algoritmos de control predictivos cuadráticos

DESCRIPCIÓN	RENDIMIENTO	PRECIO HORARIO	IMPORTE
INGENIERO INDUSTRIAL	320	13.08€/H	4185.6€
PC WORKSTATION	320	0.64€/H	204.8€
LICENCIA DE MATLAB	320	0.41€/H	131.2€
COSTES DIRECTOS COMPLEMENTARIOS*		2%	90.432€
COSTES INDIRECTOS**		3%	138.36€
TOTAL			4750.39€

Tabla 4. Identificación y pruebas de control del algoritmo desarrollado en el dron Parrot.

DESCRIPCIÓN	RENDIMIENTO	PRECIO	IMPORTE
DRONE PARROT AR ELITE EDITION	160	0.08€/H	12.8€
INGENIERO INDUSTRIAL	160	13.08€/H	2092.8€
PC WORKSTATION	160	0.64€/H	102.4€
COSTES DIRECTOS COMPLEMENTARIOS*		2%	44.16€
COSTES INDIRECTOS**		3%	67.56€
TOTAL			2319.72€

*Los costes directos complementarios se aplican a la suma de toda la unidad de obra

**Los costes indirectos se aplican al total una vez aplicado los costes directos

7.5. PRECIOS UNITARIOS

Tabla 5. Precio de las unidades de obra multiplicadas por sus mediciones

UNIDAD DE OBRA	MEDICIÓN	IMPORTE
DESARROLLO DE ALGORITMOS DE CONTROL PREDICTIVOS CUADRÁTICOS	1	4750.39€
IDENTIFICACIÓN Y PRUEBAS DE CONTROL DEL ALGORITMO DESARROLLADO EN EL DRON PARROT	1	2319.72€
TOTAL	1	7070.11€

7.6. PRESUPUESTO DE EJECUCIÓN MATERIAL

Tabla 6. Presupuesto final que incluye beneficios, gastos generales e IVA

CONCEPTO	DESCRIPCIÓN	IMPORTE
PRESUPUESTO DE EJECUCIÓN MATERIAL	Suma de las unidades de obra antes del beneficio industrial e impuestos	7070.11€
GASTOS GENERALES	7%	494.9€
SUMA		7565.01€
BENEFICIO INDUSTRIAL	6%	453.9€
PRECIO EJECUCIÓN POR CONTRATA		8018.91€
IVA	21%	1683.97€
PRESUPUESTO BASE LICITACIÓN		9702.88€

7.7. INFORMACIÓN UTILIZADA

Convenio sobre salarios de ingenieros en estudios técnicos BOE (Fecha de consulta del enlace: agosto 2020)

https://www.boe.es/diario_boe/txt.php?id=BOE-A-2019-14977

Precio licencia de Matlab(Fecha de consulta del enlace: agosto 2020)

https://es.mathworks.com/pricing-licensing.html

Precio del dron Parrot AR(Fecha de consulta del enlace: agosto 2020)

https://www.amazon.es/Parrot-AR-Drone-Elite-Sand-cuadricóptero/dp/B00FS7SSD6

PC Workstation nuevo de gama media, más periféricos recomendados (Fecha de consulta del enlace: agosto 2020)

https://www.pccomponentes.com/pccom-workstation-i-intel-core-i5-9600k-16-gb-1-tb-240-ssd-gtx-1660

ANEXOS A LA MEMORIA

Donde se adjunta el código de Matlab utilizado para el funcionamiento de todos los sistemas de control y de simulación del piloto automático.

8. ANEXOS

8.1. ALGORITMOS DE PRUEBA DE UN CONTROL PREDICTIVO POR MÍNIMOS CUADRADOS

8.1.1. Gxp

```
%% Modelo %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
closeall
clearall
s=tf('s');
Gxp=-3.77/(s*(1+1.6284*s));



SS_modb=ss(Gxp);
SS_mod=minreal(SS_modb);

Ts=0.05; %Periodo de muestreo
Gz=c2d(SS_mod,Ts); %Discretización del modelo



Gzss=ss(Gz);
[A,B,C,D]=ssdata(Gz);

ny = size(C,1); %Número de salidas
nu = size(B,2); %Número de entradas

x=zeros(size(A,2),1);

%% Parametros DMC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c=1;
p=80;

lambda1=1;

alfa1=1;



%% Inicializacion off-line %%%%%%%%%%%%%%%%%%%%%%%%%
P=ssP(p,C,A);
Q=Qss(p,C,A,B);
G=Gss(Q,c);

alfaM=alfa1*eye(p);
lambdaM=lambda1*eye(c);

H=inv(G'*alfaM*G+lambdaM)*G'*alfaM;
```

```
%Valor inicial de las predicciones
y_pred=zeros(ny,1);
deltau_ant=zeros(nu,1);
deltau_k=zeros(c*nu,1); % No hemos movido la VM

u_ant=zeros(nu,1);
u_k=zeros(nu,1); % Nuestro proceso empieza con 0 en la VM


%% Preparando la simulacion %

instantes=1000; %Instantes de simulación
sp=[1]; %Set Point cualquiera
spv=[];

for j=1:p
spv=[spv;sp];
end

u_plot=zeros(instantes,nu);
deltau_plot=zeros(instantes,nu);

%%%Simulamos%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:instantes
y_real=C*x; % Medir la salida



bias=y_real-y_pred; %Calculamos el error
biasv=[];

for j=1:p
biasv=[biasv;bias];
%       biasv=[biasv;ones(p,1).*bias(j)];
end



    deltau_k=H*(spv-P*x-Q*u_ant-1.*biasv);



    deltau_k=deltau_k(1:nu);
    u_k=u_ant+deltau_k; %Cambio en la entrada respecto u_ant

    x=A*x+B*u_k; %Calculamos el valor de x para el siguiente instante
    y_pred=C*x;
    deltau_ant=deltau_k;
    u_ant=u_k;%Actualizamos u_ant con el cambio en la entrada


for j=1:nu
y_plot(i,:)=y_real';
```

```
deltau_plot(i,j)=deltau_k(j); % Guardamos las variaciones de las acciones
de control para graficar
u_plot(i,j)=u_k(j); % Guardamos las acciones de control para graficar
end
end

%% Representacion grafica
figure
subplot(311)
plot(y_plot(:,1))
holdon
plot(sp(1,1).*ones(instantes,1),'-.')
title('Salida 1 del sistema')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

subplot(312)
plot(u_plot(:,1))
title('U1 del sistema')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


subplot(313)
plot(deltau_plot(:,1))
title('DeltaU 1 del sistema')
```

8.1.2. Gyr

```
%% Modelo %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
closeall
clearall
s=tf('s');
Gyr=2.9/(s*(1+2.4488*s));



SS_modb=ss(Gyr);
SS_mod=minreal(SS_modb);

Ts=0.05; %Periodo de muestreo
Gz=c2d(SS_mod,Ts); %Discretización del modelo



Gzss=ss(Gz);
[A,B,C,D]=ssdata(Gz);

ny = size(C,1); %Número de salidas
nu = size(B,2); %Número de entradas

x=zeros(size(A,2),1);

%% Parametros DMC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c=1;
```

```
p=80;

lambda1=1;

alfa1=1;



%% Inicializacion off-line %%%%%%%%%%%%%%%%%%%%%%%%%
P=ssP(p,C,A);
Q=Qss(p,C,A,B);
G=Gss(Q,c);

alfaM=alfa1*eye(p);
lambdaM=lambda1*eye(c);

H=inv(G'*alfaM*G+lambdaM)*G'*alfaM;




%Valor inicial de las predicciones
y_pred=zeros(ny,1);
deltau_ant=zeros(nu,1);
deltau_k=zeros(c*nu,1); % No hemos movido la VM

u_ant=zeros(nu,1);
u_k=zeros(nu,1); % Nuestro proceso empieza con 0 en la VM


%% Preparando la simulacion %

instantes=1000; %Instantes de simulación
sp=[1]; %Set Point cualquiera
spv=[];

for j=1:p
spv=[spv;sp];
end

u_plot=zeros(instantes,nu);
deltau_plot=zeros(instantes,nu);

%%%Simulamos%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:instantes
y_real=C*x; % Medir la salida



bias=y_real-y_pred; %Calculamos el error
biasv=[];

for j=1:p
biasv=[biasv;bias];
```

```
%       biasv=[biasv;ones(p,1).*bias(j)];
end



    deltau_k=H*(spv-P*x-Q*u_ant-1.*biasv);



    deltau_k=deltau_k(1:nu);
    u_k=u_ant+deltau_k; %Cambio en la entrada respecto u_ant

    x=A*x+B*u_k; %Calculamos el valor de x para el siguiente instante
    y_pred=C*x;
    deltau_ant=deltau_k;
    u_ant=u_k;%Actualizamos u_ant con el cambio en la entrada


for j=1:nu
y_plot(i,:)=y_real';
deltau_plot(i,j)=deltau_k(j); % Guardamos las variaciones de las acciones
de control para graficar
u_plot(i,j)=u_k(j); % Guardamos las acciones de control para graficar
end
end

%% Representacion grafica
figure
subplot(311)
plot(y_plot(:,1))
holdon
plot(sp(1,1).*ones(instantes,1),'-.')
title('Salida 1 del sistema')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

subplot(312)
plot(u_plot(:,1))
title('U1 del sistema')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


subplot(313)
plot(deltau_plot(:,1))
title('DeltaU 1 del sistema')
```

8.1.3. Ghy

```
%% Modelo %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
closeall
clearall
s=tf('s');
Ghy_bueno=275/((1+165.26*s)*(1+0.30732));
```

```
SS_modb=ss(Ghy_bueno);
SS_mod=minreal(SS_modb);

Ts=0.05; %Periodo de muestreo
Gz=c2d(SS_mod,Ts); %Discretización del modelo



Gzss=ss(Gz);
[A,B,C,D]=ssdata(Gz);

ny = size(C,1); %Número de salidas
nu = size(B,2); %Número de entradas

x=zeros(size(A,2),1);

%% Parametros DMC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c=2;
p=80;

lambda1=1;

alfa1=1;



%% Inicializacion off-line %%%%%%%%%%%%%%%%%%%%%%%%%
P=ssP(p,C,A);
Q=Qss(p,C,A,B);
G=Gss(Q,c);

alfaM=alfa1*eye(p);
lambdaM=lambda1*eye(c);

H=inv(G'*alfaM*G+lambdaM)*G'*alfaM;




%Valor inicial de las predicciones
y_pred=zeros(ny,1);
deltau_ant=zeros(nu,1);
deltau_k=zeros(c*nu,1); % No hemos movido la VM

u_ant=zeros(nu,1);
u_k=zeros(nu,1); % Nuestro proceso empieza con 0 en la VM


%% Preparando la simulacion %

instantes=1000; %Instantes de simulación
sp=[1]; %Set Point cualquiera
spv=[];
```

```
for j=1:p
spv=[spv;sp];
end

u_plot=zeros(instantes,nu);
deltau_plot=zeros(instantes,nu);

%%%Simulamos%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:instantes
y_real=C*x; % Medir la salida



bias=y_real-y_pred; %Calculamos el error
biasv=[];

for j=1:p
biasv=[biasv;bias];
%       biasv=[biasv;ones(p,1).*bias(j)];
end



    deltau_k=H*(spv-P*x-Q*u_ant-1.*biasv);



    deltau_k=deltau_k(1:nu);
    u_k=u_ant+deltau_k; %Cambio en la entrada respecto u_ant

    x=A*x+B*u_k; %Calculamos el valor de x para el siguiente instante
    y_pred=C*x;
    deltau_ant=deltau_k;
    u_ant=u_k;%Actualizamos u_ant con el cambio en la entrada


for j=1:nu
y_plot(i,:)=y_real';
deltau_plot(i,j)=deltau_k(j); % Guardamos las variaciones de las acciones
de control para graficar
u_plot(i,j)=u_k(j); % Guardamos las acciones de control para graficar
end
end

%% Representacion grafica
figure
subplot(311)
plot(y_plot(:,1))
holdon
plot(sp(1,1).*ones(instantes,1),'-.')
title('Salida 1 del sistema')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
subplot(312)
plot(u_plot(:,1))
title('U1 del sistema')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


subplot(313)
plot(deltau_plot(:,1))
title('DeltaU 1 del sistema')
```

8.1.4. Ghv

```
%% Modelo %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
closeall
clearall
s=tf('s');


Ghv=0.8912/(s*(1+0.45096*s));

SS_modb=ss(Ghv);
SS_mod=minreal(SS_modb);

Ts=0.05; %Periodo de muestreo
Gz=c2d(SS_mod,Ts); %Discretización del modelo



Gzss=ss(Gz);
[A,B,C,D]=ssdata(Gz);

ny = size(C,1); %Número de salidas
nu = size(B,2); %Número de entradas

x=zeros(size(A,2),1);

%% Parametros DMC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c=2;
p=80;

lambda1=1;

alfa1=1;



%% Inicializacion off-line %%%%%%%%%%%%%%%%%%%%%%%%%%
P=ssP(p,C,A);
Q=Qss(p,C,A,B);
G=Gss(Q,c);

alfaM=alfa1*eye(p);
lambdaM=lambda1*eye(c);
```

```
H=inv(G'*alfaM*G+lambdaM)*G'*alfaM;



%Valor inicial de las predicciones
y_pred=zeros(ny,1);
deltau_ant=zeros(nu,1);
deltau_k=zeros(c*nu,1); % No hemos movido la VM

u_ant=zeros(nu,1);
u_k=zeros(nu,1); % Nuestro proceso empieza con 0 en la VM


%% Preparando la simulacion %

instantes=1000; %Instantes de simulación
sp=[1]; %Set Point cualquiera
spv=[];

for j=1:p
spv=[spv;sp];
end

u_plot=zeros(instantes,nu);
deltau_plot=zeros(instantes,nu);

%%%Simulamos%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:instantes
y_real=C*x; % Medir la salida



bias=y_real-y_pred; %Calculamos el error
biasv=[];

for j=1:p
biasv=[biasv;bias];
%       biasv=[biasv;ones(p,1).*bias(j)];
end



    deltau_k=H*(spv-P*x-Q*u_ant-1.*biasv);



    deltau_k=deltau_k(1:nu);
    u_k=u_ant+deltau_k; %Cambio en la entrada respecto u_ant

    x=A*x+B*u_k; %Calculamos el valor de x para el siguiente instante
    y_pred=C*x;
    deltau_ant=deltau_k;
```

```
    u_ant=u_k;%Actualizamos u_ant con el cambio en la entrada


for j=1:nu
y_plot(i,:)=y_real';
deltau_plot(i,j)=deltau_k(j); % Guardamos las variaciones de las acciones
de control para graficar
u_plot(i,j)=u_k(j); % Guardamos las acciones de control para graficar
end
end

%% Representacion grafica
figure
subplot(311)
plot(y_plot(:,1))
holdon
plot(sp(1,1).*ones(instantes,1),'-.')
title('Salida 1 del sistema')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

subplot(312)
plot(u_plot(:,1))
title('U1 del sistema')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


subplot(313)
plot(deltau_plot(:,1))
title('DeltaU 1 del sistema')
```

8.2. ALGORITMOS DE PRUEBA DE UN CONTROL PREDICTIVO POR RESOLUCIÓN DE UNA OPTIMIZACIÓN CUADRÁTICA

8.2.1. Gxp

```
closeall
clearall
s=tf('s');
Gxp=-3.77/(s*(1+1.6284*s));



SS_modb=ss(Gxp);
SS_mod=minreal(SS_modb);

Ts=0.05;
Gz=c2d(SS_mod,Ts); %Discretización del modelo


Gzss=ss(Gz);
[A,B,C,D]=ssdata(Gz);
```

```
ny = size(C,1); %Número de salidas
nu = size(B,2); %Número de entradas

x=zeros(size(A,2),1);
%% Parámetros de limitación de u, deltau e y
deltaumax=0.5;
deltaumin=-0.5;
ymax=6;
ymin=-6;
umin=-1;
umax=1;

%% Parametros DMC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c=10;
p=80;

lambda1=1;

alfa1=1;




%% Inicializacion off-line %%%%%%%%%%%%%%%%%%%%%%%%%
P=ssP(p,C,A);
Q=Qss(p,C,A,B);
G=Gss(Q,c);

alfaM=blkdiag(alfa1*eye(p));
lambdaM=blkdiag(lambda1*eye(c));

H=blkdiag((G'*alfaM*G)+lambdaM,10E09);

I=blkdiag(eye(c)); %Tantos bloques como nu para I e IL
IL=blkdiag(tril(ones(c)));
boff=[deltaumax*ones(c,1);-deltaumin*ones(c,1)];
boff2=[umax*ones(c,1);-umin*ones(c,1);ymax*ones(ny*p,1);-
ymin*ones(ny*p,1)];
Aq=[I zeros(nu*c,1);-I zeros(nu*c,1);IL zeros(nu*c,1);-IL zeros(nu*c,1); G
-1*ones(nu*p,1);-G -1*ones(nu*p,1)];
Aq=[Aq;zeros(1,nu*c), -1];


%Valor inicial de las predicciones
y_pred=zeros(ny,1);
deltau_ant=zeros(nu,1);
deltau_k=zeros(c*nu,1); % No hemos movido la VM

u_ant=zeros(nu,1);
u_k=zeros(nu,1); % Nuestro proceso empieza con 0 en la VM


%% Preparando la simulacion %

instantes=1000; %Instantes de simulación
```

```
sp=3; %Set Point cualquiera
spv=[];

for j=1:p
spv=[spv;sp];
end

u_plot=zeros(instantes,nu);
deltau_plot=zeros(instantes,nu);

for i=1:instantes
if (i<500)
y_real=(C*x)+0.4; % Medir la salida
else
y_real=(C*x)-0.4;
end

bias=y_real-y_pred; %Calculamos el error
biasv=[];

for j=1:p
biasv=[biasv;bias];

end
y_free=P*x+Q*u_ant+1.*biasv;
    error=spv-y_free;
cq=[-G'*alfaM*error;0];
    b=[boff;boff2-[u_ant(1)*ones(c,1);-u_ant(1)*ones(c,1);y_free;-
y_free];0];


options = optimoptions('quadprog','Display','off');
    [deltau_k,fval,exitflag] =quadprog(H,cq,Aq,b,[],[],[],[],[],options);
ifexitflag==-2
disp('PROBLEMA NO FACTIBLE')
end
%deltau_k=quadprog(H,cq,Aq,b);

    deltau_k=deltau_k(1:nu);

    u_k=u_ant+deltau_k; %Cambio en la entrada respecto u_ant

    deltau_ant=deltau_k;
     x=A*x+B*u_k; %Calculamos el valor de x para el siguiente instante
     y_pred=C*x;
    u_ant=u_k;

for j=1:nu
y_plot(i,:)=y_real';
deltau_plot(i,j)=deltau_k(j); % Guardamos las variaciones de las acciones
de control para graficar
u_plot(i,j)=u_k(j); % Guardamos las acciones de control para graficar
end
end
%% Representacion grafica
figure
```

```
subplot(311)
plot(y_plot(:,1))
holdon
plot(sp(1,1).*ones(instantes,1),'-.')
title('Salida 1 de Gxp')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

subplot(312)
plot(u_plot(:,1))
title('U1 de Gxp')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


subplot(313)
plot(deltau_plot(:,1))
title('DeltaU 1 del sistema')
```

8.2.2. Gyr

```
closeall
clearall
s=tf('s');
Gyr=2.9/(s*(1+2.4488*s));



SS_modb=ss(Gyr);
SS_mod=minreal(SS_modb);


Ts=0.05;
Gz=c2d(SS_mod,Ts); %Discretización del modelo






Gzss=ss(Gz);
[A,B,C,D]=ssdata(Gz);

ny = size(C,1); %Número de salidas
nu = size(B,2); %Número de entradas

x=zeros(size(A,2),1);
%% Parámetros de limitación de u, deltau e y
deltaumax=0.5;
deltaumin=-0.5;
ymax=6;
ymin=-6;
umin=-1;
umax=1;
```

```
%% Parametros DMC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c=10;
p=80;

lambda1=1;

alfa1=1;




%% Inicializacion off-line %%%%%%%%%%%%%%%%%%%%%%%%%%
P=ssP(p,C,A);
Q=Qss(p,C,A,B);
G=Gss(Q,c);

alfaM=blkdiag(alfa1*eye(p));
lambdaM=blkdiag(lambda1*eye(c));

H=blkdiag((G'*alfaM*G)+lambdaM,10E09);

I=blkdiag(eye(c)); %Tantos bloques como nu para I e IL
IL=blkdiag(tril(ones(c)));
boff=[deltaumax*ones(c,1);-deltaumin*ones(c,1)];
boff2=[umax*ones(c,1);-umin*ones(c,1);ymax*ones(ny*p,1);-
ymin*ones(ny*p,1)];
Aq=[I zeros(nu*c,1);-I zeros(nu*c,1);IL zeros(nu*c,1);-IL zeros(nu*c,1); G
-1*ones(nu*p,1);-G -1*ones(nu*p,1)];
Aq=[Aq;zeros(1,nu*c), -1];


%Valor inicial de las predicciones
y_pred=zeros(ny,1);
deltau_ant=zeros(nu,1);
deltau_k=zeros(c*nu,1); % No hemos movido la VM

u_ant=zeros(nu,1);
u_k=zeros(nu,1); % Nuestro proceso empieza con 0 en la VM


%% Preparando la simulacion %

instantes=1000; %Instantes de simulación
sp=1; %Set Point cualquiera
spv=[];

for j=1:p
spv=[spv;sp];
end

u_plot=zeros(instantes,nu);
deltau_plot=zeros(instantes,nu);

for i=1:instantes
```

```
y_real=C*x; % Medir la salida



bias=y_real-y_pred; %Calculamos el error
biasv=[];

for j=1:p
biasv=[biasv;bias];

end
y_free=P*x+Q*u_ant+1.*biasv;
    error=spv-y_free;
cq=[-G'*alfaM*error;0];
    b=[boff;boff2-[u_ant(1)*ones(c,1);-u_ant(1)*ones(c,1);y_free;-
y_free];0];


options = optimoptions('quadprog','Display','off');
    [deltau_k,fval,exitflag] =quadprog(H,cq,Aq,b,[],[],[],[],[],options);
ifexitflag==-2
disp('PROBLEMA NO FACTIBLE')
end
%deltau_k=quadprog(H,cq,Aq,b);

    deltau_k=deltau_k(1:nu);

    u_k=u_ant+deltau_k; %Cambio en la entrada respecto u_ant

    deltau_ant=deltau_k;
     x=A*x+B*u_k; %Calculamos el valor de x para el siguiente instante
     y_pred=C*x;
    u_ant=u_k;
for j=1:nu
y_plot(i,:)=y_real';
deltau_plot(i,j)=deltau_k(j); % Guardamos las variaciones de las acciones
de control para graficar
u_plot(i,j)=u_k(j); % Guardamos las acciones de control para graficar
end
end
%% Representacion grafica
figure
subplot(311)
plot(y_plot(:,1))
holdon
plot(sp(1,1).*ones(instantes,1),'-.')
title('Salida 1 de Gyr')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

subplot(312)
plot(u_plot(:,1))
title('U1 de Gyr')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
subplot(313)
plot(deltau_plot(:,1))
title('DeltaU 1 de Gyr')
```

8.2.3. Ghy

```
closeall
clearall
s=tf('s');
Ghy_bueno=275/((1+165.26*s)*(1+0.30732));


SS_modb=ss(Ghy_bueno);
SS_mod=minreal(SS_modb);

Ts=0.05;
Gz=c2d(SS_mod,Ts); %Discretización del modelo




Gzss=ss(Gz);
[A,B,C,D]=ssdata(Gz);

ny = size(C,1); %Número de salidas
nu = size(B,2); %Número de entradas

x=zeros(size(A,2),1);
%% Parámetros de limitación de u, deltau e y
deltaumax=0.5;
deltaumin=-0.5;
ymax=3.14;
ymin=-3.14;
umin=-1;
umax=1;

%% Parametros DMC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c=10;
p=80;

lambda1=1;

alfa1=1;




%% Inicializacion off-line %%%%%%%%%%%%%%%%%%%%%%%%%%
P=ssP(p,C,A);
Q=Qss(p,C,A,B);
```

```
G=Gss(Q,c);

alfaM=blkdiag(alfa1*eye(p));
lambdaM=blkdiag(lambda1*eye(c));

H=blkdiag((G'*alfaM*G)+lambdaM,10E09);

I=blkdiag(eye(c)); %Tantos bloques como nu para I e IL
IL=blkdiag(tril(ones(c)));
boff=[deltaumax*ones(c,1);-deltaumin*ones(c,1)];
boff2=[umax*ones(c,1);-umin*ones(c,1);ymax*ones(ny*p,1);-
ymin*ones(ny*p,1)];
Aq=[I zeros(nu*c,1);-I zeros(nu*c,1);IL zeros(nu*c,1);-IL zeros(nu*c,1); G
-1*ones(nu*p,1);-G -1*ones(nu*p,1)];
Aq=[Aq;zeros(1,nu*c), -1];


%Valor inicial de las predicciones
y_pred=zeros(ny,1);
deltau_ant=zeros(nu,1);
deltau_k=zeros(c*nu,1); % No hemos movido la VM

u_ant=zeros(nu,1);
u_k=zeros(nu,1); % Nuestro proceso empieza con 0 en la VM


%% Preparando la simulacion %

instantes=1000; %Instantes de simulación
sp=1; %Set Point cualquiera
spv=[];

for j=1:p
spv=[spv;sp];
end

u_plot=zeros(instantes,nu);
deltau_plot=zeros(instantes,nu);

for i=1:instantes
y_real=C*x; % Medir la salida



bias=y_real-y_pred; %Calculamos el error
biasv=[];

for j=1:p
biasv=[biasv;bias];

end
y_free=P*x+Q*u_ant+1.*biasv;
    error=spv-y_free;
cq=[-G'*alfaM*error;0];
```

```
    b=[boff;boff2-[u_ant(1)*ones(c,1);-u_ant(1)*ones(c,1);y_free;-
y_free];0];


options = optimoptions('quadprog','Display','off');
    [deltau_k,fval,exitflag] =quadprog(H,cq,Aq,b,[],[],[],[],[],options);
ifexitflag==-2
disp('PROBLEMA NO FACTIBLE')
end
%deltau_k=quadprog(H,cq,Aq,b);

    deltau_k=deltau_k(1:nu);

    u_k=u_ant+deltau_k; %Cambio en la entrada respecto u_ant

    deltau_ant=deltau_k;
     x=A*x+B*u_k; %Calculamos el valor de x para el siguiente instante
     y_pred=C*x;
    u_ant=u_k;
for j=1:nu
y_plot(i,:)=y_real';
deltau_plot(i,j)=deltau_k(j); % Guardamos las variaciones de las acciones
de control para graficar
u_plot(i,j)=u_k(j); % Guardamos las acciones de control para graficar
end
end
%% Representacion grafica
figure
subplot(311)
plot(y_plot(:,1))
holdon
plot(sp(1,1).*ones(instantes,1),'-.')
title('Salida 1 de Ghy')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

subplot(312)
plot(u_plot(:,1))
title('U1 de Ghy')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


subplot(313)
plot(deltau_plot(:,1))
title('DeltaU de Ghy')
```

8.2.4. Ghv

```
closeall
clearall
s=tf('s');
Ghv=0.8912/(s*(1+0.45096*s));
```

```
SS_modb=ss(Ghv);
SS_mod=minreal(SS_modb);

Ts=0.05;
Gz=c2d(SS_mod,Ts); %Discretización del modelo




Gzss=ss(Gz);
[A,B,C,D]=ssdata(Gz);

ny = size(C,1); %Número de salidas
nu = size(B,2); %Número de entradas

x=zeros(size(A,2),1);
%% Parámetros de limitación de u, deltau e y
deltaumax=0.5;
deltaumin=-0.5;
ymax=3;
ymin=0;
umin=-1;
umax=1;

%% Parametros DMC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c=10;
p=80;

lambda1=1;

alfa1=1;




%% Inicializacion off-line %%%%%%%%%%%%%%%%%%%%%%%%%
P=ssP(p,C,A);
Q=Qss(p,C,A,B);
G=Gss(Q,c);

alfaM=blkdiag(alfa1*eye(p));
lambdaM=blkdiag(lambda1*eye(c));

H=blkdiag((G'*alfaM*G)+lambdaM,10E09);

I=blkdiag(eye(c)); %Tantos bloques como nu para I e IL
IL=blkdiag(tril(ones(c)));
boff=[deltaumax*ones(c,1);-deltaumin*ones(c,1)];
boff2=[umax*ones(c,1);-umin*ones(c,1);ymax*ones(ny*p,1);-
ymin*ones(ny*p,1)];
Aq=[I zeros(nu*c,1);-I zeros(nu*c,1);IL zeros(nu*c,1);-IL zeros(nu*c,1); G
-1*ones(nu*p,1);-G -1*ones(nu*p,1)];
Aq=[Aq;zeros(1,nu*c), -1];
```

```
%Valor inicial de las predicciones
y_pred=zeros(ny,1);
deltau_ant=zeros(nu,1);
deltau_k=zeros(c*nu,1); % No hemos movido la VM

u_ant=zeros(nu,1);
u_k=zeros(nu,1); % Nuestro proceso empieza con 0 en la VM


%% Preparando la simulacion %

instantes=1000; %Instantes de simulación
sp=1; %Set Point cualquiera
spv=[];

for j=1:p
spv=[spv;sp];
end

u_plot=zeros(instantes,nu);
deltau_plot=zeros(instantes,nu);

for i=1:instantes
y_real=C*x; % Medir la salida



bias=y_real-y_pred; %Calculamos el error
biasv=[];

for j=1:p
biasv=[biasv;bias];

end
y_free=P*x+Q*u_ant+1.*biasv;
    error=spv-y_free;
cq=[-G'*alfaM*error;0];
    b=[boff;boff2-[u_ant(1)*ones(c,1);-u_ant(1)*ones(c,1);y_free;-
y_free];0];


options = optimoptions('quadprog','Display','off');
    [deltau_k,fval,exitflag] =quadprog(H,cq,Aq,b,[],[],[],[],[],options);
ifexitflag==-2
disp('PROBLEMA NO FACTIBLE')
end
%deltau_k=quadprog(H,cq,Aq,b);

    deltau_k=deltau_k(1:nu);

    u_k=u_ant+deltau_k; %Cambio en la entrada respecto u_ant

    deltau_ant=deltau_k;
     x=A*x+B*u_k; %Calculamos el valor de x para el siguiente instante
```

```
     y_pred=C*x;
    u_ant=u_k;
for j=1:nu
y_plot(i,:)=y_real';
deltau_plot(i,j)=deltau_k(j); % Guardamos las variaciones de las acciones
de control para graficar
u_plot(i,j)=u_k(j); % Guardamos las acciones de control para graficar
end
end
%% Representacion grafica
figure
subplot(311)
plot(y_plot(:,1))
holdon
plot(sp(1,1).*ones(instantes,1),'-.')
title('Salida 1 de Ghv')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

subplot(312)
plot(u_plot(:,1))
title('U1 de Ghv')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


subplot(313)
plot(deltau_plot(:,1))
title('DeltaU 1 de Ghv')
```

8.3. FUNCIÓN DE INICIALIZACIÓN DEL ALGROTIMO DE CONTROL DE LA SIMULACIÓN

```
global  boff1 boff2 boff3 boff4 boff2_1 boff2_2 boff2_3  boff2_4 Aq1 Aq2
Aq3 Aq4  alfaM1 alfaM2 alfaM3 alfaM4     Q1 Q2 Q3 Q4 G1 G2 G3 G4 P1 P2 P3
P4 c1 c2 c3 c4 p1 p2 p3 p4 H1 H2 H3 H4 C1 C2 C3 C4  nu u_ant1 u_ant2 u_ant3
u_ant4
s=tf('s');
Gyr=2.9/(s*(1+2.4488*s));

Ghy_bueno=275/((1+165.26*s)*(1+0.30732));
Gxp=-3.77/(s*(1+1.6284*s));
Ghv=0.8912/(s*(1+0.45096*s));

%%Gxp
SS_Gxp=minreal(ss(Gxp));
%%Gyr

SS_Gyr=minreal(ss(Gyr));

SS_Ghy=minreal(ss(Ghy_bueno));

SS_Ghv=minreal(ss(Ghv));
```

```
Ts=0.05; %Periodo de muestreo

Dxp=c2d(SS_Gxp,Ts); %Discretización del modelo

Dyr=c2d(SS_Gyr,Ts);

Dhy=c2d(SS_Ghy,Ts);

Dhv=c2d(SS_Ghv,Ts);
%%Gxp
[A1,B1,C1,D1]=ssdata(Dxp);
%%Gyr
[A2,B2,C2,D2]=ssdata(Dyr);
%%Ghy
[A3,B3,C3,D3]=ssdata(Dhy);
%%Ghv
[A4,B4,C4,D4]=ssdata(Dhv);
ny = 1; %Número de salidas
nu = 1; %Número de entradas




%% Parámetros de limitación de u, deltau e y
%%Gxp
deltaumax1=1;
deltaumin1=-1;
%%Gyr
deltaumax2=1;
deltaumin2=-1;
%%Ghy
deltaumax3=1;
deltaumin3=-1;
%%Ghv
deltaumax4=1;
deltaumin4=-1;
%%%%%%%
%%Gxp
ymax1=30;
ymin1=-10;
%%Gyr
ymax2=30;
ymin2=-10;
%%Ghy
ymax3=2.6;
ymin3=-2.6;
%%Ghv
ymax4=20;
ymin4=0;
%%%%%%%%%%%%
%%Gxp
umin1=-1;
umax1=1;
%%Gyr
umin2=-1;
umax2=1;
```

```
%%Ghy
umin3=-1;
umax3=1;
%%Ghv
umin4=-1;
umax4=1;

%% Parametros DMC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Gxp
c1=2;
p1=50;
%%Gyr
c2=2;
p2=50;
%%Ghy
c3=1;
p3=40;
%%Ghv
c4=2;
p4=30;
%%Gxp
alfa1=1;
lambda1=5;
%%Gyr
alfa2=1;
lambda2=3;
%%Ghy
alfa3=1;
lambda3=1;
%%Ghv
alfa4=1;
lambda4=1;


%% Inicializacion off-line %%%%%%%%%%%%%%%%%%%%%%%%%
%Gxp
P1=ssP(p1,C1,A1);
Q1=Qss(p1,C1,A1,B1);
G1=Gss(Q1,c1);
%Gyr
P2=ssP(p2,C2,A2);
Q2=Qss(p2,C2,A2,B2);
G2=Gss(Q2,c2);
%Ghy
P3=ssP(p3,C3,A3);
Q3=Qss(p3,C3,A3,B3);
G3=Gss(Q3,c3);
%Ghv
P4=ssP(p4,C4,A4);
Q4=Qss(p4,C4,A4,B4);
G4=Gss(Q4,c4);

alfaM1=blkdiag(alfa1*eye(p1));
lambdaM1=blkdiag(lambda1*eye(c1));

alfaM2=blkdiag(alfa2*eye(p2));
lambdaM2=blkdiag(lambda2*eye(c2));
```

```
alfaM3=blkdiag(alfa3*eye(p3));
lambdaM3=blkdiag(lambda3*eye(c3));

alfaM4=blkdiag(alfa4*eye(p4));
lambdaM4=blkdiag(lambda4*eye(c4));



H1=blkdiag((G1'*alfaM1*G1)+lambdaM1,10E09);

H2=blkdiag((G2'*alfaM2*G2)+lambdaM2,10E09);

H3=blkdiag((G3'*alfaM3*G3)+lambdaM3,10E09);

H4=blkdiag((G4'*alfaM4*G4)+lambdaM4,10E09);



%%%%%%%%%%%%%%%
I_1=blkdiag(eye(c1)); %Tantos bloques como nu para I e IL
IL_1=blkdiag(tril(ones(c1)));
%%%%%%%%%%%%%%%
I_2=blkdiag(eye(c2)); %Tantos bloques como nu para I e IL
IL_2=blkdiag(tril(ones(c2)));
%%%%%%%%%%%%%%%
I_3=blkdiag(eye(c3)); %Tantos bloques como nu para I e IL
IL_3=blkdiag(tril(ones(c3)));
%%%%%%%%%%%%%%%
I_4=blkdiag(eye(c4)); %Tantos bloques como nu para I e IL
IL_4=blkdiag(tril(ones(c4)));



%%%%%%%
boff1=[deltaumax1*ones(c1,1);-deltaumin1*ones(c1,1)];
boff2_1=[umax1*ones(c1,1);-umin1*ones(c1,1);ymax1*ones(ny*p1,1);-
ymin1*ones(ny*p1,1)];
%%%%%%
boff2=[deltaumax2*ones(c2,1);-deltaumin2*ones(c2,1)];
boff2_2=[umax2*ones(c2,1);-umin2*ones(c2,1);ymax2*ones(ny*p2,1);-
ymin2*ones(ny*p2,1)];
%%%%%
boff3=[deltaumax3*ones(c3,1);-deltaumin3*ones(c3,1)];
boff2_3=[umax3*ones(c3,1);-umin3*ones(c3,1);ymax3*ones(ny*p3,1);-
ymin3*ones(ny*p3,1)];
%%%%%
boff4=[deltaumax4*ones(c4,1);-deltaumin4*ones(c4,1)];
boff2_4=[umax4*ones(c4,1);-umin4*ones(c4,1);ymax4*ones(ny*p4,1);-
ymin4*ones(ny*p4,1)];

Aq1=[I_1 zeros(nu*c1,1);-I_1 zeros(nu*c1,1);IL_1 zeros(nu*c1,1);-IL_1
zeros(nu*c1,1); G1 -1*ones(nu*p1,1);-G1 -1*ones(nu*p1,1)];
Aq1=[Aq1;zeros(1,nu*c1), -1];

Aq2=[I_2 zeros(nu*c2,1);-I_2 zeros(nu*c2,1);IL_2 zeros(nu*c2,1);-IL_2
zeros(nu*c2,1); G2 -1*ones(nu*p2,1);-G2 -1*ones(nu*p2,1)];
Aq2=[Aq2;zeros(1,nu*c2), -1];
```

```
Aq3=[I_3 zeros(nu*c3,1);-I_3 zeros(nu*c3,1);IL_3 zeros(nu*c3,1);-IL_3
zeros(nu*c3,1); G3 -1*ones(nu*p3,1);-G3 -1*ones(nu*p3,1)];
Aq3=[Aq3;zeros(1,nu*c3), -1];

Aq4=[I_4 zeros(nu*c4,1);-I_4 zeros(nu*c4,1);IL_4 zeros(nu*c4,1);-IL_4
zeros(nu*c4,1); G4 -1*ones(nu*p4,1);-G4 -1*ones(nu*p4,1)];
Aq4=[Aq4;zeros(1,nu*c4), -1];


%%%%%%%%%%%%%%%%%%%%%%%%
%% Inicializamos funciones
clearfunctions;

%%

%%%Observadores%%%%

polos=[0.9 0.92];
L=place(A1',C1',polos)';
Aob=A1-L*C1;
Bob=[B1 L];
Cob=eye(2);
Dob=[zeros(2,1) zeros(2,1)];
Ob1=ss(Aob,Bob,Cob,Dob,0.05);
Obr1=augstate(SS_Gxp);
%%%%%%%%
polos=[0.9 0.92];
L=place(A2',C2',polos)';
Aob=A2-L*C2;
Bob=[B2 L];
Cob=eye(2);
Dob=[zeros(2,1) zeros(2,1)];
Ob2=ss(Aob,Bob,Cob,Dob,0.05);
Obr2=augstate(SS_Gyr);
%%%%%%%%%%%%%%%%%%%%%%%%
polos=[0.8];
L=place(A3',C3',polos)';
Aob=A3-L*C3;
Bob=[B3 L];
Cob=eye(1);
Dob=[zeros(1,1) zeros(1,1)];
Ob3=ss(Aob,Bob,Cob,Dob,0.05);
Obr3=augstate(SS_Ghy);
%%%%%%%%%%%%%%%%%%
polos=[0.8 0.82];
L=place(A4',C4',polos)';
Aob=A4-L*C4;
Bob=[B4 L];
Cob=eye(2);
Dob=[zeros(2,1) zeros(2,1)];
Ob4=ss(Aob,Bob,Cob,Dob,0.05);
Obr4=augstate(SS_Ghv);
%%%%%%%%%%Flightgear%%%%%%%%%%
%Agriculturalfieldcoordinates.
long0=deg2rad(dms2degrees([-86 18 18.2]));
lat0=deg2rad(dms2degrees([30 58 4.2]));
```

8.4. FUNCIONES CICLICAS UTILIZADAS EN SIMULACIÓN PARA LA RESOLUCIÓN DE LOS ALGORITMOS DE CONTROL

8.4.1. Gxp

```
function u_k=qs_ssfunc1(input)
global Aq1 c1 boff1 boff2_1 alfaM1   H1 C1 nu P1 Q1  G1 p1
sp=input(1);
y_real=input(2);
x_obs=[input(3); input(4)];
spv=[];
persistent u_ant1;


ifisempty( u_ant1)
    u_ant1=0;
end


for j=1:p1
spv=[spv;sp];
end
biasv=[];
y_pred=C1*x_obs;
bias=y_real-y_pred; %Calculamos el error
for j=1:p1
biasv=[biasv;bias];

end


y_free=P1*x_obs+Q1*u_ant1+1.*biasv;




    error=spv-y_free;
cq=[-G1'*alfaM1*error;0];
    b=[boff1;boff2_1-[u_ant1(1)*ones(c1,1);-u_ant1(1)*ones(c1,1);y_free;-
y_free];0];

options = optimoptions('quadprog','Display','off');
    [deltau_k,~,exitflag] =quadprog(H1,cq,Aq1,b,[],[],[],[],[],options);
ifexitflag==-2
disp('PROBLEMA NO FACTIBLE')
end

    deltau_k=deltau_k(1:nu);

    u_k=u_ant1+deltau_k; %Cambio en la entrada respecto u_ant

%deltau_ant=deltau_k;

    u_ant1=u_k;
```

```
end
```

8.4.2. Gyr

```
function u_k=qs_ssfunc2(input)
global Aq2 c2 boff2 boff2_2 alfaM2   H2 C2 nu P2 Q2  G2 p2
sp=input(1);
y_real=input(2);
x_obs=[input(3); input(4)];
spv=[];
persistent u_ant2;


ifisempty( u_ant2)
    u_ant2=0;
end


for j=1:p2
spv=[spv;sp];
end
biasv=[];
y_pred=C2*x_obs;
bias=y_real-y_pred; %Calculamos el error
for j=1:p2
biasv=[biasv;bias];

end


y_free=P2*x_obs+Q2*u_ant2+1.*biasv;




    error=spv-y_free;
cq=[-G2'*alfaM2*error;0];
    b=[boff2;boff2_2-[u_ant2(1)*ones(c2,1);-u_ant2(1)*ones(c2,1);y_free;-
y_free];0];

options = optimoptions('quadprog','Display','off');
    [deltau_k,~,exitflag] =quadprog(H2,cq,Aq2,b,[],[],[],[],[],options);
ifexitflag==-2
disp('PROBLEMA NO FACTIBLE')
end

    deltau_k=deltau_k(1:nu);

    u_k=u_ant2+deltau_k; %Cambio en la entrada respecto u_ant

%deltau_ant=deltau_k;

    u_ant2=u_k;
end
```

8.4.3. Ghy

```
function u_k=qs_ssfunc3(input)
global Aq3 c3 boff3 boff2_3 alfaM3   H3 C3 nu P3 Q3  G3 p3
sp=input(1);
y_real=input(2);
x_obs=input(3);
spv=[];
persistent u_ant3;


ifisempty( u_ant3)
    u_ant3=0;
end


for j=1:p3
spv=[spv;sp];
end
biasv=[];
y_pred=C3*x_obs;
bias=y_real-y_pred; %Calculamos el error
for j=1:p3
biasv=[biasv;bias];

end


y_free=P3*x_obs+Q3*u_ant3+1.*biasv;




    error=spv-y_free;
cq=[-G3'*alfaM3*error;0];
    b=[boff3;boff2_3-[u_ant3(1)*ones(c3,1);-u_ant3(1)*ones(c3,1);y_free;-
y_free];0];

options = optimoptions('quadprog','Display','off');
    [deltau_k,~,exitflag] =quadprog(H3,cq,Aq3,b,[],[],[],[],[],options);
ifexitflag==-2
disp('PROBLEMA NO FACTIBLE')
end

    deltau_k=deltau_k(1:nu);

    u_k=u_ant3+deltau_k; %Cambio en la entrada respecto u_ant

%deltau_ant=deltau_k;

    u_ant3=u_k;
end
```

8.4.4. Ghv

```
function u_k=qs_ssfunc4(input)
```

```
global Aq4 c4 boff4 boff2_4 alfaM4   H4 C4 nu P4 Q4  G4 p4
sp=input(1);
y_real=input(2);
x_obs=[input(3); input(4)];
spv=[];
persistent u_ant4;


ifisempty( u_ant4)
    u_ant4=0;
end


for j=1:p4
spv=[spv;sp];
end
biasv=[];
y_pred=C4*x_obs;
bias=y_real-y_pred; %Calculamos el error
for j=1:p4
biasv=[biasv;bias];

end


y_free=P4*x_obs+Q4*u_ant4+1.*biasv;




    error=spv-y_free;
cq=[-G4'*alfaM4*error;0];
    b=[boff4;boff2_4-[u_ant4(1)*ones(c4,1);-u_ant4(1)*ones(c4,1);y_free;-
y_free];0];

options = optimoptions('quadprog','Display','off');
    [deltau_k,~,exitflag] =quadprog(H4,cq,Aq4,b,[],[],[],[],[],options);
ifexitflag==-2
disp('PROBLEMA NO FACTIBLE')
end

    deltau_k=deltau_k(1:nu);

    u_k=u_ant4+deltau_k; %Cambio en la entrada respecto u_ant

%deltau_ant=deltau_k;

    u_ant4=u_k;
end
```

8.5. FUNCIÓN DE CRECIÓN DE TRAYECTORIAS

```
function y=enviar_trayectorias2(u)

trayectorias={"lineal" [20 0 20] 1 30; %indicamos la x,y y altura , el
tiempo de espera y el tiempo de llegada
"heading" 1 0 10;%indicamos el nuevo heading, , el tiempo de espera y el
tiempo de llegada
"heading" 0 0 10;
"lineal" [20 20 20] 1 30;
%        "circular" [1 1 45] 0 15
};%indicamos el centro , el ángulo , el tiempo de espera y el tiempo de
llegada

xreal=u(1);
yreal=u(2);
head_real=u(4);
hreal=u(3);
t=u(5);
% Numberofwaypoints
ntray=size(trayectorias, 1);



persistenttfinalwp_anthead_anth_antflagtinicialtraycounttesp  llegada
ifisempty(flag)
flag=0;
end
ifisempty(wp_ant)
wp_ant=[0 0 0]; %estas variables permiten trabajar en coordenadas absolutas

end
ifisempty(head_ant)
head_ant=0;
end
ifisempty(h_ant)
h_ant=0;
end
% Init. persistent variables
ifisempty(traycount)
traycount = 1;
end
ifisempty(tfinal)
tfinal= 0;
end
ifisempty(tinicial)
tinicial= 0;
end
ifisempty(llegada)
  llegada= 0;
end
type=trayectorias{traycount,1};
switchtype%determina el tipo de trayectoria y como la creará en cada caso
case"lineal"
        wp1=trayectorias{traycount,2};
if (flag==0)
flag=1;
```

```
deltat=trayectorias{traycount,4};
tinicial=t;
tfinal=tinicial+deltat;
tesp=0;
end
if (t <tfinal)
tn= (t-tinicial)/(tfinal-tinicial);
           r=[(1-tn)*wp_ant+tn*wp1 head_ant];%los waypoints son creaciones
intermedias lineales entre destino y origen



else
           r=[wp1 head_ant];
if (( norm([xrealyreal] - wp1(1:2)) < 0.1 &&norm(hreal - wp1(3)) < 0.1 ))
&& (traycount<ntray) && (llegada==0)
tesp=t+trayectorias{traycount,3};
               llegada=1;
end
if (t >= tesp )  && (llegada==1)
traycount=traycount+1;
wp_ant=wp1;
h_ant=wp1(3);
flag=0;
tesp=0;
            llegada=0;
end

end

case"heading"%muy similar a lineal pero con 1 solo parámetro
        head1=trayectorias{traycount,2};
if (flag==0)
flag=1;
deltat=trayectorias{traycount,4};
tinicial=t;
tfinal=tinicial+deltat;
tesp=0;
end
if (t <tfinal)
tn= (t-tinicial)/(tfinal-tinicial);
           r=[wp_ant (1-tn)*head_ant+tn*head1];%los waypoints son
creaciones intermedias lineales entre destino y origen



else
           r=[wp_ant head1];

if (abs(head_real-head1)< 2*pi/180) && (traycount<ntray) && (llegada==0)
tesp=t+trayectorias{traycount,3};
             llegada=1;
end
if (t >= tesp )&& (llegada==1)
traycount=traycount+1;
head_ant=head1;
tesp=0;
```

```
flag=0;
              llegada=0;
end
end


case"circular"
vec_aux=trayectorias{traycount,2};
        centro=vec_aux(1:2);
angulo=vec_aux(3);
angulor=atan2(wp_ant(2)-centro(2),wp_ant(1)-centro(1));
        Radio=norm(wp_ant(1:2)-centro);
ifangulor>= 0
angulor=angulor*180/pi;
else
angulor=angulor*180/pi*-1+180;
end
        wp1=[Radio*cosd(angulor+angulo)+centro(1)
Radio*sind(angulor+angulo)+centro(2) h_ant];
if (flag==0)
flag=1;
deltat=trayectorias{traycount,4};
tinicial=t;
tfinal=tinicial+deltat;
tesp=0;
end
if (t <tfinal)
           ang1=angulor:sign(angulo):angulor+angulo;
wp=[Radio*cosd(ang1)'+centro(1),
Radio*sind(ang1)'+centro(2),h_ant*ones(size(ang1))'];
tn= (t-tinicial)/(tfinal-tinicial);
tn=ceil(tn*size(wp,1)+0.01*(tn==0));
           r=[wp(tn,1) wp(tn,2) wp(tn,3) head_ant]; %Creación del arco en
waypoints


else
            r=[wp1 head_ant];
if (( norm([xrealyreal] - wp1(1:2)) < 0.1 ) && (traycount<ntray) &&
(llegada==0))
tesp=t+trayectorias{traycount,3};
               llegada=1;
end
if (t >= tesp )  && (llegada==1)
traycount=traycount+1;
wp_ant=wp1;

flag=0;
tesp=0;
             llegada=0;
end
end
end
```

```
%% Outputs:

xref=r(1);
yref=r(2);
href=r(3);
head_ref=r(4);
y(1)=xref;
y(2)=yref;
y(4)=href;
y(3)=head_ref;



end
```