Universidad Zaragoza
1542

Proyecto Fin de Carrera.

CARACTERIZACIÓN EMPÍRICA Y DISEÑO DEL SISTEMA DE CONTROL DE TRAYECTORIA DE UN QUADROTOR MEDIANTE SIMULACIÓN.

Escuela de ingeniería y Arquitectura

Curso 2012/2013.

Autor: Inmaculada Martín Blázquez.

Director: Antonio Romeo Tello.

Resumen del PFC.

En el presente proyecto se pretende realizar una caracterización empírica de un quadrotor, a partir de una simulación en MATLAB de su dinámica realizada por terceros, y diseñar un sistema de control para la trayectoria de este.

Para la caracterización del sistema, se le ha sometido a determinados experimentos para obtener los parámetros que rigen la dinámica del aparato. Estos han sido realizados para obtener la función de transferencia del drone para las diferentes entradas de éste.

Tras esto, se ha planteado un sistema de control de la trayectoria en tres partes: “position control”, “attitude planner” y “attitude controller”. El primer bloque se encarga de realizar el control de las coordenadas X, Y y Z, el segundo se encarga del cálculo de los giros adecuados a realizar para seguir la trayectoria y el tercero se encarga de regular dichos ángulos de rotación.

Para la implementación de los reguladores, se hará una comparativa entre varios tipos y métodos de obtención para una de las partes, y se elegirá el que mejores prestaciones presente, aplicándose dicho procedimiento para el resto de entradas.

Con el fin de facilitar el comando y monitorización del aparato, se desarrollará una interfaz gráfica para uso y disfrute del usuario permitiendo realizar un seguimiento tanto de la evolución de la posición del aparato como de las acciones suministradas por los reguladores y la velocidad resultante en cada motor, además de introducir las coordenadas y la orientación en z deseadas sin necesidad de utilizar la pantalla correspondiente a la simulación del modelo.

Finalmente, se procederá a la obtención de un generador de trayectoria, el cual calculará la trayectoria a seguir para alcanzar la posición y orientación deseadas por el usuario.

Índice.

1.- Objetivo y Alcance del proyecto.

Partiendo de la versión 9.7 de la "Robotic ToolBox" para MATLAB de Peter Corke , el objetivo planteado para este proyecto es obtener una caracterización de su dinámica mediante el sometimiento del quadrotor a diferentes ensayos y realizar un control de trayectoria para éste (3.3).

Dado que todo el proyecto se ha fundamentado en una simulación realizada por terceros, la validez de los datos obtenidos está sujeta a la precisión que hayan tenido los autores del código del programa a la hora de desarrollar las fórmulas físicas que describen el comportamiento del sistema. Algo que se ve bien reflejado con los datos obtenidos en uno de los experimentos.

Las pruebas a las que se le ha sometido al drone, son semejantes a los que se le sometería a uno físico; a excepción de que en ese caso se obtendría el peso mediante una báscula. Sin embargo, al no tratarse en nuestro caso de un ente físico, este parámetro es el aportado por los diseñadores de la simulación.

Por otro lado, el alcance del presente proyecto es el desarrollo tanto del sistema de control como de una interfaz gráfica para uso y disfrute del usuario así como de la creación de un generador de trayectorias adaptado a este tipo de robot.

Respecto al control de trayectoria, se procederá a realizar una estrategia de control de las variables de situación de forma desacoplada: cada variable se controlará de forma independiente a las demás. La validez de dicha estrategia viene ligada al hecho de que la sustentación no se vea afectada cuando se realicen desplazamientos en el plano horizontal, es decir, que los ángulos de giro sobre los ejes x e y ("roll" y "pitch") sean muy pequeños, de forma que su coseno sea aproximable a uno.

Debido a que se trata de un control desacoplado, se procederá a su división en tres partes: "position controller", "attitude planner" y "attitude controller"[1]. El primer bloque se encarga del control de las coordenadas en el espacio de la tarea, por lo que sus entradas serán las X, Y y Z deseadas por el usuario; las salidas serán dos: la correspondiente al control de altitud (el cual se encuentra englobado dentro del control

[1] En este caso, la traducción de los nombres de los bloques serían el controlador de posición, el organizador de situación y el control de situación.

de posición) y las correspondientes a las acciones desarrolladas por los reguladores que controlan los movimientos en el plano horizontal.

Al ser un drone que tiene seis grados de libertad, en principio, debería tener seis variables de localización, tres de posición y tres de orientación. Sin embargo, el usuario sólo podrá introducir cuatro de ellas: las tres correspondientes a su localización en el espacio cartesiano y su orientación respecto del eje z ("yaw"). Es decir, estamos tratando con un sistema subactuado[2] al introducir un número de consignas de entrada menor al realmente necesario.

Las otras dos variables, "roll" y "pitch" (los ángulos de giro sobre x e y respectivamente), serán proporcionados por el control de posición en dicho plano. Estas dos componentes, junto a la "yaw" deseada, actuaran como entradas del organizador de situación, en el cual se llevará a cabo el control de las orientaciones respecto a los tres ejes de coordenadas, siendo la componente de giro sobre z el valor deseado por el usuario y las otras dos componentes de orientación las salida del "position controller" para x e y, las cuales son de carácter transitorio: su valor en régimen permanente (en situación de sustentación[3], en ascenso/descenso o en ausencia de perturbaciones) es nulo.

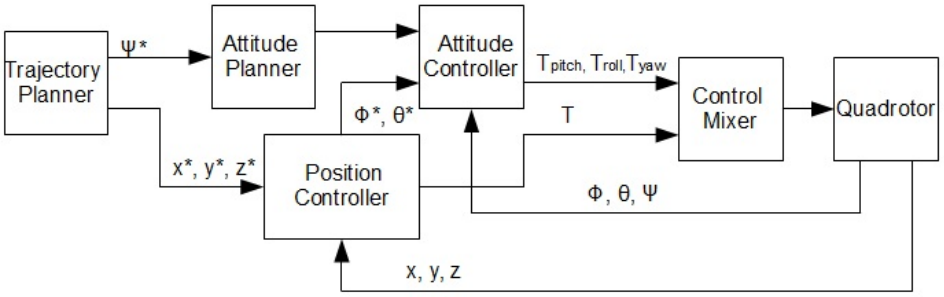

Fig. 1 Diagrama de la distribución por bloques del sistema de control. El diagrama de bloques mostrado pasa a quedar de la siguiente forma en la simulación.

[2] "[…] son aquellos con menos actuadores que grados de libertad". Castro Salguero, R. (2001) Sistemas mecánicos subactuados. Universidad Nacional de Ingeniería, Facultad de Ingeniería Eléctrica y Electrónica. Perú. (Tesis doctoral).

[3] Situación en la que el aparato se encuentra volando a una altura constante y sin que se den desplazamientos o giros en ninguno de los ejes.

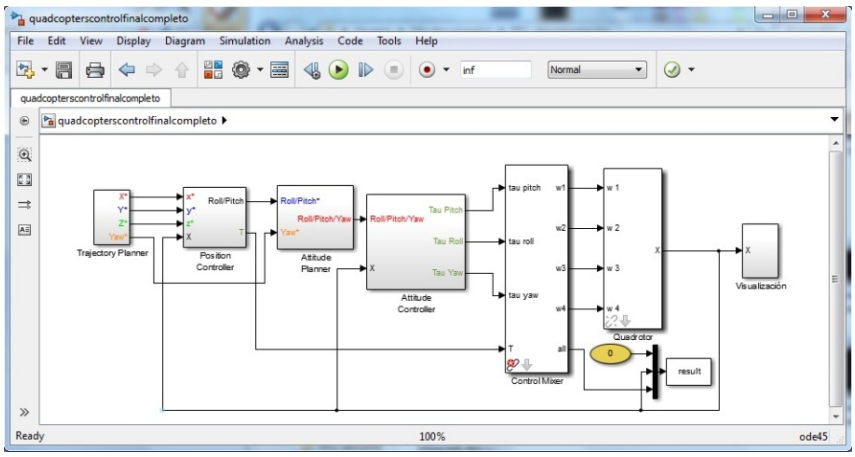

Fig. 2 Diagrama de la simulación en el que se muestra la distribución por bloques del sistema de control.

El objetivo de este bloque es obtener los ángulos de rotación “roll” y “pitch” necesarios para realizar las traslaciones en el plano XY. El control de las tres componentes de rotación se realiza, finalmente, en el control de situación.

Para obtener el sistema de control, se calcularan varios reguladores obtenidos por distintos métodos para el control de altitud, de los cuales se elegirá el que mejores prestaciones tenga: no haya sobreoscilaciones y el tiempo de respuesta sea el más bajo. Una vez elegido, se aplicará el mismo procedimiento al resto de entradas.

Sin embargo, hay que tener en cuenta que en el control de Z se realiza un control lineal sobre una variable que es cuadrática, w^2: el empuje hacia arriba depende de forma directamente proporcional de ésta, mientras que el resto de entradas depende de w.

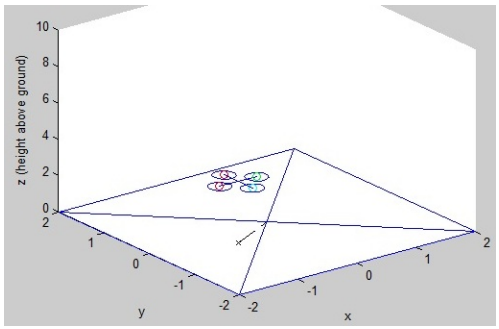

Fig. 3 Vista de la representación del quadrotor en la simulación.

La interfaz gráfica permite la visualización tanto de la trayectoria en 3D seguida como las gráficas correspondientes a las acciones de los reguladores, las velocidades de los motores y las evoluciones temporales de las coordenadas. Otra función que cumple,

además, es la de permitir introducir el destino, la duración y la aceleración que posea la trayectoria desde ésta sin necesidad de manipular la ventana de simulación.

Finalmente, el generador de trayectorias se encargará de calcular la evolución temporal deseada para ésta a partir de los datos aportados por el usuario a través de la interfaz, tras lo cual serán transferidos al workspace general[4] de MATLAB, pudiendo ser utilizados como consignas de entrada para la simulación cuando ésta se ejecute.

[4] Espacio de trabajo con las variables en MATLAB que permite pasar dichas variables a otros entornos de trabajo dentro del núcleo del programa.

2.- Introducción.

2.1.- Introducción a los quadrotors.

Un quadrotor es un robot aéreo de tipo helicóptero de cuatro rotores, también denominado VTO (Vertical tale-off and landing[5]) o UAV (Unmanned Aerial Vehicle[6]), los cuales se encuentran orientados hacia arriba y a la misma distancia del centro de masas del cuerpo del aparato. Además, se suelen construir de forma que éstos se encuentren por encima de dicho punto para que se comporte de forma estable[7].

Fig. 4 Quadrotor construido por Systems Engineering Research Laboratory de la California State University, Northridge (USA).

Sus movimientos se controlan mediante el adecuado manejo de la velocidad de los rotores, los cuales giran en el mismo sentido dos a dos: los motores uno y tres giran en sentido horario mientras que el dos y el cuatro lo hacen en sentido contrario. Es decir, los que están en el mismo brazo giran en el mismo sentido. En función de la velocidad que estos posean, podemos realizar diferentes desplazamientos a lo largo de los ejes, para algunos de los cuales se requerirán unas leves rotaciones respecto del sistema de referencia tomado desde el drone, así como mantenerlo en sustentación.

Si lo que se requiere es hacer un movimiento en el eje Z, es decir, suba o baje, la velocidad de los motores deberá ser mayor o menor que la de sustentación, w_0, pero de igual magnitud para todos ellos. En caso de que se requiera realizar giros sobre dicho

[5] "Despegue y aterrizaje en vertical". Traducido por Inmaculada Martín Blázquez.
[6] "Vehículo aéreo sin piloto". Traducido por Inmaculada Martín Blázquez.
[7] Pounds, P., Mahony, R., Corke, P. (2010). Modelling and control of a large quadrotor robot. Control Engineering Practice, vol. 18. Elsevier.

eje, se incrementa en dos de los motores la velocidad un cierto valor mientras que a los otros dos se ve reducida en dicha cantidad.

Para realizar traslaciones respecto a los ejes X e Y, hay que hacer que uno de los motores de la rama correspondiente a la dirección que se pretende seguir gire a menos velocidad que el que este enfrente: si se quiere mover sobre el eje X, el motor uno deberá girar más despacio que el motor 3. Esto provoca que se dé una rotación sobre el eje Y, modificando el empuje que hay en Z. Si el ángulo es muy pronunciado, éste no será suficiente para mantener en el aire el aparato, de forma que perderá altitud, pudiendo llegar a caer al suelo si no se resuelve la situación antes.

Realmente, la causa de esta pérdida son las limitaciones existentes respecto a las baterías que utilizan esta clase de robot como fuente de alimentación. En la situación descrita anteriormente, el sistema de control requiere una acción tan elevada que la alimentación no puede entregar la energía necesaria para que se dé dicha acción, de ahí la perdida de empuje y de altitud. Sumando a esto el hecho de que es muy difícil de implementar un control como el que podría llevar a cabo un piloto de forma manual, hace que sea complicado el que el drone pueda realizar maniobras arriesgadas como pueden ser giros de 360º sobre alguno de los ejes horizontales sin que acabe en el suelo.

A pesar de ello, este tipo de robots ofrecen la ventaja de tener una dinámica más simple que los de un solo rotor, además de mantenerse mejor en situación de sustentación al ejercerse el empuje con los cuatro rotores. Esto hace que haya una gran competencia entre los grupos de investigación de todo el mundo dedicados a este campo para encontrar un sistema de control que ofrezca un buen rendimiento sin conllevar altos consumos de potencia por parte del microprocesador encargado de realizar los cálculos.

2.2.- Cinemática y dinámica del quadrotor.

Para empezar, tenemos dos sistemas de referencia, un sistema de referencia inercial que denominaremos como absoluta, expresada de forma matemática como $\{A\}=\{x,y,z\}$, y otro sistema que se corresponde con el del drone, expresado como $\{B\}=\{b_1,b_2,b_3\}$ siendo dichos vectores los correspondientes a los ejes de {B} respecto de {A}. La orientación que presenta el aparato viene dada por una matriz de rotación, la de forma que $\{B\}=R\{A\}$.

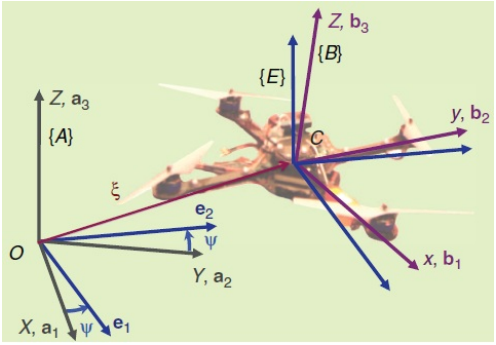

Fig. 5 Sistema de referencia inercial y sistema de referencia del drone

Para expresar dicha, matriz de rotación, utilizamos los ángulos de Euler Z-X-Y de forma que, para ir de {A} a {B} primero hacemos una rotación sobre z, cuyo ángulo se corresponde con "yaw", seguida de una rotación sobre x, que se corresponde con "roll", y, finalmente, una tercera sobre el eje y resultante de la anterior rotación. El resultado de esto es la matriz

$$R = \begin{pmatrix} C\psi C\theta - S\phi S\psi S\theta & -C\phi S\psi & C\psi S\theta + C\theta S\phi S\psi \\ C\theta S\psi + C\psi S\phi S\theta & C\phi C\psi & S\psi S\theta - C\psi C\theta S\phi \\ -C\phi S\theta & S\phi & C\phi C\theta \end{pmatrix}$$

Dicho esto, la cinemática del sólido rígido viene determinada por las leyes de Newton, cuyas ecuaciones son

$$\dot{\xi} = \nu$$

$$m\dot{\nu} = mg\vec{z} + RF$$

$$\dot{R} = R\Omega_{\times}$$

$$I\dot{\Omega} = -\Omega \times I\Omega + \tau$$

Donde $\nu \epsilon \mathbb{R}^3$ es la velocidad lineal del aparato respecto de la referencia inercial, Ω la velocidad angular de {B} respecto {A}, $\Omega_{\times}$ la matriz anti simétrica de Ω de forma que se cumpla que $\Omega_{\times}\nu = \Omega \times \nu$ para el vector resultante del producto vectorial y cualquier vector ν perteneciente a $\mathbb{R}^3$.

Los vectores F y τ, pertenecientes a {B}, representan las fuerzas no conservativas y los momentos aplicados a la estructura del quadrotor debido al movimiento aerodinámico de los rotores.

Respecto a la dinámica, vamos a utilizar un modelado basado en unos conocimientos básicos de aerodinámica, de forma que el modelo utilizado para desarrollar el control no tenga una gran complejidad.

En situación de sustentación, en régimen permanente, el empuje generado por cada rotor se describe como

$$T_i = C_T \rho A_{r_i} r_i^2 \omega_i^2 = c_T \omega_i^2$$

Siendo r_i el radio del rotor i, A_{ri} el área del disco del rotor y C_T el coeficiente de empuje, no dimensional, que depende de la geométria y del perfil del rotor. Estos parámetros, junto con la viscosidad del aire (ρ), se engloban dentro de c_T, la cual se puede obtener de forma experimental (3.2).

El par de reacción que aparece debido al giro del rotor i en el aire se puede describir de forma reducida, semejante al empuje.

$$Q_i = c_Q \omega_i^2$$

Siendo c_q también determinable experimentalmente.

Estas expresiones son válidas siempre que el empuje generado por cada rotor esté orientado en el sentido del eje Z del quadrotor. Sin embargo, esto no siempre se cumple: cuando se producen desplazamientos en horizontal se produce un efecto aerodinámico denominado “rotor flapping”, que produce una inclinación en el plano en el que se encuentra el área de disco del rotor, conllevando con ello una disminución del empuje en vertical. Dado que no se ha tenido en cuenta dicho efecto, al igual que otros efectos que denominaremos de segundo orden, en el cálculo de los reguladores, no se hará más hincapié en él.

Volviendo a la situación de sustentación, el empuje total que se aplica al drone es la suma de todos los empujes generados por cada uno de los rotores de forma individual.

$$T_\Sigma = \sum_{i=1}^{N=4} |T_i| = C_T \left(\sum_{i=1}^{N=4} \omega_i^2 \right)$$

Combinando las fuerzas producidas por los rotores y la resistencia del aire, obtenemos que los empujes aplicados al rotor i son

$$\tau_1 = c_T \sum_{i=1}^{N=4} d_i \sin(\phi_i)\, \omega_i^2$$

$$\tau_2 = -c_T \sum_{i=1}^{N=4} d_i \sin(\phi_i)\, \omega_i^2$$

$$\tau_3 = c_Q \sum_{i=1}^{N=4} \sigma_i \omega_i^2$$

Siendo d_i la distancia del rotor al centro de masas, $\sigma_i \epsilon \{-1,1\}$ para indicar el sentido de rotación del motor, ϕ_i el ángulo que forman los brazos que componen la estructura del quadrotor.

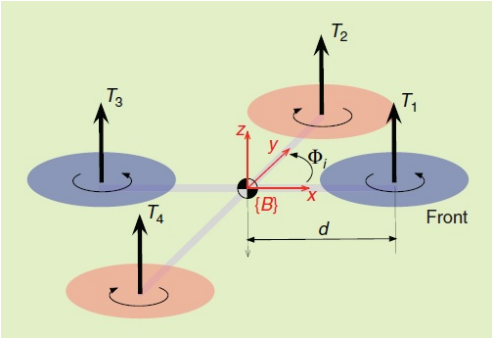

Fig. 6 Notación de las ecuaciones del movimiento del quadrotor.

En forma matricial

$$\begin{pmatrix} T_\Sigma \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{pmatrix} = \begin{pmatrix} c_T & c_T & c_T & c_T \\ 0 & dc_T & 0 & -dc_T \\ -dc_T & 0 & dc_T & 0 \\ c_Q & c_Q & c_Q & c_Q \end{pmatrix} \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix}$$

Los empujes y momentos deseados pueden obtenerse mediante la elección de la ω_i adecuada. En situación de sustentación, el empuje total del aparato es igual al peso del quadrotor, al darse que todos los rotores giran a la misma velocidad. El resto de empujes serán de valor nulo.

3.- Caracterización Experimental del Sistema.

A continuación se describen los experimentos realizados para obtener tanto los parámetros que describen la dinámica del sistema como las características de la función de transferencia, datos necesarios para poder realizar un control de trayectoria con cierta calidad.

Aunque no todos los ensayos se han realizado antes de adentrarnos en parte de control del proyecto, se ha optado por describirlos en esta sección para mayor claridad y orden de la memoria.

3.1.- Signo de la ganancia.

En este experimento se trata de obtener el signo de la ganancia de la función de transferencia. Si ésta es positiva, al introducirle una velocidad angular positiva, el movimiento del quadrotor será en sentido ascendente.

Para realizarlo, hay que tener en cuenta que la velocidad debe hacer que el sistema se eleve o, en caso contrario, el sistema caerá al suelo. En este caso, se le introduce una constante en la entrada T de 900. Al no interesar que se den desplazamientos en el plano XY ni se den cambios en la orientación sobre el eje Z, al resto de entradas se les introduce una constante de valor nulo.

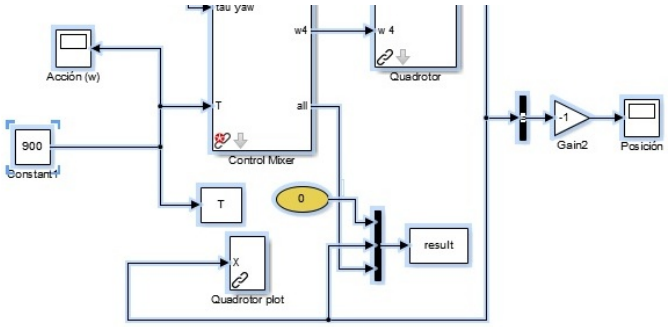

Fig. 7 Velocidad angular constante positiva por la entrada T.

Este valor hace que el sistema se desplace hacia arriba, lo que indica que la ganancia del sistema es positiva.

3.2.- Obtención experimental de c_T.

En este ensayo se trata de hacer que el sistema esté en situación de sustentación: mantenerse en una posición de forma constante en el tiempo. En dicha situación, según

el artículo de Peter Corke y compañeros de investigación, el empuje total del sistema es, siendo las otras entradas nulas:

$$T_\Sigma = 4 * c_t * w_0^2 = m * g$$

siendo w_0 la velocidad que deben tener los motores en dicha situación, m la masa del vehículo, g la aceleración de la gravedad y c_T el coeficiente de empuje, cuyo valor es el que se busca conocer.

Con este experimento lo que realmente se obtiene es la velocidad angular de los motores a la cual el sistema se mantiene sustentado en el aire, a partir del cual, despejando el parámetro de la expresión del empuje total, hallará el valor deseado.

Para ello, realizamos un control de altura, es decir, no deben darse movimientos en X o en Y, de forma que el sistema se mantenga en un valor de Z constante. Dado que no sabemos el tipo del sistema, se ha obtado por utilizar el método Ziegler-Nichols en bucle cerrado ya que nos permite calcular un regulador sin necesidad de conocer en profundidad el sistema.

Este procedimiento se basa en, utilizando regulador proporcional, buscar la ganancia que haga que la respuesta del sistema sea una oscilación sinusoidal. Con este valor y con el periodo de la onda resultante, obtendremos los coeficientes necesarios para implementar o un PI o un PID.

En este caso, manteniendo la cte de 900 para que el sistema se eleve del suelo, obtenemos que para una ganancia de 4.5, el sistema oscila con un periodo crítico de 16.3s. A partir de aquí podemos calcular los coeficientes para cualquiera de los dos reguladores sin ninguna dificultad. En este caso, se ha optado por un control PID dado que su acción inicial, proporcionada por la parte derivativa, hace que el empuje en t=0 tenga un alto valor de forma que se mueva rápidamente en el eje Z.

Aplicando Ziegler-Nichols, los parámetros resultantes son:

$$K_p = {}^{4.5}/_{2.22}$$

$$T_i = {}^{T_c}/_{2} = {}^{16.3}/_{2}$$

$$T_d = {}^{T_c}/_{8} = {}^{16.3}/_{8}$$

Aunque se ha optado por un PID, al poder disponer de la salida ${}^{dz}/_{dt}$, el esquema finalmente empleado es un PI-D.

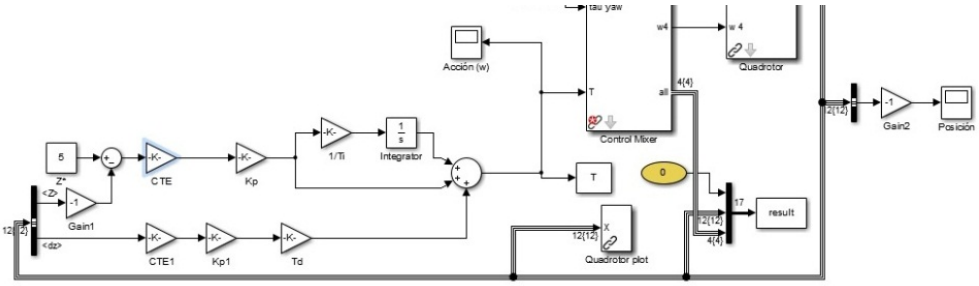

Fig. 8 Bucle de control con PI-D de la altura con una consigna de 5.

Una vez implementado esto, al quitar la constante prealimentada, se aprecia que la acción no es suficiente en los momentos iniciales para hacer que el sistema se eleve. Según Ziegler-Nichols, el valor de la ganancia es acomodable posteriormente a la aplicación del regulador sobre el sistema, por lo que se multiplica los coeficientes del regulador por una constante obtenida de forma experimental de valor 175.

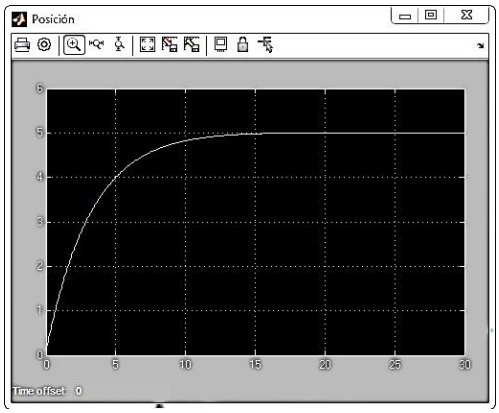

Fig. 9 Evolución de Z en función del tiempo.

Finalmente, se lanza la simulación y, tras ésta, se obtiene la velocidad de sustentación una vez que el quadrotor se encuentra a una altura constante, en este caso de un valor de 5m. El valor de w_0 es de 860.9857 ${}^{rad}/_{s}$. A partir de ahí:

$$c_T = \frac{m * g}{4 * w_0^2} = \frac{4 * 9.81}{4 * (860.9857)^2} = 1.3234 * 10^{-5}$$

3.3.- Obtención experimental de c_Q.

Para la obtención de éste parámetro se obliga al quadrotor a girar sobre su eje Z al aplicar un empuje en la entrada τ_{yaw} del sistema, aprovechando el control utilizado en el anterior experimento.

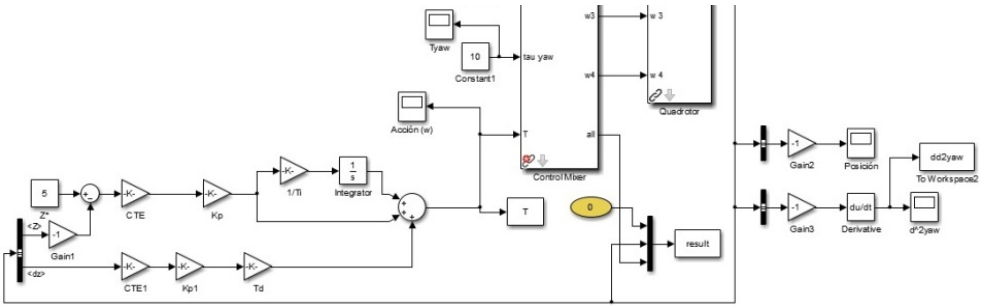

Fig. 10 Obtención experimental de c_Q aplicando un valor constante en τ_{yaw}.

Al iniciar la simulación, el aparato empieza a girar conforme empieza a elevarse hasta que el control hace que se mantenga a la altura de referencia, sin que deje de rotar sobre su eje en vertical.

Sin embargo, al analizar la aceleración obtenida en yaw, se puede ver que, tras una elevación inicial de la aceleración en escalón y un posterior descenso, está aumenta de forma cuasi lineal. Este resultado no se corresponde con lo obtenido con un aparato físico, dado que en ese caso la fricción viscosa del aire haría que ésta fuera decreciendo hasta anularse, al alcanzarse una velocidad de giro constante.

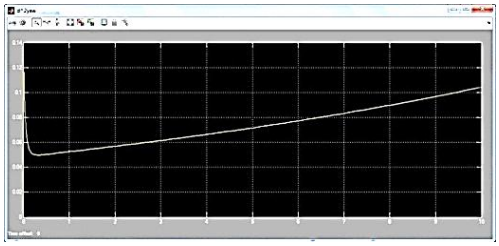

Fig. 11 Gráfica correspondiente a d^2yaw/dt^2, aplicando un valor constante en τ_{yaw}.

A la luz de estos resultados, se hicieron otras pruebas introduciendo un pulso y un escalón en dicha entrada, estando el quadrotor a una altura constante, dando lugar a un comportamiento semejante.

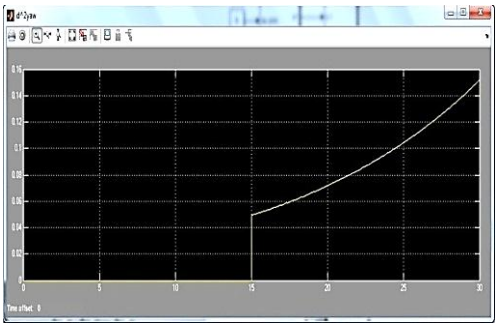

Fig. 12 Gráfica correspondiente a d^2yaw/dt^2, aplicando un escalón en t=15s.

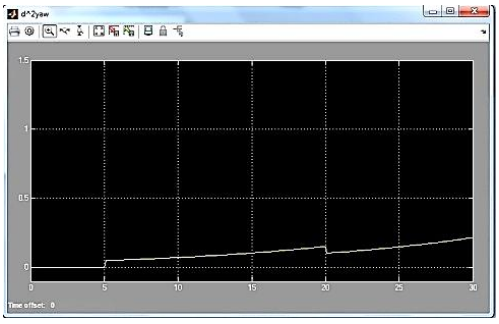

Fig. 13 Gráfica correspondiente a d^2yaw/dt^2, aplicando un pulso en t=5s de 15s de duración.

Finalmente, se obtuvo el valor inicial de la aceleración, el resultante de la aplicación del escalón, para calcular el valor del parámetro c_Q al ser éste el dato que realmente nos interesa con independencia del comportamiento posterior que exhiba el sistema.

Sabemos que el par motor es, aplicando solamente en τ_{yaw}:

$$M_m(t) = -I_{zz} * \frac{d^2yaw}{dt^2} = \tau_3 = c_Q * \sum_{i=1}^{4} \sigma_i * w_i^2$$

siendo σ_i=[-1,+1] para indicar el sentido de giro de las aspas de cada rotor: -1 para sentido anti horario y +1 para sentido horario; siendo I_{zz}=0.1490 la inercia del aparato en Z.

Dada la estructura interna del "control mixer", el controlador de los motores del quadrotor, la expresión resultante del par motor queda:

$$M_m = c_Q * (w_2^2 + w_4^2 - w_1^2 - w_3^2)$$

Que, al aplicar una τ_{yaw}=10 en t=15s, teniendo el motor una velocidad angular de w=869'7955 rad/s en dicho instante, pasa a ser de la forma

$$M_m = c_Q * ((w-10)^2 + (w-10)^2 - (w+10)^2 - (w+10)^2)$$

$$M_m = c_Q * (2 * (w-10)^2 - 2 * (w+10)^2) = c_Q * 2 * (-40w) = -I_{zz} * \frac{d^2yaw}{dt^2}_{t=15}$$

siendo $\frac{d^2yaw}{dt^2}_{t=15} = 0.0494$.

Finalmente, el valor de c_Q, obtenido de esta forma, es de $1.0696*10^{-7}$. De forma teórica, utilizando para ello las expresiones dadas por el artículo [4] del que se parte para la realización del proyecto, se obtiene un valor de $1.0697*10^{-7}$.

Dada la similitud entre ambos valores, se toma como válido el valor inicial de la aceleración en el eje yaw. Para dar más razones de peso a esta hipótesis, se ha consultado una de las obras de referencia citadas en el documento [4] en lo referente al modelado del quadrotor en la que dicen, respecto de las expresiones utilizadas de forma teórica para el cálculo de este parámetro, que "this is calculated *on the basis of uniform inflow and no viscous losse*"[8].

De ahí se deduce que, en base al modelado realizado, se dé esta situación en la simulación, la cual no se podría dar en un espacio físico real al no permitir las leyes de la física que un cuerpo posea una aceleración infinita.

3.4.- Identificación de los polos del sistema.

En vista del resultado obtenido anteriormente (2.3), y como paso previo para el cálculo de un regulador para el control de altitud, el siguiente ensayo a realizar es determinar el tipo de sistema.

El modelado de éste se realiza mediante la siguiente ecuación, en la que se cuenta la gravedad y la fricción viscosa del aire, en el campo transformado de Laplace:

$$4 * c_T * w^2(s) - mg = (m * s + f)s * x(s)$$

De dicha expresión, obtenemos la función de transferencia del sistema:

$$G(s) = \frac{x(s)}{w^2(s)}$$

$$G(s) = \frac{4 * c_T}{s * (m * s + f)}$$

Llegados a este punto, tenemos la que se supone que es la función de transferencia del quadrotor. Para comprobar si realmente es esa función de transferencia la que se ha

[8] "Esto está calculado sobre la base de un flujo de entrada uniforme y sin pérdidas viscosas". Traducido por Inmaculada Martín Blázquez. Leishman, J.G. (2006) Principles of helicopter Aerodynamics. Cambridge: Cambridge University Press. p67.

tenido en cuenta, se le aplica una w ligeramente mayor a la de sustentación y se espera a que se entre en un régimen permanente en el que la velocidad se mantenga constante.

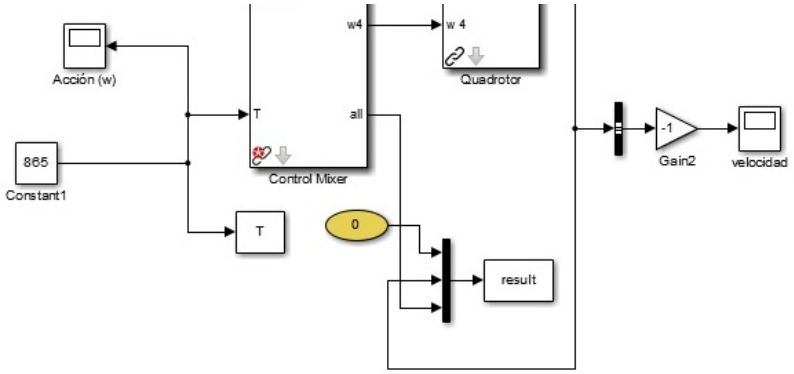

Fig. 14 Experimento para la determinación de los polos del sistema.

Esto es así debido a que para la velocidad, $\frac{dx}{dt}$, el sistema es un primer orden, por lo que su respuesta ante un escalón sería:

$$s(t) = \frac{4 * c_T}{m} * A * (1 - e^{-\frac{f}{m}*t})$$

siendo A la altura del escalón.

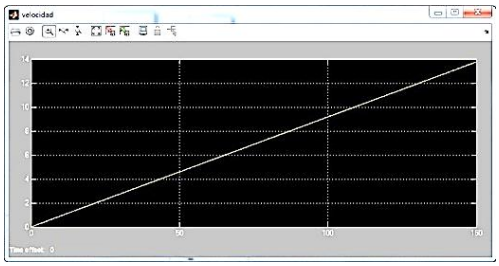

Fig. 15 Representación gráfica de la velocidad ante una entrada de tipo escalón.

Sin embargo, se ve que la respuesta tiene forma de rampa o, lo que es lo mismo, el sistema tiene un polo en el origen. Esto viene a confirmar del todo la hipótesis planteada anteriormente sobre la falta de consideración de la fricción viscosa del aire.

4.- Control de trayectoria.

En este apartado plantearemos las tres partes en que se compone en el control de trayectoria: el control de posición, el organizador y el control de situación[9].

En el primer bloque se encuentran encapsulados tanto el control de las coordenadas X e Y, cuya salida serán los ángulos “roll” y pitch”, como del control de altitud. El siguiente bloque, el “attitude planner”, se encarga de recoger tanto los ángulos de giro adecuados para realizar los traslados en el plano horizontal así como el valor de “yaw” establecido por el usuario. Finalmente, el “attitude controller” se encarga de realizar el control de éstos.

Dichos ángulos son:

- Roll: ángulo de giro sobre el eje X.
- Pitch: ángulo de giro sobre el eje Y.
- Yaw: ángulo de giro sobre el eje Z.

Los cuales difieren de los utilizados tradicionalmente en robótica dado que en este tipo de robots se utilizan los sistemas de referencia de la navegación aérea.

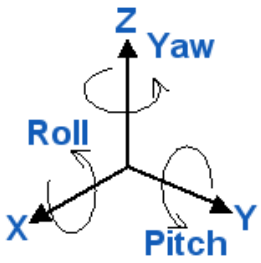

Fig. 16 Componentes de la orientación sobre los ejes de coordenadas cartesianas.

El motivo de realizar esta división es obtener un sistema de control de trayectoria desacoplado, es decir, realizar el control de cada componente de forma individual. La ventaja de esto es que permite que dicho sistema sea más simple al no depender la acción de los reguladores de las otras variables.

Sin embargo, esta división da lugar a un control más delicado en lo referente a los giros sobre los ejes horizontales: tanto “pitch” como “roll” influyen a la hora de realizar desplazamientos en el plano XY de forma que, si se da una fuerte inclinación sobre uno

[9] “Position controller”, “Attitude planner” y “Attitude controller”.

de estos ejes, se reduce el valor del empuje en Z y el quadrotor cae al suelo al no poder compensar dicha pérdida debido a que la acción del control de altitud sólo depende de la altura.

Una vez dicho esto, hay que tener en cuenta la variable sobre la que se va a realizar el control de cada una de las entradas. En el caso del control de Z se va a realizar un control lineal respecto a una variable de control, w^2, la cual no es lineal sino que tiene forma cuadrática. Esto se debe a que el empuje que produce movimientos en dicho eje es dependiente de forma lineal del cuadrado de la velocidad angular de los distintos rotores.

Por el contrario, los empujes que provocan cambios en la componente yaw o en los ejes X e Y no depende de w^2, aunque a simple vista pueda parecerlo, por lo que el control de estas variables sí que se tratará de un control lineal propiamente dicho.

4.1.- Control de altitud.

Para el control de altitud vamos a plantear cuatro tipos de reguladores, obtenidos por cuatro procedimientos distintos, de los cuales se elegirá el que mejores prestaciones presente.

Estos cuatro reguladores son:

1- Regulador proporcional derivativo, calculado utilizando el lugar de las raíces.
2- Regulador con realimentación de doble lazo, “servopropulsor.
3- Regulador proporcional integrador derivativo, utilizando el método de Ziegler-Nichols en bucle cerrado.
4- Regulador obtenido aplicando técnicas frecuenciales.

En todos los casos, la función de transferencia a manejar será la resultante de anular el efecto de las perturbaciones debidas a la gravedad. Esto se hará haciendo una prealimentación de la velocidad de sustentación, w_0^2. Concretamente, la función de transferencia resultante con la que operaremos es la obtenida tras la realización del ensayo “Identificación de los polos del sistema” (3.4).

4.1.1.- Regulador proporcional derivativo: lugar de las raíces.

Viendo los resultados del último experimento llevado a cabo (3.4) para la identificación de los parámetros que describen la dinámica del sistema, sabemos que su función de transferencia tiene dos integradores. Esto conlleva a que los errores de posición y velocidad sean nulos pero hacen que el sistema sea marginalmente estable, de ahí que la respuesta sean oscilaciones sinusoidales respecto el punto correspondiente a la entrada tipo escalón.

En este caso, al aplicar un proporcional derivativo lo que hacemos es incluir un cero en la función de transferencia para dotar al sistema de cierta estabilidad en bucle cerrado.

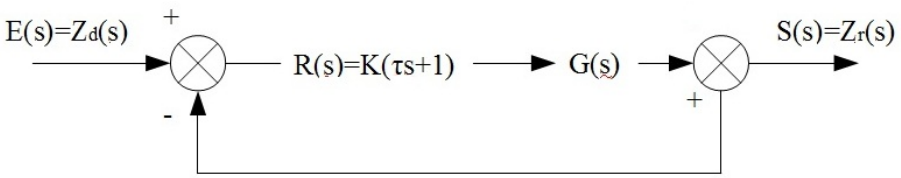

Fig. 17 Diagrama de bloques del regulador Proporcional Derivativo, siendo K la parte proporcional y τ la constante de tiempo de la parte derivativa. G(s) es la función de transferencia del sistema para Z. Z_d es el valor introducido por el usuario.

Incluyendo el regulador, la función de transferencia resultante en bucle cerrado, con la perturbación debida al efecto de la gravedad sobre el aparato anulada con la prealimentación de velocidad, es:

$$F(s) = \frac{R(s) * G(s)}{1 + R(s) * G(s)}$$

$$F(s) = \frac{K_d * (\tau_d * s + 1) * {}^{4 * c_T}/_{m}}{s^2 + K_d * (\tau_d * s + 1) * {}^{4 * c_T}/_{m}}$$

Para el cálculo del regulador nos serviremos del método del lugar de las raíces el cual se basa en la representación del lugar geométrico de los polos en función del valor de la ganancia del sistema en bucle cerrado.

El punto de partida para el trazo de la gráfica es una ganancia nula, valor para el cual la función de transferencia es también nula. A partir de ese punto, parten dos arcos de circunferencia, la cual se encontrará centrada en el valor del cero, correspondientes a las posiciones que toman los polos conforme aumenta K.

Si se aumenta mucho el valor de la ganancia, se llegará a un punto en el que los polos serán reales, punto en el cual el sistema pasa a ser estable. A partir de dicho valor, el seguir aumentando el valor de K provocará que uno de los polos tienda a infinito, quedando anulado su efecto, mientras que el otro tenderá a acercarse al cero. En función de la influencia que queramos que tengan los polos, obtendremos el valor de la ganancia.

En nuestro caso, no sólo tenemos que obtener un valor de K, sino que también tenemos que obtener un valor de τ_d: para obtener el primero, hemos de dar un valor al segundo parámetro previamente. El criterio más fácil para ello es tener en cuenta el tiempo de subida; esto se tratará con más detalle más adelante.

En busca de obtener la mayor cantidad de datos posibles en los que basar la decisión final, empezamos en un punto medio, τ_d=1s y realizamos la representación con ayuda de MATLAB.

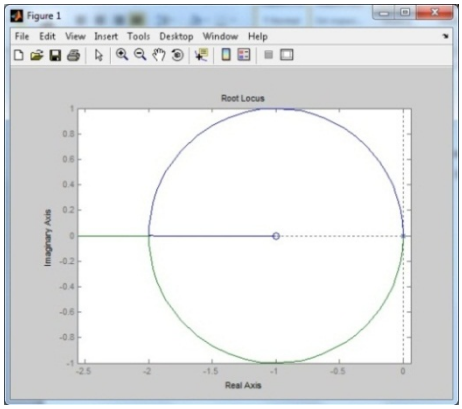

Fig. 18 Representación de la posición de los polos respecto de la ganancia con una τ_d=1s.

Por ejemplo, para una posición en el eje de abscisas de -2.2, el valor de la ganancia es de $3.05*10^5$. Al ser la constante de 1s, los valores de K resultan tan elevados debido a que la ganancia del sistema, $c_{T,}$ es del orden de 10^{-5}.

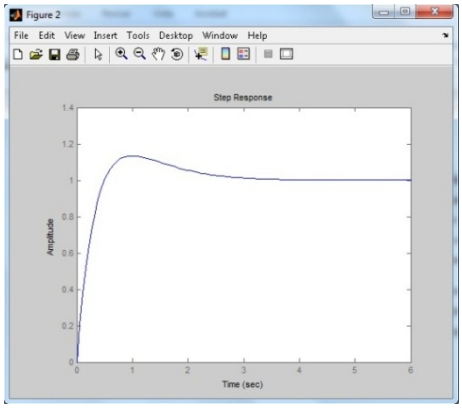

Fig. 19 Respuesta del sistema ante una entrada de tipo escalón con $K_d=3.05*10^5$.

Una vez visto el comportamiento en bucle cerrado, pasamos a implementar el regulador en nuestro esquema. Sin embargo, vemos que en este caso no se produce una oscilación.

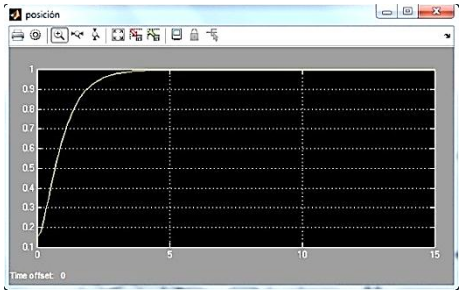

Fig. 20 Respuesta del sistema ante una entrada de tipo escalón con una $K_d=3.05*10^5$.

Esta diferencia de comportamientos se debe a posibles errores en la obtención de la función de transferencia del drone. Sin embargo, al dar unas buenas prestaciones, admitiremos como bueno el regulador obtenido con el lugar de las raíces de F(s).

Aplicando una K_d de $3.05*10^5$, obtenemos un tiempo de subida de unos 7.15s [10]. En caso de que se deseara hacer más rápida la respuesta, se podría hacer disminuyendo la constante de tiempo del regulador y provocando, a su vez, un aumento del valor de la ganancia necesaria para estabilizar el sistema. Además de provocar acciones iniciales más elevadas.

Si bien el sistema presenta un comportamiento muy deseable, el hecho de utilizar valores tan elevados de ganancia hace que no se pueda perder de vista el valor de las

[10] Consideramos el tiempo de subida el tiempo que tarda en alcanzar la salida el valor de entrada desde el 0% hasta el 100%.

acciones del regulador, teniendo en cuenta de que la prealimentación de la w_0^2 adiciona una acción de unos $7.5*10^5$ a la resultante del regulador. Al tratarse de un proporcional derivativo, el valor de la acción a tener en cuenta será la existente en t=0, la cual será muy elevada.

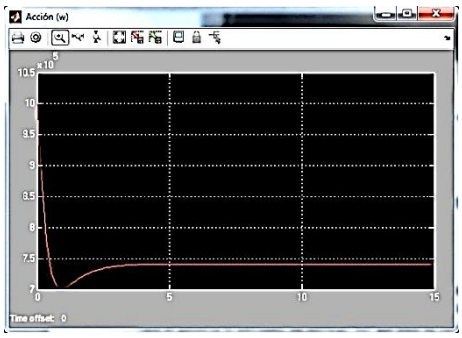

Fig. 21 Acción del regulador proporcional derivativo con $K_d=3.05*10^5$ y $\tau_d=1s$.

En concordancia con la acción de la prealimentación del cuadrado de la velocidad de sustentación y la elevada ganancia del regulador, la acción inicial que se da es casi del orden de 10^6. Para evitar estos valores, se puede reducir la ganancia a costa de aumentar la τ_d de la parte derivativa, lo que tiene como consecuencia un aumento del tiempo de subida.

Otra opción es ir disminuyendo la ganancia, manteniendo el valor de 1s, al diferir la respuesta del sistema incluyendo la prealimentación de w_0^2 de la correspondiente a F(s). Sin embargo, si se quiere mantener el tipo de respuesta, el valor de K_d no podrá disminuirse en demasía. Concretamente, para ganancias menores de $2.5*10^5$, la respuesta presentará sobrepasamientos del valor de consigna; la ganancia final será de $2.5*10^5$.

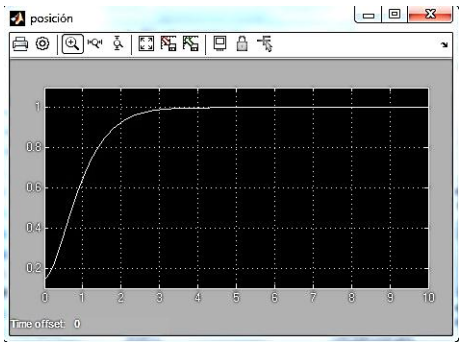

Fig. 22 Respuesta con $K_d=2.5*10^5$ y $\tau_d=1s$.

El disminuir la ganancia sin modificar la constante de tiempo, da lugar a una disminución del tiempo de subida, pasa a ser de unos 5s, sin haber tenido que sacrificar el amortiguamiento del sistema. Respecto a la acción, la disminución del valor en t=0 apenas es significativa.

En caso de tomar como premisa el evitar aumentar la ya de por si acción inicial elevada, se podría optar por aumentar la τ_d. Sin embargo, esto ralentizará el sistema en demasía: con una τ_d=2.5s, se pasaría a tardar más de 16s en alcanzar 1m de altura.

4.1.2.- Regulador con realimentación de doble lazo: "servopropulsor".

Este tipo de control está compuesto por dos proporcionales y dos lazos de realimentación: uno de posición y otro de velocidad.

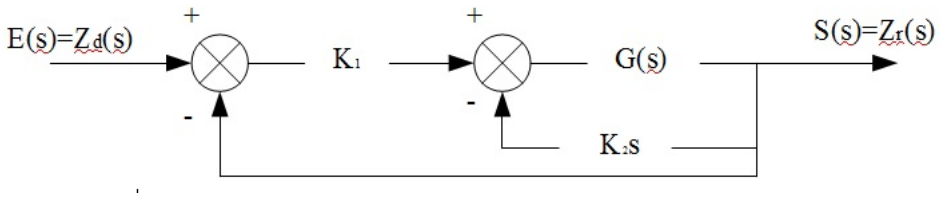

Fig. 23 Diagrama de bloques del regulador con realimentación de doble lazo o "servopropulsor". Z_d es el valor introducido por el usuario.

Simplificando el esquema, resulta una función de transferencia

$$F(s) = \frac{G(s) * K_1 \Big/ 1 + K_2 * G(s) * s}{1 + G(s) * K_1 \Big/ 1 + K_2 * G(s) * s} = \frac{G(s) * K_1}{1 + K_2 * G(s) * s + G(s) * K_1}$$

$$F(s) = \frac{K_1 * K/s^2}{1 + K_2 * K/s^2 * s + K_1 * K/s^2} = \frac{K_1 * K}{s^2 + K_2 * K * s + K_1 * K}$$

$$F(s) = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}$$

El sistema resultante es un segundo orden cuya respuesta depende de los valores de los proporcionales del esquema, los cuales se obtienen a partir de las especificaciones que elijamos para el sistema. En nuestro caso: que no haya sobreoscilaciones

amortiguadas, lo que conlleva a una ξ=1, y un tiempo de respuesta[11] de 1s, para ello ω_n=4.75.

$$K_1 * K = \omega_n^2$$

$$K_1 = \frac{\omega_n^2}{K} = \frac{4.75^2}{4 * {c_T}/{m}}$$

siendo m=4Kg y c_T=1.3234*10^{-5}: K_1=1.7049*10^6.

Y, el cálculo de la segunda constante:

$$2\xi\omega_n = K_2 * K = 2\omega_n$$

$$K_2 = \frac{2\omega_n}{K} = \frac{2\omega_n}{4 * {c_T}/{m}}$$

$$K_2 = \frac{2\omega_n m}{4c_T} = 7.1785 * 10^5$$

Resumiendo, utilizando un servopropulsor obtenemos un sistema de segundo orden cuyo coeficiente de amortiguamiento y su frecuencia natural dependen de dos constantes; sus valores dependerán de las especificaciones a considerar.

Una vez calculados, pasamos a ver su comportamiento con el drone.

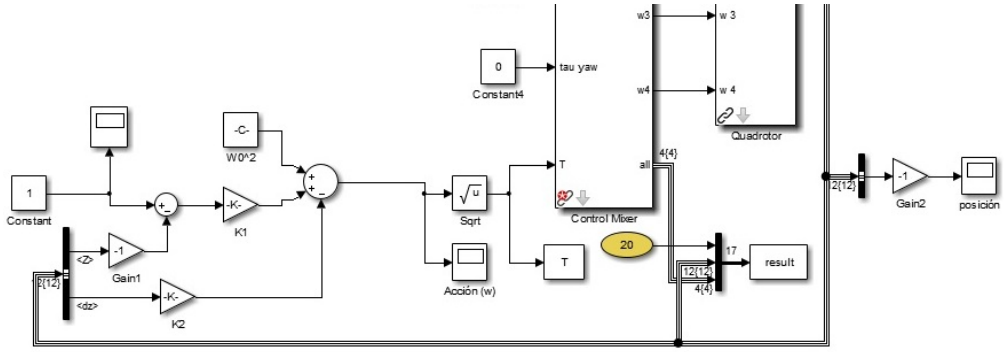

Fig. 24 Esquema del servopropulsor en simulink.

Al aplicar este control a nuestro sistema vemos que, efectivamente, se cumple el tiempo de respuesta establecido en el diseño del regulador. La acción inicial es superior a 10^6.

[11] Tiempo que tarda la salida en alcanzar una franja de valores entorno a un 5% del valor final.

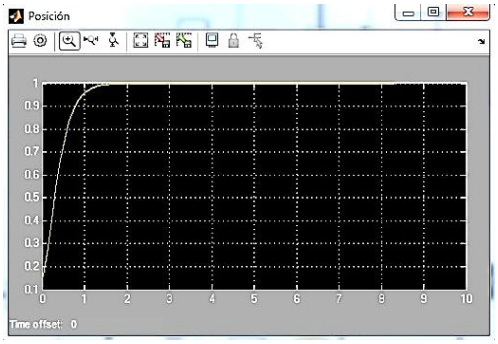

Fig. 25 Respuesta del quadrotor con drone.

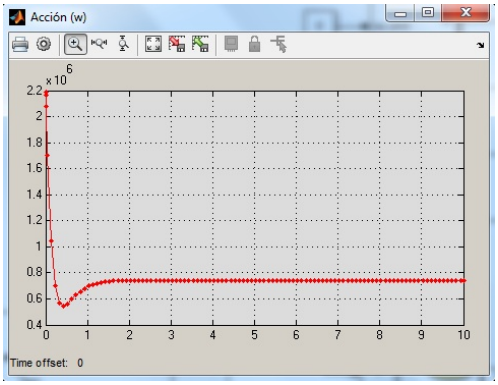

Fig. 26 Acción del servopropulsor.

4.1.3.- Regulador proporcional integral derivativo (PID): método de Ziegler-Nichols en bucle cerrado.

Este método es el mismo que se utilizó a la hora de realizar el experimento para obtener el valor de c_T (2.2), por lo que no nos explayaremos mucho en su explicación.

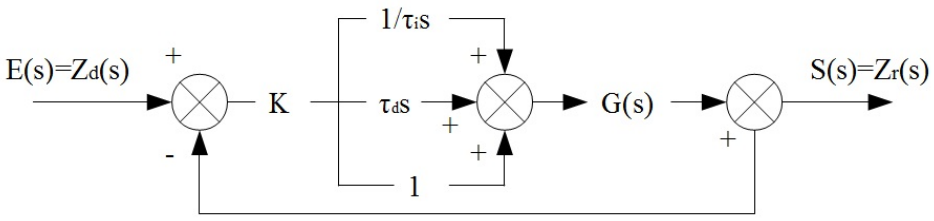

Fig. 27 Diagrama de bloques del regulador proporcional integral derivativo en serie, siendo Z_d el valor introducido por el usuario.

Dado que es un sistema con dos integradores, el sistema oscilara con una ganancia muy pequeña, sin embargo, debido a nuestro control lineal con w^2 y a que las entradas al quadrotor se corresponden con w, utilizaremos un valor de ganancia relativamente elevado con vistas al regulador final.

El valor del periodo crítico es de 17.248s, por lo que los coeficientes del PID serán:

$$\tau_i = \frac{17.248}{2}$$

$$\tau_d = \frac{17.248}{8}$$

$$K_p = \frac{10000}{1.7}$$

Siendo τ_i la constante de tiempo de la parte integradora, τ_d la constante de tiempo de la parte derivativa y K_p la parte proporcional del regulador.

Con dichos valores, aplicamos el regulador al sistema obteniendo una respuesta subamortiguada.

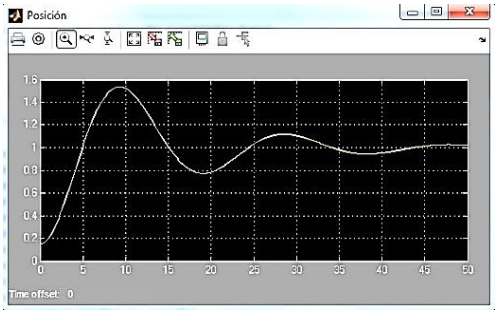

Fig. 28 Respuesta ante un escalón con un regulador PID.

En aras de buscar la mejor respuesta posible, se aumenta el valor de la ganancia. A pesar de ello, la mejor respuesta posible es con cierto sobrepasamiento de la señal de consigna, para una ganancia del proporcional de 10^7, y conllevando acciones iniciales muy fuertes.

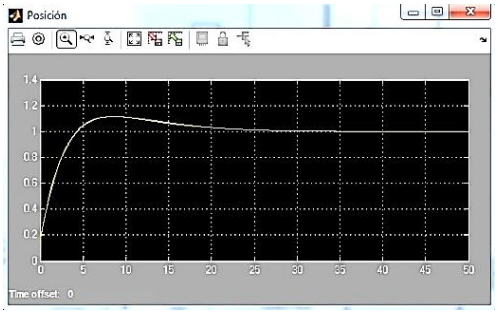

Fig. 29 Respuesta ante un escalón con un PID con $K_p=10^7$.

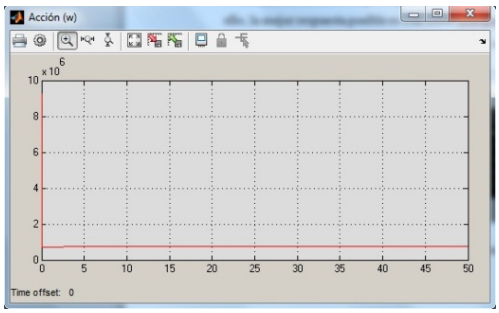

Fig. 30 Acción del PID con una ganancia de 10^7 ante una entrada en escalón.

4.1.4.- Regulador obtenido aplicando técnicas frecuenciales.

Para el cálculo del regulador utilizando métodos frecuenciales, lo primero que hay que hacer es representar el diagrama de Bode del sistema, G(s), a partir del cual conoceremos los aportes que necesita, tales como aumentos o decrementos de ganancia así como de fase para que se ajuste a las especificaciones cuyo cumplimiento se exijan.

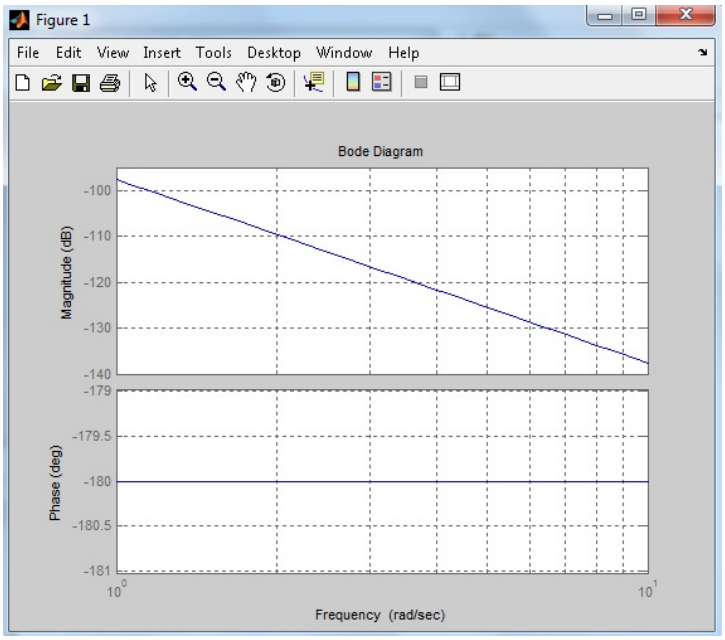

Fig. 31 Diagrama de Bode de G(s). Gráfica de arriba: diagrama de módulos; gráfica de abajo: diagrama de fases.

En la gráfica se puede ver todo lo que ya sabemos sobre el sistema. El diagrama de módulos tiene forma de rampa con una pendiente de $-40^{dB}/_{déc}$, lo cual nos indica que hay dos integradores en el origen; el hecho de que no haya transiciones significa que no hay más polos o ceros en la función de transferencia. También se detecta la existencia de los integradores en el hecho de que el diagrama de fases es una línea recta de valor -180º.

Una vez analizado el diagrama de Bode de G(s), analizamos la estabilidad del sistema midiendo el margen de fase y el margen de ganancia. El primero es "el ángulo que debe girar la curva G(jω) con respecto al origen, para que el punto de cruce de la ganancia del lugar geométrico pase a través del punto (-1, j0)"[12] de la traza de Nyquist mientras que el segundo parámetro es la distancia, en el diagrama de módulos, del punto de cruce de fases al punto (-1, j0).

Según el signo de estos, el sistema será estable o inestable en bucle cerrado. Si son positivos ambos ($\omega_f > \omega_g$) el sistema será estable; en caso contrario, será inestable.

Para analizar la estabilidad, en nuestro caso, no tenemos en consideración el margen de ganancia dado que este se determina de la forma

$$M_g = 0dB - |\,G(j\omega)|_{\mathrm{Arg}(G(j\omega))=-180º}$$

y en nuestro caso éste es positivo.

Otra forma de entender el margen de fases es como el valor del argumento de G(jω), cuando su módulo vale 0dB, más 180º.

$$M_f = G(j\omega)_{|G(j\omega)|=0} + 180º$$

En conclusión, nuestro sistema es marginalmente estable. Necesitamos un aporte de fase para que el sistema sea estable: necesitamos un PAF.

El regulador proporcional de avance de fase lo que provoca es un cambio en el diagrama de fases a la frecuencia que deseemos.

$$R(s) = \frac{1+\tau s}{1+\alpha\tau s}, \qquad 0 < \alpha < 1$$

Como podemos ver, este regulador tiene un polo y un cero, siendo el cero más rápido que el polo. Esto se traduce, en términos frecuenciales, a que tiene dos frecuencias angulares, siendo la del polo mayor que la del cero. Es decir, éste último empieza a actuar antes que el primero. Eligiendo adecuadamente τ y α tendremos el aporte de fase que necesitamos en la banda frecuencial elegida. También provoca un aumento de la ganancia a altas frecuencias, aumentando la frecuencia de corte.

[12] Piedrafita Moreno, R., Romeo Tello, A. (2009) Control Automático en los domionios frecuencial y de tiempo discretos. Zaragoza: Kronos Editorial. p57.

Concretamente,

$$\omega >> \omega_{máx}: 20\log_{10}(1/\alpha)$$

$$\omega = \omega_{máx}: 10\log(1/\alpha)$$

Las especificaciones de diseño establecidas para este regulador son: S.O.=0%, t_r=1s y ε_p=0, el cual se cumple por las características del sistema. Una vez establecidas, el procedimiento de diseño es:

1. Determinar el M_f y la ω_c para el régimen frecuencial transitorio que deseamos. Estos valores los obtendremos partiendo de las especifiaciones anteriormente descritas:

$$S.O. = e^{-\xi\pi/\sqrt{1-\xi^2}} = 0\% \underset{\xi=1}{\Longrightarrow} M_f^d = 80º^{13}$$

$$t_r = \frac{4.75}{\xi\omega_n} = 1s \underset{\xi=1}{\Longrightarrow} \omega_n = 4.75$$

$$\frac{\omega_c^d}{\omega_n} = \sqrt[2]{\sqrt[2]{4\xi^4+1} - 2\xi^2} \underset{\xi=1}{\Longrightarrow} \frac{\omega_c^d}{\omega_n} = 0.4859$$

$$\omega_c^d = 2.3079\, rad/s$$

2. Parametrización del PAF. Para aprovechar de forma eficiente el máximo pico de fase que aporta el regulador, hacemos que $\omega_c^d = \omega_{máx}$, pasando a ser la frecuencia de corte deseada.

$$\phi_{máx} = M_f^d - M_f^{Arg\left(G(j\omega=j\omega_c^d)\right)} = 80º - 0 = 80º$$

$$\alpha = \frac{1-\sin\phi_{máx}}{1+\sin\phi_{máx}} = 0.076543$$

$$\tau = \frac{1}{\omega_{máx}\sqrt[2]{\alpha}} = 4.9526$$

3. Ajustar la ganancia en función del ε_p. En nuestro caso no hace falta, dado que éste es nulo al tener dos integradores en cadena directa.

[13] Con un margen de fase de 75º sería suficiente pero, para asegurarnos el régimen transitorio, le sumamos un coeficiente de seguridad de ζ=5º.

Una vez hecho todo esto, comprobamos como ha cambiado el diagrama de Bode del nuevo sistema.

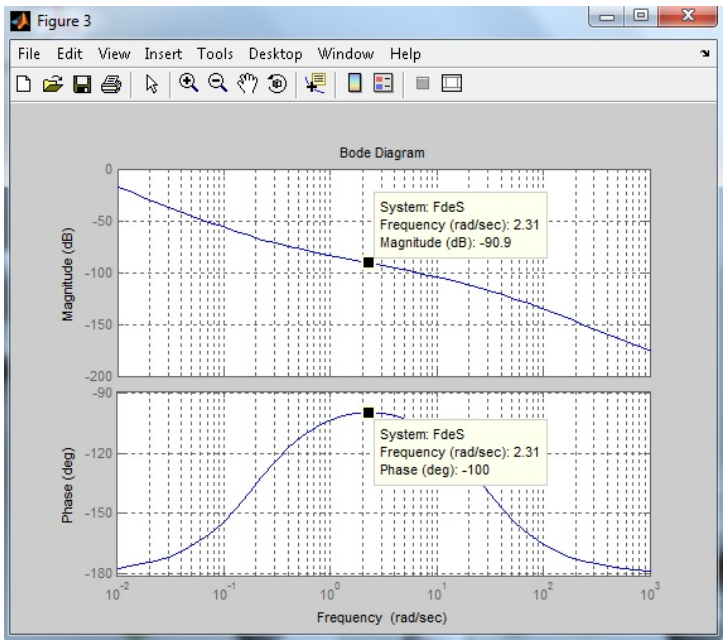

Fig. 32 Diagrama de Bode del sistema incluyendo el PAF. Gráfica de arriba: diagrama de módulos; gráfica de abajo: diagrama de fases.

Vemos que en el diagrama de fases aparece un pico cuyo valor máximo, de -100º, se da para la frecuencia de corte que hemos establecido nosotros durante su parametrización. Como podemos ver, el valor del módulo para dicha frecuencia no es nulo, sino que vale -91dB aprox, por lo que tenemos que realizar el último paso para obtener definitivamente el regulador.

4. <u>Ajuste del diagrama de módulos.</u> Ahora lo que tenemos que hacer es añadir una ganancia que haga que para la ω_c^d se dé, efectivamente, un margen de fase de 80º. En nuestro caso, lo que tenemos que hacer es subir el diagrama de módulos, es decir, añadir una ganancia positiva.

$$|G(j\omega)|_{\omega=w_c^d} = 0dB \iff |G(j\omega)|_{\omega=w_c^d} + 20\log_{10} K_r = 0dB$$

$$|G(j\omega)|_{\omega=w_c^d} + 20\log_{10} K_r = -91 + 20\log_{10} K_r = 0dB$$

$$K_r = 10^{91/20} = 35481$$

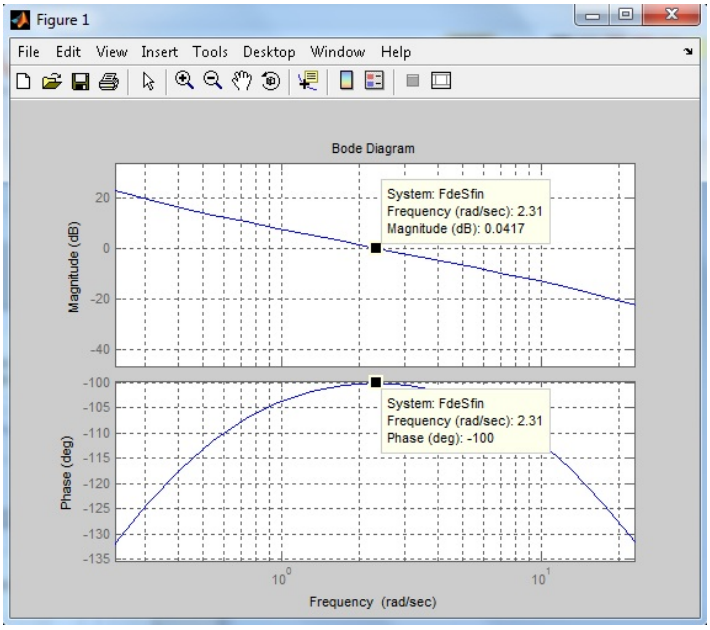

Fig. 33 Diagrama de Bode del sistema incluyendo el PAF ajustado. Gráfica de arriba: diagrama de módulos; gráfica de abajo: diagrama de fases.

Finalmente, en el diagrama de módulos del sistema con el regulador proporcional de avance de fase obtenido, tras el ajuste del diagrama de módulos, se puede apreciar que realmente el margen de fase es de 80º.

Tras todo esto, sólo queda estudiar el comportamiento en bucle cerrado ante un escalón. En la gráfica se ve que hay una cierta sobreoscilación, siendo el tiempo de respuesta mayor que el establecido por nosotros.

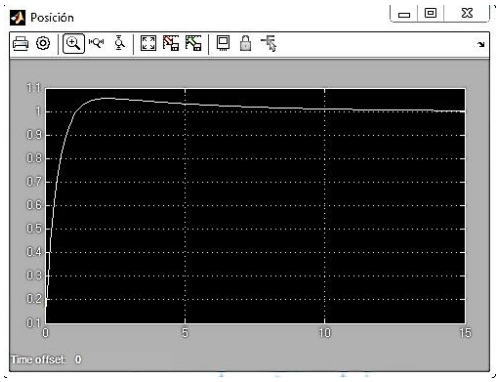

Fig. 34 Respuesta ante un escalón del drone con la inclusión del PAF en bucle cerrado.

4.1.5.- Elección del regulador.

Tras analizar los diferentes tipos de reguladores propuestos, se ve claramente que el regulador que mejores prestaciones da es el servopropulsor. El tiempo de respuesta del

sistema es de 1s, el que hemos impuesto nosotros, y sin que haya sobreoscilaciones ni se produzcan errores de ejecución en el desarrollo de la simulación. Del resto de reguladores, el único que se le acerca a ese tiempo de respuesta es el PAF y, aun así, lo hace con un sobrepasamiento de la señal de consigna.

Si tomamos en consideración las acciones iniciales, los cuatro aportan valores muy elevados de estas, sin contar la acción que introducimos nosotros con la realimentación de w_0^2. Sin embargo, dado que no se va a implementar de forma física, no es un parámetro importante a la hora de elegir el corrector adecuado. Esto mismo es aplicable a los valores de las ganancias.

Sin embargo, para complementar el regulador elegido, vamos a implementar una estructura de control por prealimentación denominado “servomecanismo”. Lo que conseguimos con estructura es que la salida siga lo más rápidamente posible las variaciones de la entrada cuando éstas se produzcan, al adicionarse una acción dependiente de éstas a la del regulador principal.

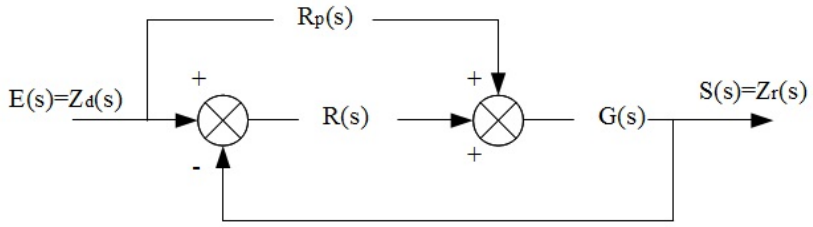

Fig. 35 Diagrama de bloques de un servomecanismo en bucle cerrado. La entrada, Z_d, es el valor dado por el usuario.

Al introducir la prealimentación, la salida del sistema queda de la forma

$$S(s) = \frac{R(s)_p + R(s)}{1 + R(s)G(s)} * G(s)E(s) + \frac{1}{1 + R(s)G(s)} * P(s)$$

Esto influye al regulador presente en la prealimentación en la forma de que se debe cumplir que su función de transferencia sea la unidad.

$$\frac{R(s)_p + R(s)}{1 + R(s)G(s)} * G(s) = 1 \iff R(s)_p = \frac{1}{G(s)} \; ; G(s) = {}^{K}/_{s^2} \Rightarrow \frac{1}{G(s)} = \frac{s^2}{K}$$

Para no ralentizar la simulación en demasía, se procederá a calcular la primera y la segunda derivada fuera de simulink y, después, se transferirán los datos resultantes a éste. El proceso a seguir se explicará en el apartado 6.

4.2.- Control de la componente de orientación "Yaw".

Lo primero que debemos hacer es obtener la función de transferencia correspondiente a la entrada τ_{yaw}. Para ello, partimos del empuje que provoca un giro en el eje Z

$$\tau_{yaw} = c_Q * (w_2^2 + w_4^2 - w_1^2 - w_3^2) = c_Q * (2w_2^2 - 2w_1^2)$$

Ahora bien, si aplicamos empuje en dicha entrada cuando el quadrotor se encuentra en situación de sustentación, entonces queda de la forma

$$\tau_{yaw} = c_Q * \left[2(w_{yaw} - w_0)^2 - 2(w_{yaw} + w_0)^2\right] = 2 * c_Q * (-4 * w_{yaw} * w_0)$$

$$\tau_{yaw} = I_{zz} * \frac{d^2 yaw}{dt^2} \Rightarrow \tau_{yaw}(s) = I_{zz} * s^2 yaw(s)$$

Igualando los términos, la función de transferencia resultante es

$$\frac{\psi(s)}{w_{yaw}(s)} = \frac{-8 * w_0 * c_Q}{I_{zz} * s^2} = \frac{K}{s^2} = \frac{-0.004945}{s^2}$$

En este caso, aunque al principio pueda parecer que se trata de un control sobre una variable cuadrática, como ocurre en el apartado 4.1, en realidad es un control lineal sobre una variable lineal. Es decir, sobre "w". Tras este apunte, pasamos a obtener los coeficientes del servopropulsor, cuyo calculo está ya explicado en el subapartado correspondiente del control de altitud, con las mismas especificaciones: ξ=1 y t_r=1.

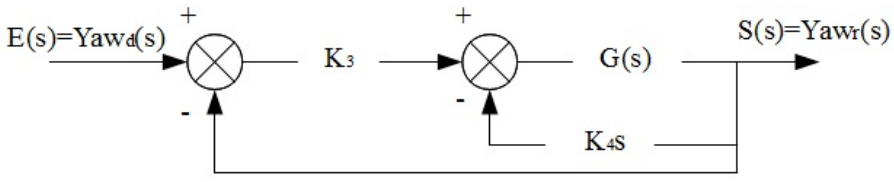

Fig. 36 Diagrama de bloques del regulador servopropulsor, siendo Yaw_d el valor deseado por el usuario.

$$K_3 = \frac{\omega_n^2}{K} = \frac{4.75^2}{0.004945} = -4.5627 * 10^3$$

$$K_4 = \frac{2\omega_n}{K} = \frac{2\omega_n}{0.004945} = -1.9211 * 10^3$$

Una vez calculados, pasamos a implementarlo en la simulación y vemos la respuesta del sistema, el correspondiente a esta entrada, para un escalón de 0.1 que se aplicará cuando el aparato esté a una altura constante. El valor del escalón es muy reducido dado que en operación normal no se le exigirá cambios de "yaw" bruscos, sino que se le introducirá una consigna más suave.

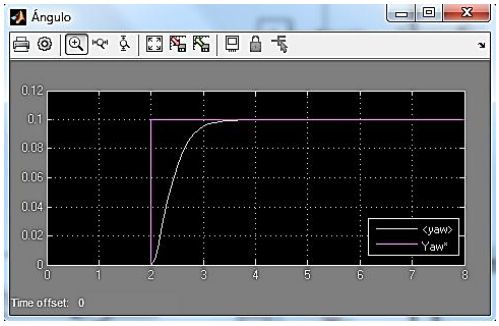

Fig. 37 Respuesta del sistema correspondiente a τ_{yaw} ante un escalón de 0.1 en dicha entrada, en situación de sustentación.

4.3.- Control de las componentes de orientación "pitch"/"roll" y posición en el plano horizontal.

Para la obtención de un regulador que controle los desplazamientos en horizontal, hay que realizar en primera instancia un buen control de los ángulos "pitch" y "roll", los cuales se verán modificados durante dichas traslaciones. Dichas modificaciones en los valores de éstos provocaran una disminución del empuje en vertical al cambiar el plano horizontal del quadrotor, de forma que éste sea T_z=T*cos(θ) si se produce un movimiento en el eje X o T_z=T*cos(ϕ) si es en el eje Y. Para que el valor del empuje en Z se mantenga lo más constante posible, estos ángulos deben ser muy pequeños de manera que el coseno de éstos sea prácticamente la unidad.

Los empujes que permiten cambiar la posición en el plano XY son:

$$\tau_{roll} = dc_T(w_2^2 - w_4^2)$$

$$\tau_{pitch} = dc_T(w_3^2 - w_1^2)$$

siendo d la distancia del rotor al centro de gravedad.

El primero de los empujes se corresponde con el necesario para provocar un giro sobre el eje X, que dará lugar a un desplazamiento en Y, y el segundo con el necesario para dar lugar a un giro sobre este último, que conllevará a una traslación en X. El parámetro d es la distancia del rotor al centro de gravedad.

Para el control de X/Y hay que tener en cuenta los marcos de referencia de los ejes utilizados. Esto se debe a que nosotros introducimos los valores de estas dos componentes de la posición desde el sistema de referencia inercial "world", mientras que para realizar el control necesitamos los correspondientes al sistema del aparato, el cual es móvil pero solidario a los ejes X e Y inerciales. Es decir, convertiremos estos puntos del sistema "world" a puntos del sistema "drone".

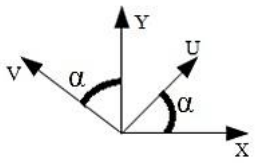

Fig. 38 Rotación en el caso de dos sistemas de referencia, siendo OUV solidario a OXY.

4.3.1- Control de roll/pitch.

Suponiendo que el aparato se encuentra en situación de sustentación, es decir, que los motores giran a una velocidad w_0 y que no se produce giro alguno sobre el eje X, no hay variación de "yaw", si introducimos una consigna en τ_{ptich} o en τ_{roll}, las cuales denominaremos w_p y w_r, las velocidades angulares quedan de la forma

$$w_2 = w_r - w_0 \; ; \; w_4 = -w_r - w_0$$

$$w_1 = w_p + w_0 \; ; \; w_3 = -w_p + w_0$$

Las expresiones de los empujes para realizar giros en los ejes horizontales son:

$$\tau_{roll} = -4dc_T w_0 w_r = I_{xx}\frac{d^2 roll}{dt^2}$$

$$\tau_{\text{pitch}} = -4dc_T w_0 w_p = I_{yy}\frac{d^2 pitch}{dt^2}$$

A partir de aquí, resulta sencillo sacar la función de transferencia entre los ángulos y los incrementos o decrementos de w_p y w_r.

$$\frac{\phi(s)}{w_r(s)} = \frac{-4dc_T w_0}{I_{xx} * s^2} = \frac{K}{s^2}$$

$$\frac{\theta(s)}{w_r(s)} = \frac{-4dc_T w_0}{I_{xx} * s^2} = \frac{K}{s^2}$$

Siendo K:

$$K = \frac{-4dc_T w_0}{I_{xx}} = -0.1751$$

Una vez obtenidas, el siguiente paso es el cálculo del regulador de doble lazo, o servopropulsor, para su control. Al darse que I_{xx}=I_{yy}, sólo tenemos que hacer una vez los cálculos dado que los valores de las constantes serán los mismos en ambos casos.

Como el procedimiento es el mismo que el realizado en el apartado 4.1.2, en vez de explayarnos con la explicación pasamos a la resolución del valor de K_1 y K_2 para este caso. Las especificaciones son las mismas que las exigidas en el apartado anteriormente citado.

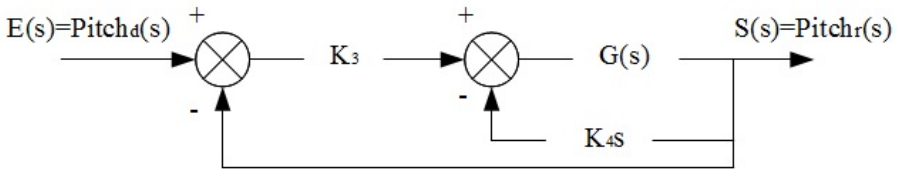

Fig. 39 Diagrama de bloques del regulador con realimentación de doble lazo para "Pitch" de la orientación, cuyo valor de entrada se corresponde con la salida del control de posición (no establecido por el usuario).

$$K_3 = \frac{\omega_n^2}{K} = \frac{4.75^2}{0.1751} = -128.8674$$

$$K_4 = \frac{2\omega_n}{K} = \frac{9.5}{0.1751} = -54.26$$

La peculiaridad que presenta este control es que la entrada viene del control X/Y y no de una consigna correspondiente a la deseada por el usuario. Por lo que el bucle de control explicado en este apartado (4.3) es un poco más complejo que los anteriormente explicados.

4.3.2.- Control de X/Y.

Como en los casos anteriores, el regulador elegido es el regulador con doble lazo de realimentación al que se añadirá una prealimentación de consigna, la cual será también, como en el control de altitud, doblemente derivativa. Ésta se calculara fuera de la simulación para no sobrecargar simulink y evitar que se ralentice la ejecución de la simulación.

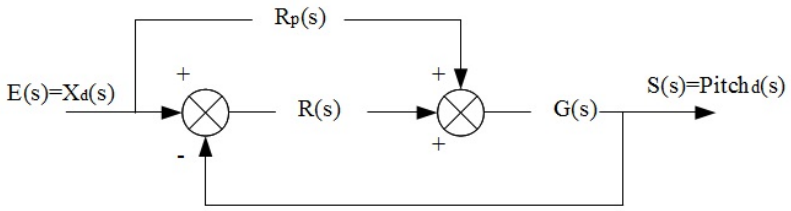

Fig. 40 Diagrama de bloques de la estructura servomecanismo en bucle cerrado, siendo la X_d el valor que desea el usuario y $Pitch_d$ la salida de dicho control.

Sin embargo, al utilizar un modelo muy simple de la dinámica del drone para su cálculo, pueden aparecer ciertos comportamientos oscilantes debido a determinados efectos aerodinámicos, principalmente el llamado "rotor flapping[14]", que hacen éste control no esté tan bien ajustado como los anteriores.

En este caso, tenemos que distinguir dos sistemas de referencia: el correspondiente a la referencia absoluta y el correspondiente con la referencia "body frame", es decir, la del cuerpo del aparato. Esto se debe a que las rotaciones que se producen en yaw hacen que las coordenadas absolutas no sean las mismas respecto al cuerpo del aparato.

Sin embargo, los sensores incluidos en el mismo nos dan la posición "absoluta". Por lo que tenemos que obtener el error en el primer sistema de referencia y, mediante rotación en Z, pasarlo a las coordenadas que maneja el quadrotor.

[14] Efecto explicado en el artículo "Multirotor Aerial Vehicles: Modeling, Estimation and Control of Quadrotor". Mahony, R., Kumar, V., Corke, P. IEEE ROBOTICS & AUTOMATION MAGAZINE (SEPTEMBER 2012, 20-3).

Una vez explicado esto, pasamos al cálculo de las constantes propiamente dicho. Las funciones de transferencia que tenemos de las coordenadas X/Y desde la referencia del cuerpo del quadrotor con los ángulos “roll” y “pitch”, suponiendo que ψ=0,son:

$$\frac{X'(s)}{\theta(s)} = \frac{g}{s^2}$$

$$\frac{Y'(s)}{\phi(s)} = \frac{-g}{s^2}$$

Sabemos que la relación de estos ángulos con las coordenadas en el plano horizontal, respecto de la referencia absoluta, son:

$$\ddot{X} = g[\cos(\psi)\,\theta + \sin(\psi)\,\phi)]$$

$$\ddot{Y} = g[-\cos(\psi)\,\phi + \sin(\psi)\,\theta)]$$

Para la obtención del regulador con realimentación de doble lazo utilizamos las primeras mientras que las segundas las utilizaremos para la prealimentación. En este caso, el tiempo de respuesta se establece en 3s.

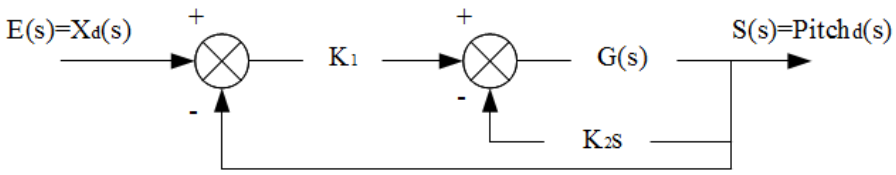

Fig. 41 Diagrama de bloques del servopropulsor para x, para yaw<90º. X_d es la consigna introducida por el usuario y $Pitch_d$ la salida de dicho control.

Haciendo los cálculos explicados en el apartado 4.1.2, los cuales haremos para el caso de X y trasladaremos al caso de Y, los valores de las constantes son:

$$K_1 = \frac{\omega_n^2}{g} = \frac{1.5833}{9.81} = 0.25555$$

$$K_2 = \frac{2\omega_n}{g} = \frac{2 * 1.5833}{9.81} = 0.3228$$

Respecto a la prealimentación, una vez obtenidas las segundas derivadas, serán transferidas a simulink y transformadas al espacio de trabajo del robot.

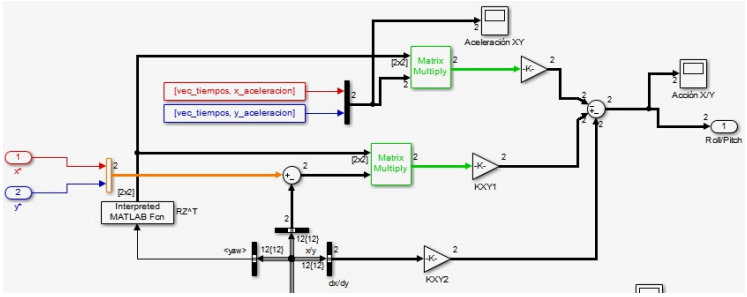

Fig. 42 Esquema final del control en x/y.

4.4. Ajuste final del control.

Dado que se han obtenido los valores de las constantes de los servopropulsores con distintos tiempos de respuesta, debemos proceder a un ajuste del control para que todas las entradas tarden el mismo tiempo en llegar al valor de la consigna.

Debido a que el modelo utilizado para obtener el regulador en X/Y es mucho más simple de lo que en realidad es, ajustamos los tiempos de respuesta de los otros lazos al de dicho lazo.

Las constantes para un tiempo de respuesta de 3s quedan:

- Control de altitud: K_1=1’8943*10^5; K_2=2’39283*10^5.
- Control de orientación en Z: K_1=-506’9655; K_2=-640’3775.

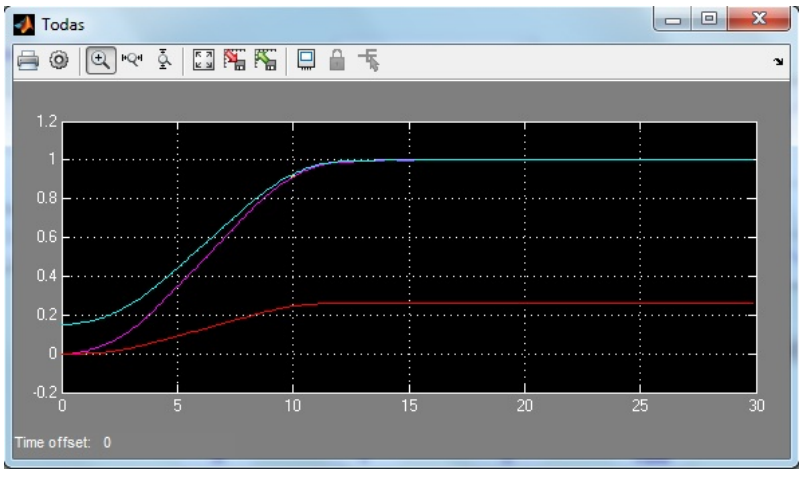

Fig. 43 Respuesta del sistema: punto introducido por el usuario X=1,Y=1,Z=1 y Yaw=15º (en la gráfica en radianes) con el generador de trayectorias y las prealimentaciones ya implementadas.

5.- Generador de trayectoria.

Una vez transferidas las coordenadas de introducidas por el usuario a la interfaz gráfica, se procede a calcular la trayectoria a seguir para alcanzar dicha posición. En nuestro caso, el tipo de trayectoria será de tipo trapezoidal para la velocidad.

Este tipo de trayectoria permite que los cambios de valor de ésta se den de forma menos brusca que si tuviera forma de escalón al darse que, en dichos tramos, la evolución temporal de la trayectoria se corresponde con un polinomio de segundo orden. Dicho de otra manera, es una trayectoria derivable tres veces, estando la aceleración compuesta por siete tramos.

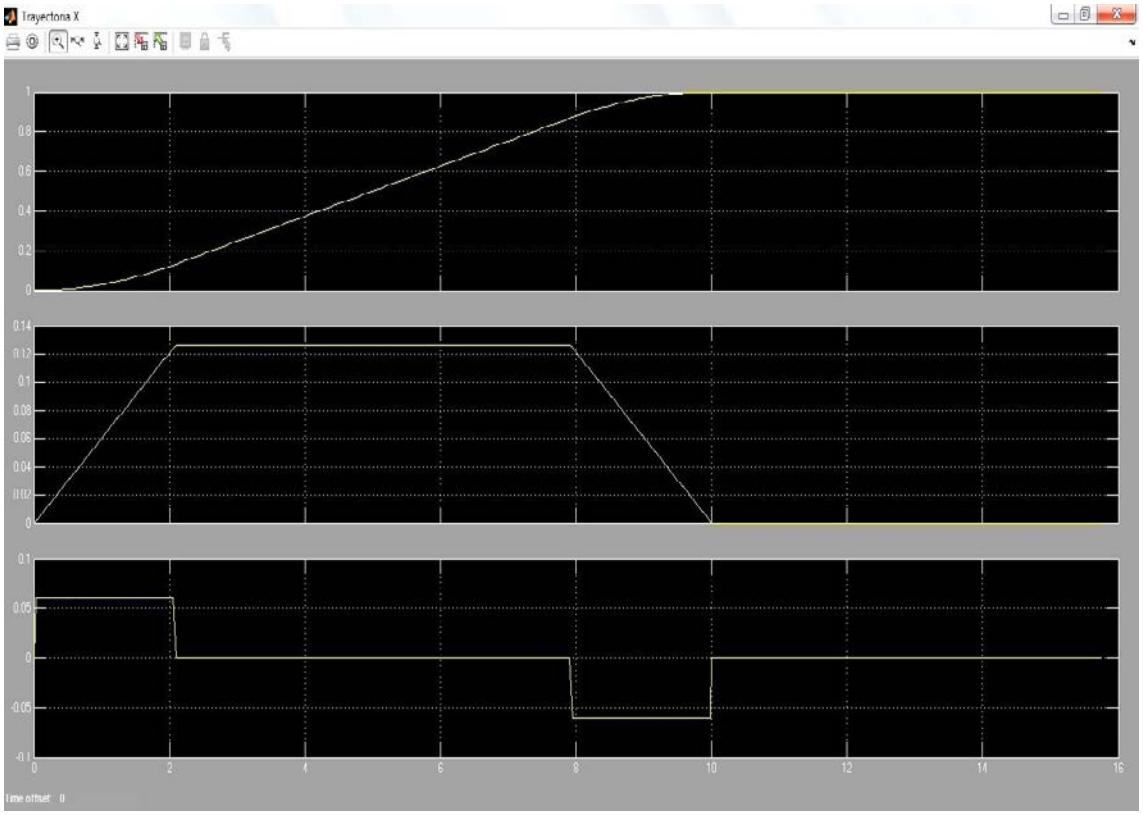

Fig. 44 Posición, velocidad y aceleración de entrada.

Realmente, en este tipo de trayectoria, la aceleración tendría forma de escalón. Sin embargo, en la simulación no resulta así a que en el bloque de introducción de la segunda derivada en simulink también se activa la casilla correspondiente a “interpolación de datos” de forma que, cuando no disponga de datos, el programa “rellene” esos huecos. Esto hará que los escalones que realmente conforman la gráfica

de la aceleración, tengan una forma más parecida a un trapecio precisamente por la interpolación lineal que éste realiza.

Como se procederá a explicar en apartados siguientes (6.2.7), los datos se recogen de los “edit text” de la interfaz cuando el usuario pulsa el botón de “Generar trayectoria”. Una vez obtenidos, se procede a realizar el cálculo de la trayectoria.

```
T_rotacion_destino=eul2tr(yawusuario,0,0); %obtener matriz de rotación
de destino
% componer matriz: (R p)
%                   (0 1)
Tdestino=[T_rotacion_destino [xusuario,yusuario,zusuario]';[0,0,0,1]];
handles.Tdestino=Tdestino;

Torigen=handles.Torigen;
%componer trayectoria
trayectoria_trf=ctrajTPZ(Torigen,Tdestino,vector_tiempos,aceleracionus
uario);

% obtener x, y,z
trayectoria_x=trayectoria_trf(:,13);
handles.trayectoria_x=trayectoria_x; % para que puedan ser utilizados
por
% otras funciones
% añadimos dato al final para evitar problemas con la interpolacion de
datos
trayectoria_x=[trayectoria_x;xusuario];
assignin('base','x_trayectoria',trayectoria_x);
trayectoria_y=trayectoria_trf(:,14);
handles.trayectoria_y=trayectoria_y; % para que puedan ser utilizados
por
% otras funciones
% añadimos dato al final para evitar problemas con la interpolacion de
datos
trayectoria_y=[trayectoria_y;yusuario];
assignin('base','y_trayectoria',trayectoria_y);
trayectoria_z=trayectoria_trf(:,15);
handles.trayectoria_z=trayectoria_z; % para que puedan ser utilizados
por
% otras funciones
% añadimos dato al final para evitar problemas con la interpolacion de
datos
trayectoria_z=[trayectoria_z;zusuario];
assignin('base','z_trayectoria',trayectoria_z);

% obtener yaw
vector_yaws=[];
[filas,columnas]=size(trayectoria_trf); % para saber cuantas filas y
% columnas tiene trayectoria_trf para el bucle for
for i=1:filas
     matriz=reshape(trayectoria_trf(i,:),4,4);
     angulo=tr2eul(matriz);
     yawi=angulo(1)+angulo(3);
     vector_yaws=[vector_yaws,yawi];
end
vector_yaws=[vector_yaws,yawusuario];
```

```
assignin('base','yaw_trayectoria',vector_yaws');

% los vectores antes de añadir un punto mas para la interpolacion de
datos
% y que serán utilizados para el caculo de las velocidades y
aceleraciones
posicionx=handles.trayectoria_x;
posiciony=handles.trayectoria_y;
posicionz=handles.trayectoria_z;
[filas,columnas]=size(posicionx); % para saber cuantas filas y
% tiene trayectoria_trf para el bucle for
% los tres tienen la misma longitud. Solo se calcula para uno.

% bucle x.
%calculo velocidad de x
for i=2:filas;
    velocidadxi=(posicionx(i)-posicionx(i-1))/0.01;
    velocidadx(i-1)=[velocidadxi];
end
% calculo aceleracion de x
for i=2:filas-1;
    aceleracionxi=(velocidadx(i)-velocidadx(i-1))/0.01;
    aceleracionx(i-1)=[aceleracionxi];
end
aceleracionx=[0,aceleracionx,0,0]; % añadimos un cero al principio
para que
% el primer valor sea nulo y dos al final para completar el vector y
evitar
% los problemas de interpolacion de datos
assignin('base','x_aceleracion',aceleracionx'); % a workspace
velocidadx=[0,velocidadx,0]; % lo mismo que con la aceleracion
assignin('base','x_velocidad',velocidadx'); % a workspace

% bucle y.
% calculo velocidad de y
for i=2:filas;
    velocidadyi=(posiciony(i)-posiciony(i-1))/0.01;
    velocidady(i-1)=[velocidadyi];
end
% calculo aceleracion de y
for i=2:filas-1;
    aceleracionyi=(velocidady(i)-velocidady(i-1))/0.01;
    aceleraciony(i-1)=[aceleracionyi];
end
aceleraciony=[0,aceleraciony,0,0];% añadimos un cero al principio para
que
% el primer valor sea nulo y dos al final para completar el vector y
evitar
% los problemas de interpolacion de datos
assignin('base','y_aceleracion',aceleraciony'); % a workspace
velocidady=[0,velocidady,0]; % lo mismo que con la aceleracion
assignin('base','y_velocidad',velocidady'); % a workspace

% bucle z.
% calculo velocidad de z
for i=2:filas;
    velocidadzi=(posicionz(i)-posicionz(i-1))/0.01;
    velocidadz(i-1)=[velocidadzi];
end
% calculo aceleracion de z
```

```
for i=2:filas-1;
    aceleracionzi=(velocidadz(i)-velocidadz(i-1))/0.01;
    aceleracionz(i-1)=[aceleracionzi];
end
aceleracionz=[0,aceleracionz,0,0];% añadimos un cero al principio para
que
% el primer valor sea nulo y dos al final para completar el vector y
evitar
% los problemas de interpolacion de datos
assignin('base','z_aceleracion',aceleracionz'); % a workspace
velocidadz=[0,velocidadz,0]; % lo mismo que con la aceleracion
assignin('base','z_velocidad',velocidadz'); % a workspace

guidata(hObject, handles); % actualiza los valores de las variables
```

Lo primero es, con el valor de yaw, obtener la matriz de rotación correspondiente al punto de destino y, una vez calculada, se obtiene la matriz de transformación de dicho punto componiéndola con la resultante de “eul2tr” y el vector columna correspondiente a la posición. Después se procede a guardar dichos datos en “handles.Tdestino”, para poder ser utilizados en otra función (6.2.2).

Tras este paso, se establece el valor de “Torigen”, siendo su valor “handles.Torigen” (6.1 y 6.2.2), el cual tomará un valor en función de si es la primera ejecución o ya se ha realizado una previamente. Una vez obtenidas ambas transformaciones, junto con el vector de tiempos y el valor de la aceleración, se llama a la función “ctrajTPZ”. Esta función es la que compone la trayectoria cartesiana en una matriz de nx16, almacenándose dichos datos en la variable “trayectoria_trf”; siendo 16 el número correspondiente a la cantidad de elementos que hay en una matriz 4x4.

Una vez obtenida ésta, vamos sacando los datos, primero, correspondientes a las tres coordenadas cartesianas, x,y,z. Tras esto, procedemos al cálculo de la componente de giro sobre z. Debido a que en este caso, respecto a los ángulos de Euler nos encontramos siempre en situación de singularidad[15], el valor de “yaw” se divide entre el primer y el tercer ángulo de Euler, de ahí que realicemos un bucle “for” que va desde uno hasta el número de filas existentes en “trayectoria_trf” haciendo la suma de estos dos ángulos y almacenándolos dentro de un vector.

Dichos vectores de almacenamiento de datos son enviados después al workspace de forma que simulink pueda utilizarlos a la hora de realizar la simulación. Para utilizarlos como señal de entrada, enviamos dichos datos al lugar de trabajo general en formato

[15] El segundo y el tercer ángulo son de valor nulo.

columna (en el caso del vector de yaw) o, directamente, tal cual se obtienen de "trayectoria_trf". Antes de esto, se guardan en variables de tipo "handles" para que puedan ser utilizados por otras funciones de la interfaz (6.2.7).

Esta función también se encarga del cálculo de la primera y segunda derivada de las entradas de forma que no sobrecargamos simulink y provocamos, con ello, una ralentización de la ejecución de la simulación al realizar las prealimentaciones doblemente derivativas de consigna. Al igual que para el caso de "yaw", se obtiene el número de filas que tiene uno de los vectores de posición y se utiliza ese valor para los bucles "for" donde se harán las diferencias hacía atrás entre los valores.

Finalmente, se procede a guardar todas las variables "handles" utilizadas.

Como curiosidad, al ser simulaciones infinitas, se ha tenido que añadir, tras el cálculo de los vectores, al final de los mismos, el último valor obtenido de forma que, al tener activada la casilla de interpolación de los datos del bloque de simulink, la trayectoria se mantenga constante para tiempos mayores a los establecidos por el usuario sin que se den incrementos lineales de ésta debido a dicha "orden".

En los casos de las velocidades y las aceleraciones, al ser vectores con una longitud menor que los de posición, se han rellenado con ceros al principio y, en el caso de las aceleraciones, también al final. Además, también se evita así la problemática de la interpolación de datos realizada por simulink que se ha mencionado anteriormente.

5.1.- Código completo del botón de "Generate trajectory".

En este apartado se muestra el código completo correspondiente al generador de trayectorias, el cual se explica tanto en 6.2.7 como 5.

```
% --- Executes on button press in generartrayectoria.
function generartrayectoria_Callback(hObject, eventdata, handles)
% hObject    handle to generartrayectoria (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
xusuario=get(handles.xdeseada,'String');
xusuario=str2num(xusuario); % pasar a número

yusuario=get(handles.ydeseada,'String');
yusuario=str2num(yusuario);% pasar a número

zusuario=get(handles.zdeseada,'String');
zusuario=str2double(zusuario);
```

```
%limitar z mín
if zusuario<0.2
    set(handles.MensajeError, 'String','ERROR. Zmín=0.2');
 else
    set(handles.MensajeError, 'String','');
end

yawusuario=get(handles.yawdeseada,'String');
yawusuario=str2num(yawusuario)*pi/180; %pasar a número y despues a
radianes.

duracionusuario=get(handles.duracion,'String');
duracionusuario=str2num(duracionusuario); %pasar a número
vector_tiempos=[0:0.01:duracionusuario]; % crear vector de tiempos
vector_tiempos2=[0:0.01:duracionusuario+0.01];
assignin('base','vec_tiempos',vector_tiempos2');%transferir

aceleracionusuario=get(handles.aceleracion,'String');
aceleracionusuario=str2num(aceleracionusuario); %pasar a número

T_rotacion_destino=eul2tr(yawusuario,0,0); %obtener matriz de rotación
de destino
% componer matriz: (R p)
%                  (0 1)
Tdestino=[T_rotacion_destino [xusuario,yusuario,zusuario]';[0,0,0,1]];
handles.Tdestino=Tdestino;

Torigen=handles.Torigen;
%componer trayectoria
trayectoria_trf=ctrajTPZ(Torigen,Tdestino,vector_tiempos,aceleracionus
uario);

% obtener x, y,z
trayectoria_x=trayectoria_trf(:,13);
handles.trayectoria_x=trayectoria_x; % para que puedan ser utilizados
por
% otras funciones
% añadimos dato al final para evitar problemas con la interpolacion de
datos
trayectoria_x=[trayectoria_x;xusuario];
assignin('base','x_trayectoria',trayectoria_x);
trayectoria_y=trayectoria_trf(:,14);
handles.trayectoria_y=trayectoria_y; % para que puedan ser utilizados
por
% otras funciones
% añadimos dato al final para evitar problemas con la interpolacion de
datos
trayectoria_y=[trayectoria_y;yusuario];
assignin('base','y_trayectoria',trayectoria_y);
trayectoria_z=trayectoria_trf(:,15);
handles.trayectoria_z=trayectoria_z; % para que puedan ser utilizados
por
% otras funciones
% añadimos dato al final para evitar problemas con la interpolacion de
datos
trayectoria_z=[trayectoria_z;zusuario];
assignin('base','z_trayectoria',trayectoria_z);

% obtener yaw
```

```
vector_yaws=[];
[filas,columnas]=size(trayectoria_trf); % para saber cuantas filas y
% columnas tiene trayectoria_trf para el bucle for
for i=1:filas
     matriz=reshape(trayectoria_trf(i,:),4,4);
     angulo=tr2eul(matriz);
     yawi=angulo(1)+angulo(3);
     vector_yaws=[vector_yaws,yawi];
end
vector_yaws=[vector_yaws,yawusuario];
assignin('base','yaw_trayectoria',vector_yaws');

% los vectores antes de añadir un punto mas para la interpolacion de
datos
% y que serán utilizados para el caculo de las velocidades y
aceleraciones
posicionx=handles.trayectoria_x;
posiciony=handles.trayectoria_y;
posicionz=handles.trayectoria_z;

% bucle x.
%calculo velocidad de x
for i=2:1001;
    velocidadxi=(posicionx(i)-posicionx(i-1))/0.01;
    velocidadx(i-1)=[velocidadxi];
end
% calculo aceleracion de x
for i=2:1000;
    aceleracionxi=(velocidadx(i)-velocidadx(i-1))/0.01;
    aceleracionx(i-1)=[aceleracionxi];
end
aceleracionx=[0,aceleracionx,0,0]; % añadimos un cero al principio
para que
% el primer valor sea nulo y dos al final para completar el vector y
evitar
% los problemas de interpolacion de datos
assignin('base','x_aceleracion',aceleracionx'); % a workspace
velocidadx=[0,velocidadx,0]; % lo mismo que con la aceleracion
assignin('base','x_velocidad',velocidadx'); % a workspace

% bucle y.
% calculo velocidad de y
for i=2:1001;
    velocidadyi=(posiciony(i)-posiciony(i-1))/0.01;
    velocidady(i-1)=[velocidadyi];
end
% calculo aceleracion de y
for i=2:1000;
    aceleracionyi=(velocidady(i)-velocidady(i-1))/0.01;
    aceleraciony(i-1)=[aceleracionyi];
end
aceleraciony=[0,aceleraciony,0,0];% añadimos un cero al principio para
que
% el primer valor sea nulo y dos al final para completar el vector y
evitar
% los problemas de interpolacion de datos
assignin('base','y_aceleracion',aceleraciony'); % a workspace
velocidady=[0,velocidady,0]; % lo mismo que con la aceleracion
assignin('base','y_velocidad',velocidady'); % a workspace
```

```
% bucle z.
% calculo velocidad de z
for i=2:1001;
    velocidadzi=(posicionz(i)-posicionz(i-1))/0.01;
    velocidadz(i-1)=[velocidadzi];
end
% calculo aceleracion de z
for i=2:1000;
    aceleracionzi=(velocidadz(i)-velocidadz(i-1))/0.01;
    aceleracionz(i-1)=[aceleracionzi];
end
aceleracionz=[0,aceleracionz,0,0];% añadimos un cero al principio para
que
% el primer valor sea nulo y dos al final para completar el vector y
evitar
% los problemas de interpolacion de datos
assignin('base','z_aceleracion',aceleracionz'); % a workspace
velocidady=[0,velocidady,0]; % lo mismo que con la aceleracion
assignin('base','z_velocidad',velocidadz'); % a workspace

guidata(hObject, handles); % actualiza los valores de las variables
```

6.- Interfaz gráfica de usuario.

Una vez terminado el diseño del sistema de control de trayectoria, pasamos a crear una interfaz de operación para un posible usuario que permita una fácil supervisión de la actuación del quadrotor, sin necesidad de tener amplios conocimientos sobre su dinámica ni del control de situación y de utilizar la pantalla de simulación. Para ello, seguimos utilizando el programa MATLAB pero la extensión GUI[16] para realizar dicha interfaz.

En este apartado se tratará de explicar el código del programa relevante a la hora de obtener las gráficas, dado que al colocar bloques tales como “axes” o “edit text” en el archivo “.Fig” se crean funciones en el archivo “.m” de creación de creación de dichos bloques, las cuales no tienen mayor utilidad que el hecho de poder utilizar dichos bloques para escribir textos o dibujar gráficas en ellos dependiendo, por ejemplo, de la opción elegida en un menú.

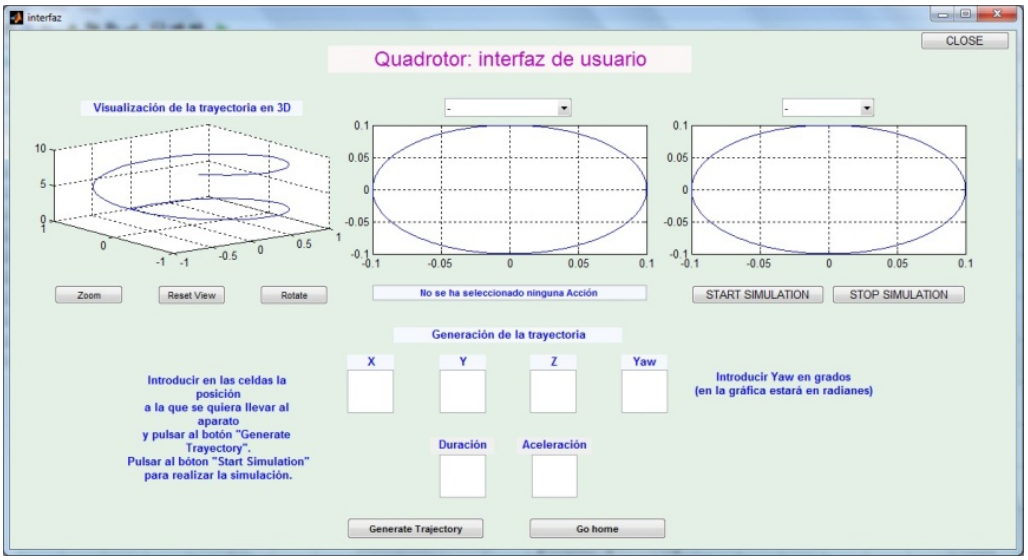

Fig. 45 Interfaz gráfica.

6.1.- Inicio de la interfaz y de la simulación.

Al abrir la interfaz, también se abre la simulación. Esto se consigue creando una función que haga dicha tarea y que sea llamada desde la función de inicio de la GUI. Tras esto, se carga el archivo “puma560”, el cual sólo sirve para poder utilizar las funciones empleadas en el cálculo de la trayectoria.

[16] Graphical Users Interface.

```
% --- Executes just before interfaz is made visible.
function interfaz_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to Fig.ure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to interfaz (see VARARGIN)

model_open(handles)
puma560; % cargar modelo puma.
%Dibuja un circulo en dos de los gráficos hasta que haya datos.
dibujarcirculoAccionesGraf(handles)
dibujarcirculoCoordenadasGraf(handles)
% Dibujo en 3D la gráfica de la trayectoria
diagramaen3d(handles)

set(handles.s, 'String','No se ha seleccionado ninguna Acción');

%establecer condiciones iniciales en simulink
z0 = [0 0 -0.15];             %   z0      Position initial conditions
1x3
n0 = [0 0 0];               %   n0      Ang. position initial
conditions    1x3
v0 = [0 0 0];               %   v0      Velocity Initial conditions
1x3
o0 = [0 0 0];               %   o0      Ang. velocity initial
conditions    1x3
init = [z0 [n0 v0 o0]];
assignin('base','xInitial',init);
handles.Torigen=[eye(3,3) [0 0 0.15]';0,0,0,1];

% Choose default command line output for interfaz
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

La función que abre el archivo de simulink es “model_open(handles)”. Cuyo código es:

```
% Función para abrir el archivo de simulink
function model_open(handles)
% Make sure the diagram is still open
  if  isempty(find_system('Name','quadcopterscontrolfinalcompleto')),
    open_system('quadcopterscontrolfinalcompleto');

  end;
```

Básicamente, lo que hace es: encuentra el sistema y si este no tiene entradas o salidas (“isempty devuelve el valor “true”) ábrelo.

Las otras funciones a las que se llama desde la función de inicio son “dibujarcirculoAccionesGraf(handes)”, ”dibujarcirculoCoordenadasGraf(handles)” y “diagramaen3D(handles)” que lo que hacen es dibujar un circulo, las dos primeras, en

su gráfica asignada y, la última, dibuja una trayectoria circular con sentido ascendente para mostrar como dibujo por defecto al abrir la interfaz.

Código correspondiente a la función “dibujarcirculoCoordenadasGraf(handles)”:

```
%función para que aparezca un dibujo por defecto en CoordenadasGraf
function dibujarcirculoCoordenadasGraf(handles)
%Dibuja un circulo en dos de los gráficos hasta que haya datos.
ang=0:0.01:2*pi;
r=0.1;
x=0;
y=0;
xp=r*cos(ang);
yp=r*sin(ang);
plot(handles.CoordenadasGraf,x+xp,y+yp);
grid(handles.CoordenadasGraf);

%función para que aparezca un dibujo por defecto en AccionesGraf
function dibujarcirculoAccionesGraf(handles)
%Dibuja un circulo en dos de los gráficos hasta que haya datos.
ang=0:0.01:2*pi;
r=0.1;
x=0;
y=0;
xp=r*cos(ang);
yp=r*sin(ang);
plot(handles.AccionesGraf,x+xp,y+yp);
grid(handles.AccionesGraf);
```

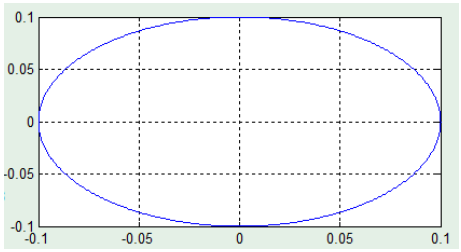

Fig. 46 Visualización del círculo en la gráfica de acciones.

```
function diagramaen3d(handles)
    t=0:0.1:10;
    plot3(handles.Trayectoria3D,sin(t),cos(t),t);
    grid(handles.Trayectoria3D);
```

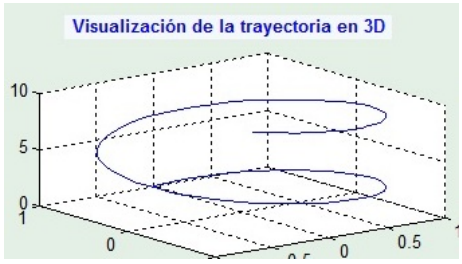

Fig. 47 Visualización dibujo por defecto en la gráfica de la trayectoria en 3D.

Tras todo esto, establece las condiciones iniciales de la simulación, poniendo el punto inicial en (0,0,-0.15) y el resto de valores a cero, que son los ángulos de giro sobre x, y, z y las velocidades tanto lineales como angulares del aparato. Una vez envidadas las condiciones al workspace, se construye la matriz de transformación de dicho punto.

6.2.- Botones de la interfaz gráfica.

6.2.1.- Botón de “START SIMULATION”.

Al pulsar el botón de “START SIMULATION”, se realiza una llamada a la función correspondiente. Esta función lo que hace es volver a abrir la simulación en caso de que ésta esté cerrada y enviar a dicho programa la orden de que comience a realizarla, si la altura introducida por el usuario es mayor que 0.2. En caso contrario, la simulación no empezará a ejecutarse.

Otra cosa que también hace es que, en caso de que se haya ejecutado otra simulación anteriormente, vuelve a poner todos los “pop-up” menús en su posición y los gráficos con sus dibujos por defecto.

Una vez hecho todo lo anterior, guarda todas las variables modificadas durante la ejecución de la función, hace invisibles los “edit text” de introducción de datos, para asegurar que el usuario no introduce datos nuevos durante la ejecución de la simulación, e inhabilita el botón de “STOP SIMULATION” un tiempo proporcional al establecido el usuario para la trayectoria.

```
% --- Executes on button press in Startbutton.
% Función para empezar a realizar la simulación.
function Startbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Startbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% asegurar que está abierto el archivo de simulink
model_open(handles)

% poner en posición por defecto los gráficos
set(handles.SelectorCoordenadas,'value',1);
SelectorCoordenadas_Callback(hObject, eventdata, handles)
set(handles.SelectorAcciones,'value',1);
SelectorAcciones_Callback(hObject, eventdata, handles);
diagramaen3d(handles)

zus=get(handles.zdeseada,'String'); % obtener valor de z introducido
```

```
% por el usuario
zus=str2double(zus);
% limite de z min.
if zus<0.2

set_param('quadcopterscontrolfinalcompleto','SimulationCommand','stop'
);
else

set_param('quadcopterscontrolfinalcompleto','SimulationCommand','start
');
set_param('quadcopterscontrolfinalcompleto','SimulationCommand','start
');

    % hacer invisibles los edit text de introducción de datos
    set(handles.xdeseada,'Visible','off');
    set(handles.ydeseada,'Visible','off');
    set(handles.zdeseada,'Visible','off');
    set(handles.yawdeseada,'Visible','off');
    set(handles.duracion,'Visible','off');
    set(handles.aceleracion,'Visible','off');
    set(handles.stopbutton,'Visible','off');
    pause(handles.duracionusuario*4);
    set(handles.stopbutton,'Visible','on');
end
guidata(hObject, handles); % actualiza los valores de las variables
% Fin función para realizar la simulación
```

6.2.2.- Botón de “STOP SIMULATION”.

En este caso, la función del botón envía una instrucción a la simulación para que se detenga. Al ser una simulación infinita, es la única manera para detener su ejecución y poder pasar a ver las gráficas. El botón permanece desactivado un tiempo proporcional al introducido por el usuario para la trayectoria (6.2.1).

Tras parar la ejecución de la simulación, llama a la función “dibujartrayectoria(handles)”, que describiremos en 6.3.3.

```
% Hint: place code in OpeningFcn to populate AccionesGraf
% --- Executes on button press in stopbutton.
function stopbutton_Callback(hObject, eventdata, handles)
% hObject    handle to stopbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set_param('quadcopterscontrolfinalcompleto','SimulationCommand','stop'
)
dibujartrayectoria(handles)

% recoger ultimos datos obtenidos en la simulación
x_nueva_ini=evalin('base','result(end,2)');
y_nueva_ini=evalin('base','result(end,3)');
z_nueva_ini=evalin('base','result(end,4)');

yaw_nueva_ini=evalin('base','result(end,5)');
```

```
dx_nueva_ini=evalin('base','result(end,8)');
dy_nueva_ini=evalin('base','result(end,9)');
dz_nueva_ini=evalin('base','result(end,10)');

dyaw_nueva_ini=evalin('base','result(end,11)');

z0 = [x_nueva_ini y_nueva_ini z_nueva_ini]; % z0 Position initial
conditions         1x3
n0 = [yaw_nueva_ini 0 0]; % n0 Ang. position initial conditions    1x3
v0 = [dx_nueva_ini dy_nueva_ini dz_nueva_ini]; %   v0      Velocity
Initial conditions         1x3
o0 = [dyaw_nueva_ini 0 0]; %   o0      Ang. velocity initial
conditions    1x3

init = [z0 n0 v0 o0];
assignin('base','xInitial',init);

handles.Torigen=handles.Tdestino;

% hace visibles los edit text de introducción de datos
set(handles.xdeseada,'Visible','on');
set(handles.ydeseada,'Visible','on');
set(handles.zdeseada,'Visible','on');
set(handles.yawdeseada,'Visible','on');
set(handles.duracion,'Visible','on');
set(handles.aceleracion,'Visible','on');

guidata(hObject, handles);
```

Tras dibujar la trayectoria, se recogen los últimos datos obtenidos de la simulación para establecerlos como nuevo punto de partida. Básicamente, lo que se hace es recoger los valores de x, y,z y yaw finales junto con el de las velocidades lineales y la velocidad angular entorno a z; el resto de valores los dejamos a cero[17].

Otra cosa que se hace es establecer la nueva matriz de transformación del origen, que será la anterior transformada de las coordenadas de destino establecidas por el usuario, de forma que se den simulaciones encadenadas.

Finalmente, se hacen visibles los “edit text” de introducción de datos.

6.2.3.- Botón de “CLOSE”.

Al pulsar este botón, lo que se hace es cerrar la interfaz, guardar los posibles cambios en el modelo de simulink, cerrar dicha ventana y las posibles ventanas que puedan quedarse abiertas tras cerrar la ventana de simulación. Antes de ello, pone los “pop-up” menús y las gráficas en su posición por defecto.

[17] Los giros sobreX e Y tienen carácter transitorio, por lo que sólo tienen un valor distinto de cero cuando se está desplazando en horizontal. Si tomamos ese punto como inicio en la siguiente simulación, este no se estará desplazando en horizontal.

En caso de que se hubiera hecho una rotación o un zoom sobre la representación de la trayectoria, se vuelve a visualizar como si se hubiera pulsado el botón de "Reset View".

```
% --- Executes on button press in closebutton.
function closebutton_Callback(hObject, eventdata, handles)
% hObject    handle to closebutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% poner en posición y dibujo por defecto
set(handles.SelectorCoordenadas,'value',1);
SelectorCoordenadas_Callback(hObject, eventdata, handles)
set(handles.SelectorAcciones,'value',1);
SelectorAcciones_Callback(hObject, eventdata, handles);
diagramaen3d(handles);
zoom(handles.Trayectoria3D,'off');
rotate3d(handles.Trayectoria3D, 'off');

close(interfaz) % Cierra la interfaz gráfica
save_system('quadcopterscontrolfinalcompleto'); % guarda el archivo de
simulink
close_system('quadcopterscontrolfinalcompleto'); % Cierra el archivo
de simulink.
close all
```

6.2.4.- Botón de "Zoom" (para la gráfica de representación de la trayectoria).

Este botón permite hacer un zoom sobre la gráfica en la que se representa la trayectoria e inhabilita el poder hacer rotaciones en caso de que se hubiera pulsado el botón de "Rotate". Sin embargo, en caso de que se hubiera hecho una rotación previamente, mantiene la vista sin modificar.

```
% --- Executes on button press in Zoom.
function Zoom_Callback(hObject, eventdata, handles)
% hObject    handle to Zoom (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
rotate3d(handles.Trayectoria3D, 'off');
zoom(handles.Trayectoria3D,'on');
```

Fig. 48 Visualización de "zoom" en la gráfica de trayectoria.

6.2.5.- Botón de “Rotate” (para la gráfica de representación de la trayectoria).

Al igual que en el botón anterior pero habilitando, en este caso, el poder realizar rotaciones en la gráfica en caso de que se pulse dicho botón. Sin embargo, en caso de que se hubiera hecho un zoom anteriormente, mantiene la vista sin modificar.

```
% --- Executes on button press in Rotate.
function Rotate_Callback(hObject, eventdata, handles)
% hObject    handle to Rotate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
rotate3d(handles.Trayectoria3D, 'on');
zoom(handles.Trayectoria3D,'off');
```

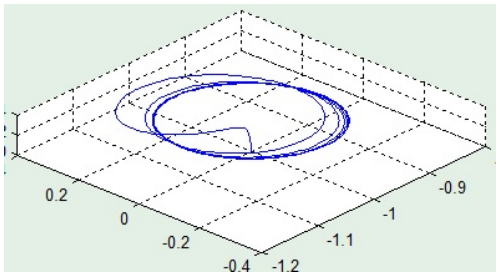

Fig. 49 Visualización de “rotate” en la gráfica de trayectoria.

6.2.6.- Botón de “Reset View” (para la gráfica de representación de la trayectoria).

Este botón permite volver a la apariencia inicial de la gráfica de representación de la trayectoria y desactiva la posibilidad de realizar las anteriores operaciones.

```
% --- Executes on button press in ResetView.
function ResetView_Callback(hObject, eventdata, handles)
% hObject    handle to ResetView (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
dibujartrayectoria(handles);
zoom(handles.Trayectoria3D,'off');
rotate3d(handles.Trayectoria3D, 'off');
```

6.2.7.- Botón de “Generate Trajectory”

Al pulsar el botón “Generate Trajectory” lo que se hace es recoger los datos introducidos en los “edit text” por el usuario, pasarlos a formato numérico y utilizarlos para calcular la trayectoria.

Dado que es dentro de esta función donde se realiza el cálculo de la trayectoria, se explicará en este apartado lo correspondiente a la adquisición de los datos. El cálculo de

la trayectoria se ha descrito en el apartado correspondiente al generador de trayectorias propiamente dicho (5).

```
% --- Executes on button press in generartrayectoria.
function generartrayectoria_Callback(hObject, eventdata, handles)
% hObject    handle to generartrayectoria (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
xusuario=get(handles.xdeseada,'String');
xusuario=str2num(xusuario); % pasar a número

yusuario=get(handles.ydeseada,'String');
yusuario=str2num(yusuario);% pasar a número

zusuario=get(handles.zdeseada,'String');
zusuario=str2double(zusuario);

%limitar z mín
if zusuario<0.2
    set(handles.MensajeError, 'String','ERROR. Zmín=0.2');
 else
    set(handles.MensajeError, 'String','');
end

yawusuario=get(handles.yawdeseada,'String');
yawusuario=str2num(yawusuario)*pi/180; %pasar a número y despues a
radianes.

duracionusuario=get(handles.duracion,'String');
duracionusuario=str2num(duracionusuario); %pasar a número
vector_tiempos=[0:0.01:duracionusuario]; % crear vector de tiempos
vector_tiempos2=[0:0.01:duracionusuario+0.01];
assignin('base','vec_tiempos',vector_tiempos2');%transferir

aceleracionusuario=get(handles.aceleracion,'String');
aceleracionusuario=str2num(aceleracionusuario); %pasar a número
```

Lo primero que hacemos es recoger los datos de x e y y pasarlos de string a número. Después procedemos a obtener el valor de z introducido por el usuario y se hace la comprobación de que éste es mayor que 0.2m con un bucle “if”: si es menor, aparece un mensaje de error indicando la altura mínima permitida (6.2.1).

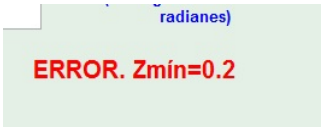

Fig. 50 Mensaje de error.

Tras esto, recogemos el valor de la “yaw” deseada y se pasa a radianes. Con esto terminamos la parte correspondiente a la obtención de la posición y orientación, respecto de Z elegida.

Los otros dos datos que se necesitan son el tiempo en el que se quiere hacer la traslación y la aceleración que el usuario desea. Con el valor de la duración, se crean dos vectores de tiempos, uno que se lanzará al workspace de MATLAB (el cual será utilizado por simulink) y otro que se utilizará para el cálculo de la trayectoria (5).

Respecto a la aceleración, puede darse un error que no aparezca en la pantalla de la interfaz gráfica sino que aparece en la ventana de comandos de MATLAB debido a que la aceleración y el tiempo requeridos no sean acordes el uno con el otro. Es decir, que se requiera una aceleración relativamente elevada y un tiempo relativamente bajo, de forma que la forma de la gráfica de la aceleración sea un triángulo y no un trapecio: acelera con pendiente casi infinita; por lo que se recomienda al usuario que no pierda de vista dicha pantalla.

6.2.8.- Botón “Go home”.

Lo que hace este botón es volver al punto inicial y poner en posición y dibujo por defecto los menús y las gráficas.

```
% --- Executes on button press in posicion_home.
function posicion_home_Callback(hObject, eventdata, handles)
% hObject    handle to posicion_home (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%establecer condiciones iniciales en simulink
z0 = [0 0 -0.15];             %   z0      Position initial conditions
1x3
n0 = [0 0 0];                 %   n0      Ang. position initial
conditions    1x3
v0 = [0 0 0];                 %   v0      Velocity Initial conditions
1x3
o0 = [0 0 0];                 %   o0      Ang. velocity initial
conditions    1x3
init = [z0 [n0 v0 o0]];
assignin('base','xInitial',init);
handles.Torigen=[eye(3,3) [0 0 0.15]';0,0,0,1];
set(handles.SelectorCoordenadas,'value',1);
SelectorCoordenadas_Callback(hObject, eventdata, handles)
set(handles.SelectorAcciones,'value',1);
SelectorAcciones_Callback(hObject, eventdata, handles);
zoom(handles.Trayectoria3D,'off');
rotate3d(handles.Trayectoria3D, 'off');
diagramaen3d(handles);
guidata(hObject, handles); % actualiza los valores de las variables
```

6.3.- Mostrar las gráficas.

Aquí se explicarán las líneas de código correspondientes al trazado de las evoluciones temporales de las coordenadas X, Y, Z y Yaw así como las de las acciones de los reguladores y las velocidades de los motores. También se procederán a explicar a las correspondientes a la visualización de la trayectoria en 3D.

6.3.1.- Visualización de las coordenadas X, Y, Z y de la componente de orientación Yaw.

El mostrar una u otra gráfica se realiza en función de la opción elegida en un menú de tipo “pop-up”, la cual se realiza con la instrucción “switch… case”.

Los casos enumerados son los que se corresponden con las posiciones que ocupan en el menú los nombres de las componentes de la trayectoria que pueden elegirse para su representación, mientras que el caso denominado “otherwise” está asignado a lo que se muestra en la opción “-”, que es la que resulta por defecto al abrirse o cerrarse la interfaz y que muestra un círculo cuando no se elige ninguna de las otras opciones.

Al elegir una de las opciones, se recogen los datos transferidos de simulink al workspace y se utilizan para dibujar la gráfica correspondiente.

Debido al código de la simulación, la componente Z de salida del bloque “quadrotor” es negativa, por lo que cogemos el valor de dicha variable de un osciloscopio, el cual guarda los datos como “structure with time” que tiene como entrada el valor absoluto de Z. El resto se recogen de la variable de almacenamiento de simulink “result”, el cual es una estructura de tipo array[18].

En estas gráficas también se visualiza la referencia para poder compararla con la evolución de la salida. Para que no se borre de la gráfica la representación de la referencia al dibujar la coordenada que se desea ver, se utiliza la instrucción “hold(handles.CoordenadasGraf,'on');”. Al inicio o en caso de que se elija otra coordenada, se deshace el “hold” para que se puedan trazas las siguientes sin superponerse con la elección anterior.

```
function SelectorCoordenadas_Callback(hObject, eventdata, handles)
% hObject    handle to SelectorCoordenadas (see GCBO)
```

[18] “result(1:end,1)” significa que los datos que interesan son los de la fila 1 hasta la última de la columna 1.

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
SelectorCoordenadas contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
SelectorCoordenadas
coord=get(handles.SelectorCoordenadas,'value');
hold(handles.CoordenadasGraf,'off');
  switch coord
   case 2
       tiempo=evalin('base','result(1:end,1)'); %Pasa de workspace a
gui
       X=evalin('base','result(1:end,2)'); %Pasa de workspace a gui
       x_ref=evalin('base','x_referencia.signals.values'); %Pasa de
workspace a gui
       plot(handles.CoordenadasGraf,tiempo,x_ref); %dibuja en el axes
CoordenadasGraf
       hold(handles.CoordenadasGraf,'on');
       plot(handles.CoordenadasGraf,tiempo,X,'r'); %dibuja en el axes
CoordenadasGraf
       grid(handles.CoordenadasGraf);
   case 3
       tiempo=evalin('base','result(1:end,1)'); %Pasa de workspace a
gui
       Y=evalin('base','result(1:end,3)'); %Pasa de workspace a gui
       y_ref=evalin('base','y_referencia.signals.values'); %Pasa de
workspace a gui
       plot(handles.CoordenadasGraf,tiempo,y_ref); %dibuja en el axes
CoordenadasGraf
       hold(handles.CoordenadasGraf,'on');
       plot(handles.CoordenadasGraf,tiempo,Y,'r'); %dibuja en el axes
CoordenadasGraf
       grid(handles.CoordenadasGraf);
   case 4
       tiempo=evalin('base','Zr.time'); %Pasa de workspace a gui
       Z=evalin('base','Zr.signals.values'); %Pasa de workspace a gui
       z_ref=evalin('base','z_referencia.signals.values'); %Pasa de
workspace a gui
       plot(handles.CoordenadasGraf,tiempo,z_ref); %dibuja en el axes
CoordenadasGraf
       hold(handles.CoordenadasGraf,'on');
       plot(handles.CoordenadasGraf,tiempo,Z,'r'); %dibuja en el axes
CoordenadasGraf
       grid(handles.CoordenadasGraf);
   case 5
       tiempo=evalin('base','result(1:end,1)'); %Pasa de workspace a
gui
       Yaw=evalin('base','result(1:end,5)'); %Pasa de workspace a gui
       yaw_ref=evalin('base','yaw_referencia.signals.values'); %Pasa
de workspace a gui
       plot(handles.CoordenadasGraf,tiempo,yaw_ref); %dibuja en el
axes CoordenadasGraf
       hold(handles.CoordenadasGraf,'on');
       plot(handles.CoordenadasGraf,tiempo,Yaw,'r'); %dibuja en el
axes CoordenadasGraf
       grid(handles.CoordenadasGraf);
   otherwise
       dibujarcirculoCoordenadasGraf(handles)
  end
```

6.3.2.- Visualización de las acciones y velocidades angulares de los motores.

En este caso, al elegir una opción del menú, no sólo se representa la gráfica sino que aparece un mensaje de texto debajo explicando a que corresponde cada gráfica.

```
% Función del selector de acciones para mostrar uno u otro mensaje
dependiendo de la
% acción del regulador elegida.
% --- Executes on selection change in SelectorAcciones.
function SelectorAcciones_Callback(hObject, eventdata, handles)
% hObject    handle to SelectorAcciones (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
SelectorAcciones contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
SelectorAcciones
set(handles.s, 'String','No se ha seleccionado ninguna Acción');
v=get(handles.SelectorAcciones,'value');
switch v
   case 2
        set(handles.s, 'String','Acción correspondiente al control de
altitud');
        tiempo=evalin('base','T.time'); %Pasa de workspace a gui
        T=evalin('base','T.signals.values'); %Pasa de workspace a gui
        plot(handles.AccionesGraf,tiempo,T); %dibuja en el axes
AccionesGraf
        grid(handles.AccionesGraf);
   case 3
        set(handles.s, 'String','Acción correspondiente al control de
posición en X');
        tiempo=evalin('base','TauPitch.time'); %Pasa de workspace a
gui
        TauPitch=evalin('base','TauPitch.signals.values'); %Pasa de
workspace a gui
        plot(handles.AccionesGraf,tiempo,TauPitch); %dibuja en el axes
AccionesGraf
        grid(handles.AccionesGraf);
   case 4
        set(handles.s, 'String','Acción correspondiente al control de
posición en Y');
        tiempo=evalin('base','TauRoll.time'); %Pasa de workspace a gui
        TauRoll=evalin('base','TauRoll.signals.values'); %Pasa de
workspace a gui
        plot(handles.AccionesGraf,tiempo,TauRoll); %dibuja en el axes
AccionesGraf
        grid(handles.AccionesGraf);
   case 5
        set(handles.s, 'String','Acción correspondiente al control de
orientación respecto Z');
        tiempo=evalin('base','TauYaw.time'); %Pasa de workspace a gui
        TauPitch=evalin('base','TauYaw.signals.values'); %Pasa de
workspace a gui
```

```
        plot(handles.AccionesGraf,tiempo,TauPitch); %dibuja en el axes
AccionesGraf
        grid(handles.AccionesGraf);
   case 6
        set(handles.s, 'String', 'Velocidad del motor 1');
        tiempo=evalin('base','result(1:end,1)'); %Pasa de workspace a
gui
        w1=evalin('base','result(1:end,14)'); %Pasa de workspace a gui
        plot(handles.AccionesGraf,tiempo,w1); %dibuja en el axes
AccionesGraf
        grid(handles.AccionesGraf);
   case 7
        set(handles.s, 'String', 'Velocidad del motor 2');
        tiempo=evalin('base','result(1:end,1)'); %Pasa de workspace a
gui
        w2=evalin('base','result(1:end,15)'); %Pasa de workspace a gui
        plot(handles.AccionesGraf,tiempo,w2); %dibuja en el axes
AccionesGraf
        grid(handles.AccionesGraf);
   case 8
        set(handles.s, 'String', 'Velocidad del motor 3');
        tiempo=evalin('base','result(1:end,1)'); %Pasa de workspace a
gui
        w3=evalin('base','result(1:end,16)'); %Pasa de workspace a gui
        plot(handles.AccionesGraf,tiempo,w3); %dibuja en el axes
AccionesGraf
        grid(handles.AccionesGraf);
   case 9
        set(handles.s, 'String', 'Velocidad del motor 4');
        tiempo=evalin('base','result(1:end,1)'); %Pasa de workspace a
gui
        w4=evalin('base','result(1:end,17)'); %Pasa de workspace a gui
        plot(handles.AccionesGraf,tiempo,w4); %dibuja en el axes
AccionesGraf
        grid(handles.AccionesGraf);
    otherwise
        set(handles.s, 'String','No se ha seleccionado ninguna
Acción');
        dibujarcirculoAccionesGraf(handles)
end
```

6.3.3.- Visualización de la trayectoria.

Esta función se ejecuta cuando es llamada por la función correspondiente al botón de stop tras parar la simulación. Lo que hace es recoger los datos de las coordenadas y dibujarlos en una gráfica 3D. Se recogen tanto los de la trayectoria seguida como los de la trayectoria deseada. Estos últimos proceden del generador de trayectorias (5).

A pesar de que no se muestra en el código, para realizar una visualización en 3D en los objetos “axis” lo primero que hay que hacer modificar la característica “View” en el “Property Inspector”. En este caso, se ha introducido los valores [37.0 0.5]: 37.0 de azimut y 0.5 de elevación.

```
function dibujartrayectoria(handles)
    hold(handles.Trayectoria3D,'off');
    X=evalin('base','result(1:end,2)'); %Pasa de workspace a gui
    Y=evalin('base','result(1:end,3)'); %Pasa de workspace a gui
    Z=evalin('base','Zr.signals.values'); %Pasa de workspace a gui
    plot3(handles.Trayectoria3D,X,Y,Z,'r');
    hold(handles.Trayectoria3D,'on');
   plot3(handles.Trayectoria3D,handles.trayectoria_x,handles.trayector
ia_y,handles.trayectoria_z);
    grid(handles.Trayectoria3D);
    hold(handles.Trayectoria3D,'off');
```

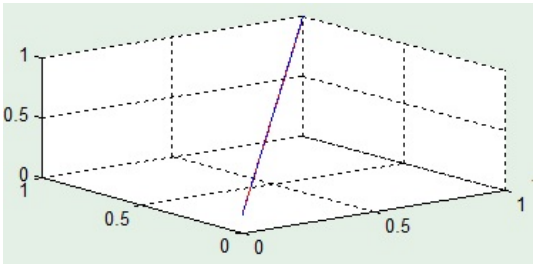

Fig. 51 Visualización de la trayectoria tanto deseada (azul) como la realmente recorrida (roja), punto X=1, Y=1, Z=1 y ψ=90º, aunque no se visualice.

7.- Conclusiones.

Las conclusiones obtenidas del sistema de control son que tenemos un sistema con un tiempo de respuesta de 3s en cada variable y, en general, sin que se den sobreoscilaciones mayores a unos cuantos milímetros o milésimas de radian. La excepción puede ser cuando "yaw" toma un valor de 90º/-90º, donde nos encontramos en una situación de singularidad al producirse un cambio en la dependencia de los ángulos "roll" y "pitch" respecto de x/y, pudiendo darse oscilaciones un poco más acusadas.

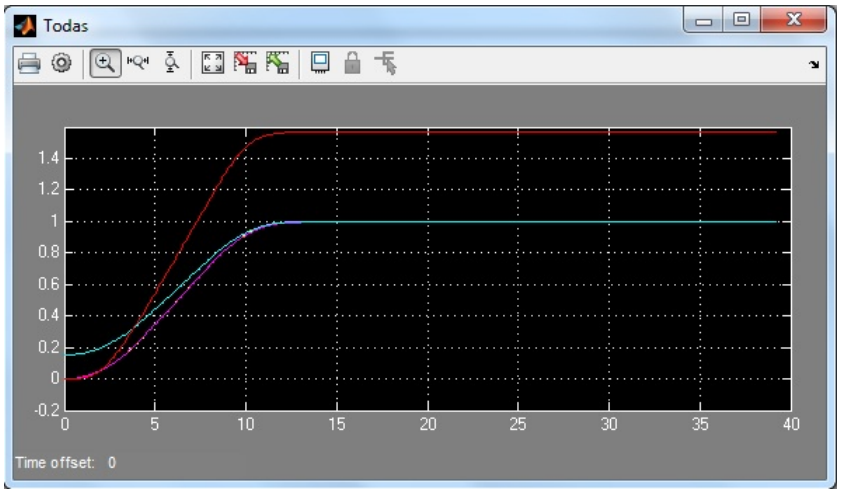

Fig. 52 Respuesta en el caso de singularidad, punto X=1, Y=1, Z=1 y ψ=90º (en la gráfica en radianes).

Sin embargo, hay que tener en cuenta que la "buena" actuación de los reguladores se debe a que el drone se encuentra en condiciones ideales, y no sólo por la ausencia de fricción viscosa con el medio, sino por el hecho de está en una situación climática estable (no hay viento que afecte a su trayectoria, la temperatura podría decirse que se mantiene constante y, por lo tanto, la densidad del aire también) y no tenemos limitaciones en lo que a alimentación eléctrica se refiere. Además, de no tener en cuenta efectos aerodinámicos de segundo orden que, en una situación real, si afectarían tanto a la hora del seguimiento de la trayectoria como a la estrategia de control elegida.

Respecto a la interfaz gráfica, se podrían haber añadido ciertas mejoras como la representación de las gráficas en tiempo real, el mantener los ejes al realizar un zoom sobre la gráfica de la trayectoria o que la ventana de la interfaz se maximice

automáticamente[19] pero que debido a las limitaciones existentes, en lo que a la programación de GUI se refiere, no han permitido su implementación.

De lo realizado en el proyecto, como último punto, reseñar el hecho de que se han ido utilizando funciones para el cálculo de la trayectoria de distintas versiones de la librería “robotic toolbox” (generador de trayectorias, 5).

Como conclusión final puede decirse que el tema tratado en este proyecto es un campo de estudio que se encuentra en plena ebullición tanto en el desarrollo de los componentes necesarios para su construcción como en el cálculo de estrategias de control que permitan un buen control del mismo sin requerir un alto consumo de la fuente de alimentación por parte de los sistemas electrónicos que lo integran.

[19] En este caso, no hay ninguna función documentada que permita ampliar el tamaño de la ventana gráfica ni hay implementada en MATLAB una función que permita hacerlo de manera programable.

8.- Referencias bibliográficas e Informáticas.

A continuación, se enumeran los textos y páginas web consultados para la realización de este proyecto así como los programas informáticos utilizados.

8.1.- Bibliografía.

[1] Corke, P. (2012) Manual for Robotics ToolBox for MATLAB 9.7.

[2] Piedrafita Moreno, R. (2009) Control de sistemas industriales Continuos. Zaragoza: Editorial Cronos.

[3] Piedrafita Moreno, R., Romeo Tello, A. (2009) Control Automático en los dominios frecuencial y de tiempo Discreto.

[4] Mahony,R., Kumar, V., Corke, P. (2012). Multirotor Aerial Vehicles: Modeling, Estimation and Control of Quadrotor. IEEE ROBOTICS & AUTOMATION MAGAZINE (SEPTEMBER 2012, 20-3).

[5] Leishman, J.G. (2006) Principles of helicopter Aerodynamics. Cambridge: Cambridge University Press.

[6] Pounds, P. (2007) Design, construction and control of a large quadrotor micro air vehicle. The Australian National University. (Tesis doctoral).

[7] Pounds, P., Mahony, R., Corke, P. (2010). Modelling and control of a large quadrotor robot. Control Engineering Practice, vol. 18. Elsevier.

[8] Lee, T., Leok, M., McClamroch, N. H. (2010) Geometric Tracking Control of a quadrotor UAV on SE(3). 49th IEEE Conference on Decision and Control.

[9] Yu Yali, Sun Feng, Wang Yuanxi. (2012) Controller Design of Quadrotor Aerial Robot. 2012 International Conference on Medical Physics and Biomedical Engineering. Elsevier B.V.

[10] Kumar, V. Cartesian Trayectory Planning for Robot Manipulators. http://www.seas.upenn.edu/~meam520/notes02/CartesianControl10.pdf

[11] Barientos, A., Peñín, L.F., Balaguer, C., Aracil, R. (2009) Fundamentos de Robótica. Madrid: Editorial McGraw Hill.

[12] http://www.mathworks.es/matlabcentral/ (página web de la comunidad de usuarios de MATLAB.)

[13] Parada Pardo, E. (2012) Quadcopter: construcción, control de vuelo y navegación GPS. Universidad Carlos III de Madrid, Departamento de Ingeniería de sistemas y automática.

8.2.- Programas de cálculo y software.

- Robotics ToolBox 9.7 for MATLAB. Peter Corke, 2012[20].
- MATLAB R2012b. Mathworks, 2012.
- MATLAB 7.1. Mathworks, 2010.

[20] Versión de: P.Corke. (1996) A robotics toolbox for MATLAB. IEEE Robotics and Automation Magazine. (Sept. 1996; vol. 3, pp. 2432).

9.- Anexos.

En los anexos se incluyen el esquema final del control, la interfaz gráfica junto con su código[21], las funciones utilizadas en la obtención de la trayectoria y la librería donde se encuentra el archivo correspondiente al código de la simulación del quadrotor.

[21] Es necesario realizar la ejecución del código de la interfaz para que ésta se abra.