Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Grado

DISEÑO DE UN SISTEMA DE CONTROL DE UN CUADRICÓPTERO PARA USO DOCENTE

Alumno: Parra Montes, Jesús

Tutor: Prof. D. Pablo Cano Marchal
Dpto: Ingeniería Electrónica y Automática

, 2022

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Electrónica y Automática

Don Pablo Cano Marchal, tutor del Proyecto Fin de Carrera titulado: Diseño de un sistema de control de un cuadricóptero para uso docente, que presenta Jesús Parra Montes autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, de 2022

El alumno: Los tutores:

NOMBRE_ALUMNO NOMBRE _TUTOR

ÍNDICE

1. INTRODUCCIÓN

En el presente Trabajo Fin de Grado (TFG), se procede al diseño de un sistema de control de un cuadricóptero para uso docente. Para ello, se parte de un TFM realizado en el año 2020, en el que se diseña tanto los componentes del dron, el diseño electrónico y mecánico y un control estabilizador mediante un control en variables de estado. En este TFG se va a partir de su programación de inicialización de todos sus componentes, pero se va simplificar la programación de control debido a que este proyecto va destinado, cómo su título dice, al uso docente. Es decir, que los alumnos y alumnas de cursos de inferiores puedan realizar un ensayo de identificación y control estabilizador del cuadricóptero de una manera más o menos sencilla. Entonces, para la realización de este TFG se parte del Trabajo Fin de Máster, comentado anteriormente, y de los conocimientos adquiridos en el Grado de Ingeniería Electrónica Industrial, sobre todo, los dos últimos años.

1.1 JUSTIFICACIÓN

El hecho de la creación de UAVs (de las siglas en inglés Unmanned Aerial Vehicule), vienen de muy lejos en tiempo y, sobre todo, de la motivación del ser humano en intentar asemejarse a los pájaros e intentar mejorar su vuelo. A partir de esto, lo primero que hizo el ser humano fue la creación de un vehículo aéreo controlado por una persona que estaba montada en dicho vehículo, el siguiente paso fue la creación de un vehículo controlado por un ser humano, pero la diferencia que había era que se hacía desde una forma remota, es decir, un ejemplo sería un sistema radiocontrol o lo que conocemos como el sector de los RPAs

(Remotely Piloted Aircraft). El sector de los RPAs se contempla como un valor tecnológico en alza, orientado a servicios múltiples en los ámbitos civil y militar. Su pujante incursión les lleva a ser cada vez más necesarios en aplicaciones tecnológicas de sectores básicos para la sociedad civil, sin mencionar su uso sistemático en el ámbito militar.

Por último, y lo más actual, es el control de dicho sistema mediante una programación que hace que el sistema sea prácticamente autónomo en su recorrido o vuelo. Para la realización de este proyecto nos vamos a centrar en este último punto que he comentado.

2. ANTECEDENTES

En este capítulo se expone la evolución de los drones, y en especial de los cuadricópteros, desde su creación hasta la actualidad.

2.1 COMIENZO Y DESARROLLO

La idea del avión no tripulado es antigua. A pesar de que a menudo asociamos los drones con los robots militares de hoy, los aviones no tripulados, de una forma u otra, se han utilizado durante décadas. Uno de los primeros usos registrados fue por los austriacos en julio de 1849 después de que se pusieran en marcha alrededor de doscientos globos aerostáticos no tripulados montados con bombas en la ciudad de Venecia.

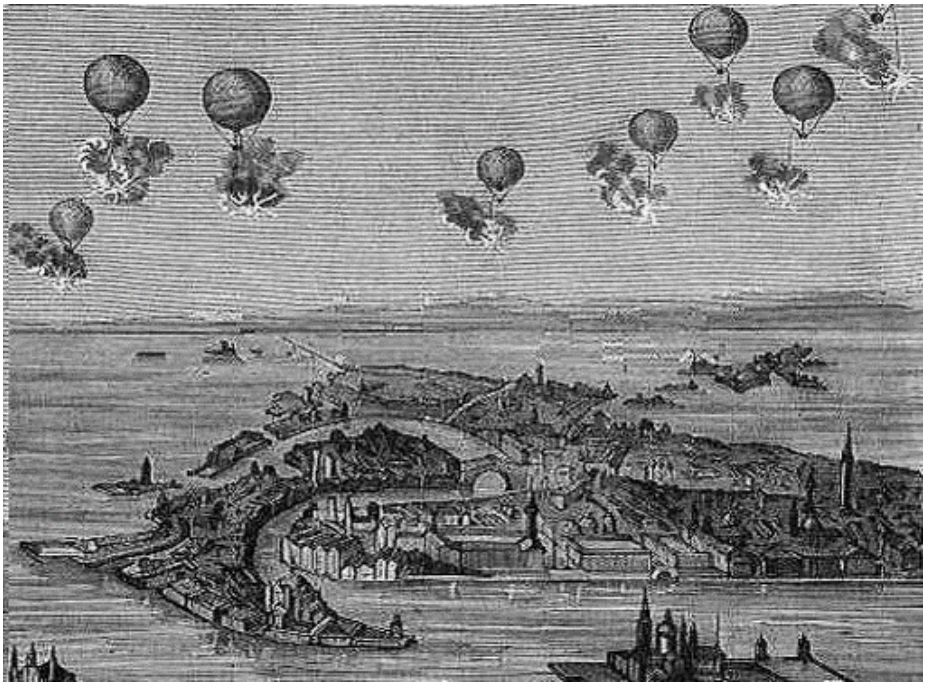


Figura 2.1: Bombardeo de Venecia

Pero no fue hasta 1907 cuando se creó el primer cuadricóptero, a manos de Jacques y Louis Breguet; sin embargo, se necesitaba de cuatro hombres para estabilizar y apenas se levantaba dos pies del suelo.

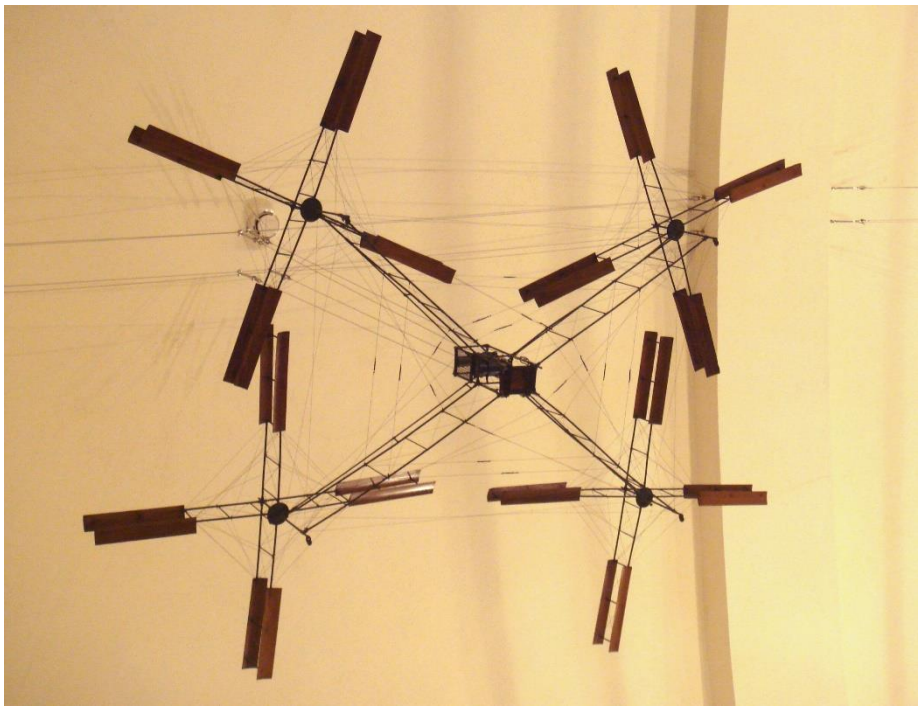


Figura 2.2: Giroplano Breguet-Richet

El siguiente avance importante, fue en 1917, cuando el Ruston Proctor Aerial Target, controlado por radio (basado en la tecnología RC del inventor Nikola Tesla), se convirtió en el primer avión sin piloto; se pretendía usar como una bomba, más nunca se llevó a cabo.

En 1943, durante la Segunda Guerra Mundial, fue creado el FX-1400, apodado «Fritz X», para ser utilizado por los militares alemanes. La primera arma de control remoto que realmente se puso en uso operativo.

Figura 2.3: Fritz X

Una bomba de 2,300 libras que se usó para hundir barcos. Este no solo fue el primer avión no tripulado militar que se desplegó correctamente, sino que también fue el antecesor de los modernos misiles antibuque y otras armas guiadas de precisión.

En la década de los 60, los avances en la tecnología de transistores significaron que, por primera vez, los componentes miniaturizados controlados por radio estaban disponibles para los clientes a un costo razonable.

Lo que siguió fue un *boom* de popularidad en los aviones RC de los Estados Unidos. En su mayoría, en forma de kit, estos aviones RC ofrecían de todo, desde modelos con capacidad de vuelo interior, hasta modelos al aire libre mucho más grandes.

Desde entonces ha habido un gran avance en el desarrollo de los UAVs, tal ha sido este que, en el año 2010, la compañía francesa Parrot lanzó su Parrot AR Drone, el primer dron listo para volar que se podía controlar completamente a través de Wi-Fi, usando un teléfono inteligente.

Figura 2.4: Parrot AR Drone

Actualmente, un dron puede hacer casi cualquier cosa, de hecho, existen algunos que llevan incorporados inteligencia artificial y son capaces de tomar decisiones en sus vuelos.

2.2 DRONES INNOVADORES

A continuación, veremos algunos de los cuadricópteros más innovadores que existen en la actualidad.

- Automóvil volador – NEC: La empresa japonesa Tecnológica NEC presentó hace poco tiempo su modelo de "Automóvil volador" de cuatro hélices y con capacidad para dos personas. Se está invirtiendo en mejorar la suspensión y la estabilidad y no se espera su salida hasta el año 2030.

Figura 2.3: Automóvil volador (Tecnología NEC)

- EHang: La empresa china Beijing Yi-Hang Creation Science & Technology Co. se ha esforzado en invertir un desarrollo diferenciador de modelo efectivo de Dron Taxi ó Vehículo Dron denominado EHang.

Figura 2.4: Vehículo dron EHang

- Hybrix 2.1: La empresa Quaternium ha desarrollado una serie de modelos de Drones de gran envergadura destinados a labores de regado, control de plagas, y gestión de los terrenos, utilizando un sistema híbrido, a través de gasolina y electricidad. Este modelo comienza a tomar fuerza en el sector agrario que quiera apostar por la calidad y la precisión.

Figura 2.5: Hybrix 2.1

3. OBJETIVOS

Los objetivos de este proyecto son los siguientes:

- Diseño de un método de identificación del sistema a controlar, con el cuál podamos tener el sistema totalmente definido.
- Implementación de dicho método de identificación.
- Realización de una interfaz de usuario para poder controlar las variables del proceso de identificación.
- Diseño de un controlador estabilizador en los ángulos de ataque y alabeo. Que nos permita, con una programación sencilla, estabilizar el dron.
- Implementación del controlador estabilizador.
- Diseño de un controlador estabilizador avanzado. Que ya con una programación más compleja y de forma mejorada, se realiza el control estabilizador.
- Implementación del controlador avanzado.
- Implementación del código de envío y recepción de datos entre el microcontrolador y un PC.

El control del sistema se realizará en una placa de desarrollo libre como es el caso de la Teensy 3.2 que es una tarjeta de bajo costo basada en un procesador ARM Cortex-M4. El entorno de programación del diseño de los controladores será Arduino y el desarrollo de la interfaz de usuario será una combinación de Arduino y Processing.

4. MATERIALES Y MÉTODOS

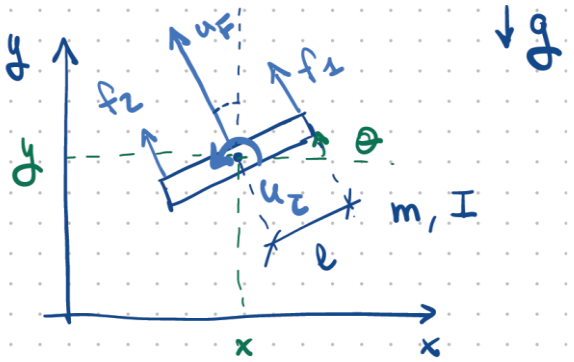
En este capítulo abordaremos el tema en cuestión y empezaremos por la de identificación de la señal del sistema, luego diseñaremos un controlador sencillo que controle de una manera más o menos simple el sistema y, por último, diseñaremos un controlador avanzado que controle de una manera mucho más precisa el sistema.

4.1 IDENTIFICACIÓN

Esta parte la dividiremos en una primera que se va destinar a la definición de nuestra planta, definición de dicho proceso de identificación y, luego, en los diferentes métodos que se han intentado para poder identificar el sistema.

4.1.1 DEFINICIÓN DE LA PLANTA

El sistema de un dron en dos 2D que es nuestro se definiría con el siguiente sistema de ecuaciones:



1. $m \cdot \ddot{x} = -u_F \cdot \sin\theta \quad (4.1.1.1)$
2. $m \cdot \ddot{y} = u_F \cdot \cos\theta - mg \ (4.1.1.2)$
3. $I \cdot \ddot{\theta} = u_\tau \quad (4.1.1.3)$

Figura 4.1.1.1: Sistema dron 2D

Donde m es la masa del propio dispositivo, $\ddot{x}$ *e* $\ddot{y}$ son las derivadas segundas de las posiciones del dron con respecto al eje x e y, $\ddot{\theta}$ es la derivada segunda del ángulo que forma con respecto al horizontal, u_F y u_τ es la fuerza total y el momento total que ejerce el dron e I es la inercia que tiene nuestro sistema. Con la tercera ecuación de nuestro sistema de ecuaciones podemos definir la planta de nuestro sistema, ya que en

nuestro caso no va haber desplazamiento en ninguno de los ejes. Por lo tanto, trabajando con la ecuación 4.1.1.3 y sabiendo que la entrada de nuestro sistema sería el momento aplicado y la salida sería el ángulo formado, poniendo una en función de la otra, obtendríamos:

$$\frac{\ddot{\theta}}{u_\tau} = \frac{1}{I} \qquad (4.1.1.4)$$

Pasándolo al dominio de Laplace:

$$P(s) = \frac{\theta(s)}{u_\tau(s)} = \frac{1}{I \cdot s^2} = \frac{K_P}{s^2} \qquad (4.1.1.5)$$

Con esto ya tendríamos la forma de nuestra planta, lo que nos quedaría sería definir la ganancia que tiene la planta que lo explicaremos y desarrollaremos en los siguientes apartados.

4.1.2 DEFINICIÓN DE LA IDENTIFICACIÓN

Podemos definir la identificación de sistemas, como los estudios de técnicas que persiguen la obtención de modelos matemáticos de sistemas dinámicos a partir de mediciones realizadas en el proceso: entradas o variables de control, salidas o variables controladas y perturbaciones.

El enfoque de la identificación se puede realizar en función de la estructura del modelo, y del comportamiento físico o no del mismo. Podemos distinguir:

- Black-box: los parámetros del modelo no tienen una interpretación física. Un modelo basado en leyes fundamentales es muy complicado o se desconoce.
- Gray-box: algunas partes del sistema son modeladas basándose en principios fundamentales, y otras como una caja negra. Algunos de los parámetros del modelo pueden tener una interpretación física; a

este tipo de modelos también se les conoce como “Tailor-made”, estimando sólo los parámetros no conocidos.

- White-box: la estructura del modelo se obtiene a partir de leyes fundamenta- les. Los parámetros tienen una interpretación física.

Tipos de modelos

Existen varias formas de catalogar los modelos matemáticos: deterministas o estocásticos, dinámicos o estáticos, de parámetros distribuidos o concentrados, lineales o no lineales, y de tiempo continuo o tiempo discreto; i.e.:

- Modelos mentales. Sin formalismo matemático.
- Modelos no paramétricos. Se caracterizan mediante gráficos, diagramas o representaciones que describen las propiedades dinámicas mediante un número no finito de parámetros, i.e., respuesta al impulso, al escalón, o en frecuencia.
- Modelos paramétricos o matemáticos. Describen las relaciones entre las variables del sistema mediante expresiones matemáticas; i.e., ecuaciones diferenciales en sistemas continuos y ecuaciones de diferencias en sistemas discretos.

En función del tipo de sistema y de la representación matemática utilizada, los sistemas pueden clasificarse en:

- Determinísticos o estocásticos: se dice que un modelo es determinístico, cuando expresa la relación entre entradas y salidas mediante una ecuación exacta; se estudia la relación entre la entrada y la salida con una parte no conocida. Por contra, un modelo es estocástico si posee un cierto grado de incertidumbre. Que- dan definidos mediante conceptos probabilísticos o estadísticos.

- Dinámicos o estáticos: un sistema es estático cuando la salida depende únicamente de la entrada en ese instante de tiempo. La función que relaciona las entradas con las salidas, es independiente del tiempo. En un sistema dinámico las salidas varían con el tiempo, el valor actual de la salida en función del tiempo transcurrido desde la aplicación de la entrada. Tienen como objetivo conocer el comportamiento dinámico de un proceso.
- Continuos o discretos: los sistemas continuos trabajan con señales continuas, se formalizan mediante ecuaciones diferenciales. Los sistemas discretos trabajan con señales muestreadas, se describen por medio de ecuaciones en diferencias.
- De parámetros concentrados: no se considera la variación en función del espacio.
- Lineales o no lineales: un sistema lineal se define por una función matemática lineal.

$$y'(t) + a \cdot y(t) = u(t) \quad es\ lineal$$

$$y'(t) + a \cdot y^2(t) + b \cdot y^3(t) = u(t) \quad no\ es\ lineal$$

Con lo expuesto anteriormente, utilizaremos un modelo no paramétrico y, en concreto, un análisis en frecuencia, ya que para su utilización no se supone ningún tipo de estructura para el modelo: no se emplea explícitamente un vector de parámetros en el modelo matemático, sino que se determinan características del sistema a partir del modelo matemático y, en general, mediante el auxilio de funciones espectrales o haciendo uso de la información entrada-salida del sistema en cuestión (que es nuestro caso).

4.1.3 MÉTODOS DE IDENTIFICACIÓN

Con el ensayo en frecuencia vamos a determinar la ganancia de nuestro sistema y así, tener nuestro sistema totalmente definido.
Por lo tanto, para ello vamos a introducir una función seno en nuestra planta del sistema (el cuadricóptero), vamos a ver su evolución en el tiempo y definiremos nuestra planta relacionando la salida con la entrada. Es decir, tendríamos una entrada y una salida función seno, pero la salida tendría un desfase, i.e.:

$$u(t) = A_e \cdot \sin(\omega t) \qquad (4.1.3.1)$$

$$y(t) = A_s \cdot \sin(\omega t + \varphi) \qquad (4.1.3.2)$$

Así tendríamos lo que vamos buscando que es la identificación de nuestro sistema. Las variables de la función de entrada al sistema (amplitud y frecuencia) quedarían definidas en tiempo real por el usuario mediante la interfaz creada. Teóricamente lo que nos debería de salir tendría que ser lo siguiente:

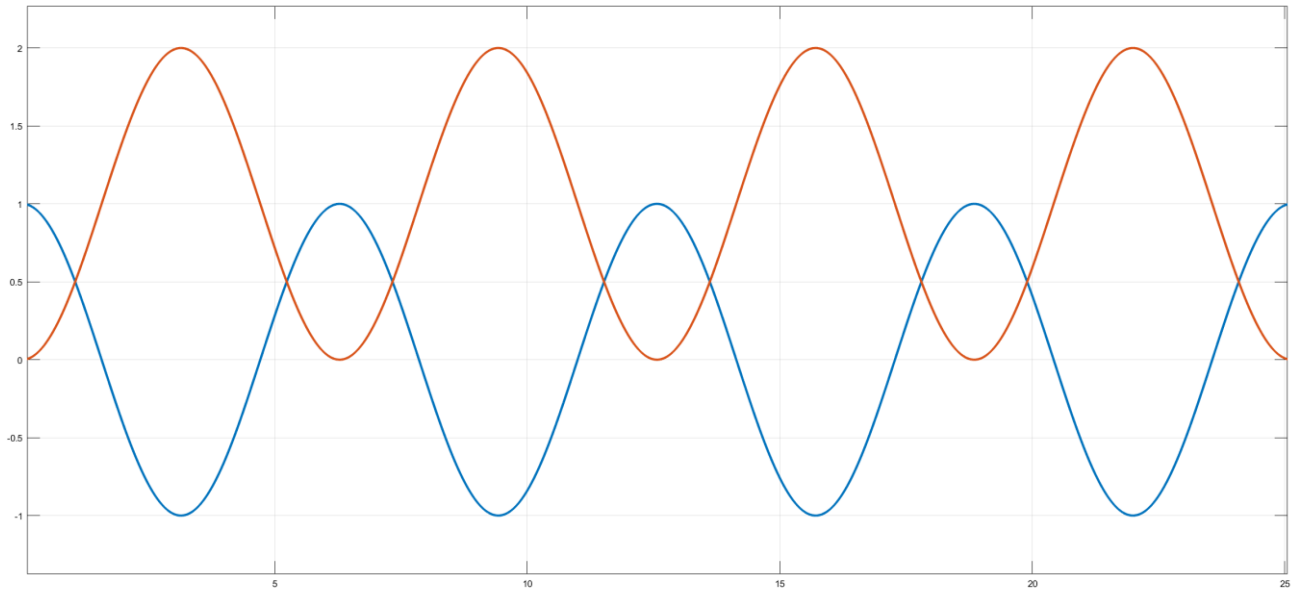


Figura 4.1.3.1: Gráfica teórica de ensayo de identificación donde la entrada al sistema es una función seno

La complicación vino a que, al ser un sistema con doble integrador, le afecta de una manera más significativa las perturbaciones del entorno y, en este caso específico, venía relacionado con las vibraciones que generaba el propio dron al oscilar, ya que esas vibraciones hacían que el dron cediera hacia un lado cuando le daba fuerzas de mayor rango, hice varias pruebas e intente corregir dicho error (reduciendo la fuerza en los rotores y dándole más frecuencia de oscilación,…), pero finalmente esta opción se tuvo que descartar porque no era viable ya que obtuve la siguiente gráfica:

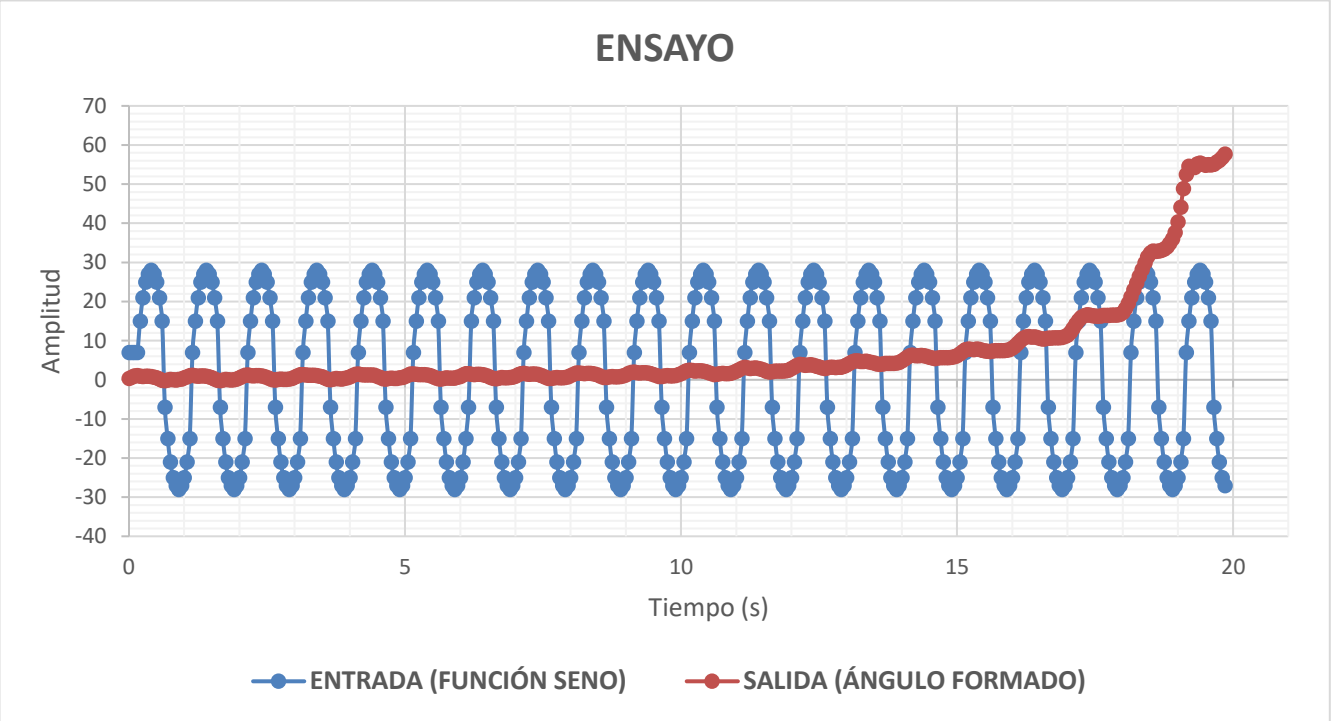


Figura 4.1.3.2: Gráfica práctica de ensayo de identificación donde la entrada al sistema es una función seno

Otra opción que intenté, fue la de introducir una realimentación unidad al sistema y como señal de entrada, una señal escalón, y que, el sistema al ser un doble integrador, el sistema debería oscilar como una función seno con unas características definidas, es decir, tendríamos el siguiente diagrama de bloques:

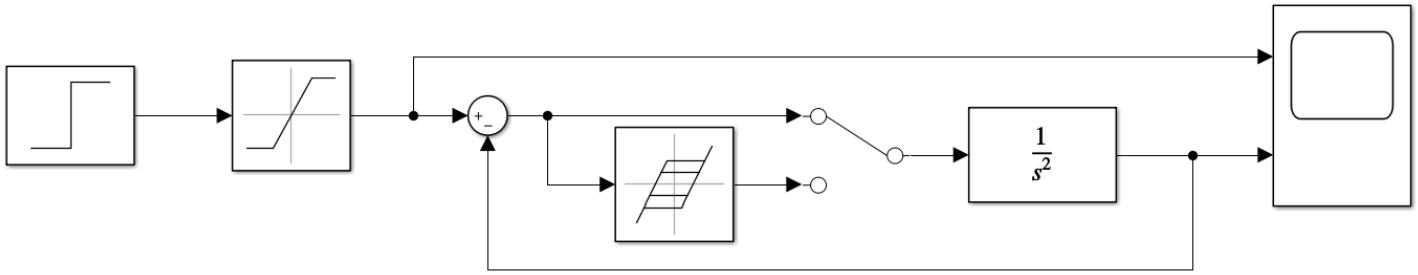


Figura 4.1.3.3: Diagrama de bloques del 2º método de identificación

Que obtendríamos la siguiente gráfica:

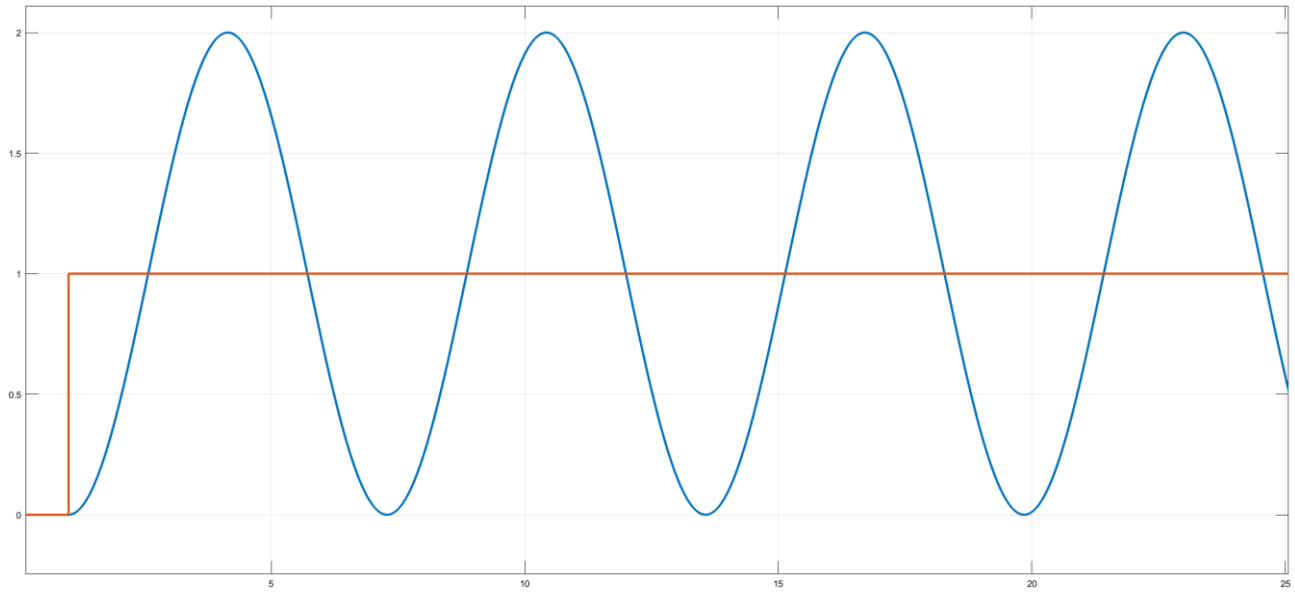


Figura 4.1.3.4: Gráfica teórica de ensayo de identificación donde la entrada al sistema es una función escalón

Pero, al introducir la perturbación en el sistema la amplitud de dicha función seno (señal de salida) iba con el tiempo y, finalmente, no podimos implementar dicha opción ya que no se mantenía lo suficientemente estable en el tiempo para poder definirla, i.e.:

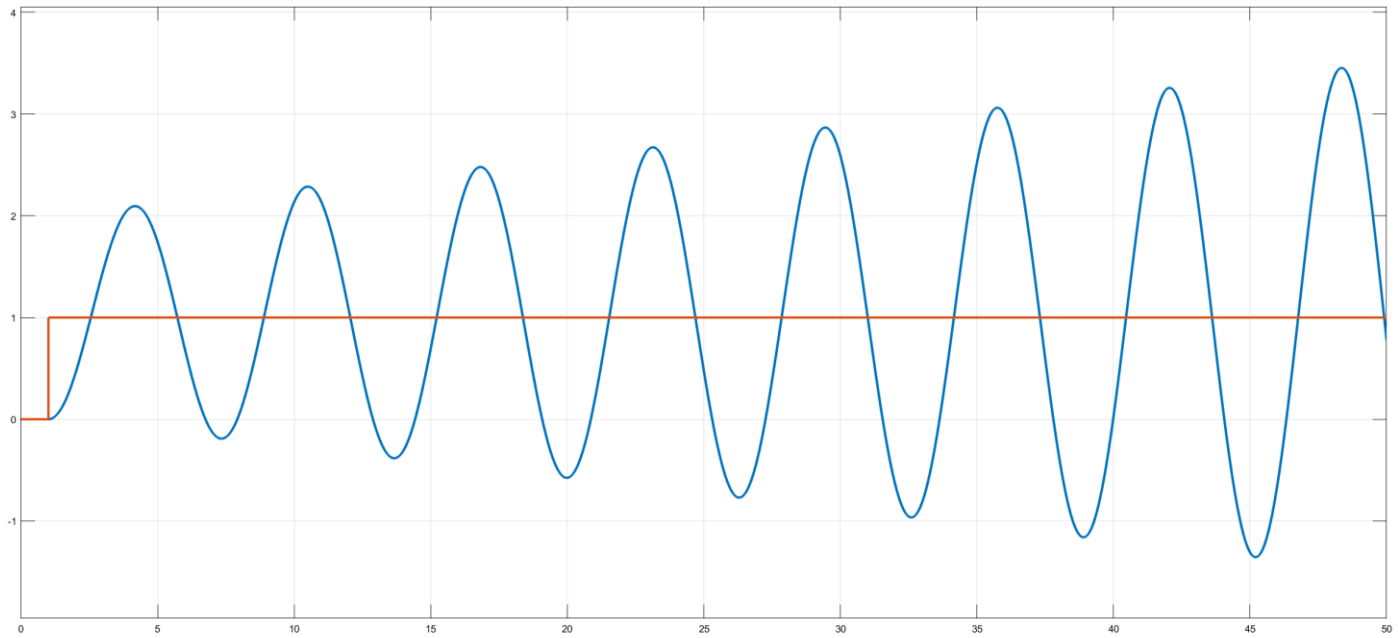


Figura 4.1.3.5: Gráfica experimental de ensayo de identificación donde la entrada al sistema es una función escalón

Finalmente, buscamos otras posibles opciones, pero no encontramos una forma relativamente fácil y directa para poder implementarla. Ya que la finalidad de este TFG es que lo puedan realizar alumnos y alumnas de cursos inferiores con unas nociones básicas de control e identificación de sistemas, como lo he comentado anteriormente.

4.2 CONTROLADOR

En esta primera parte, diseñaremos un controlador más sencillo como es un controlador PD (Proporcional-Diferencial). En la primera parte, se diseñará teóricamente el controlador como tal y, en la segunda parte, se realizará una implementación del mismo.

4.2.1 DISEÑO DEL CONTROLADOR

Un controlador PD es un elemento de transferencia de un sistema de control de bucle cerrado que comprende componentes de elemento tanto Proporcional como Diferencial. El componente diferencial responde a la velocidad en la que el error de control cambia. El valor es

multiplicado por el coeficiente de acción-derivativa y sumado al componente Proporcional (lo que, por su parte, actúa proporcionalmente en un error de control específico). Como resultado, el controlador PD puede responder un error de control inminente y, por lo tanto, lograr una acción derivativa durante el proceso de control. El controlador PD es un controlador rápido, sin embargo, al igual que el controlador Proporcional, no puede corregir completamente un error de control. Partiremos del siguiente diagrama de bloques, que define un proceso general donde existe una acción de control y una realimentación (enfatizar que sin realimentación es casi imposible controlar un sistema).

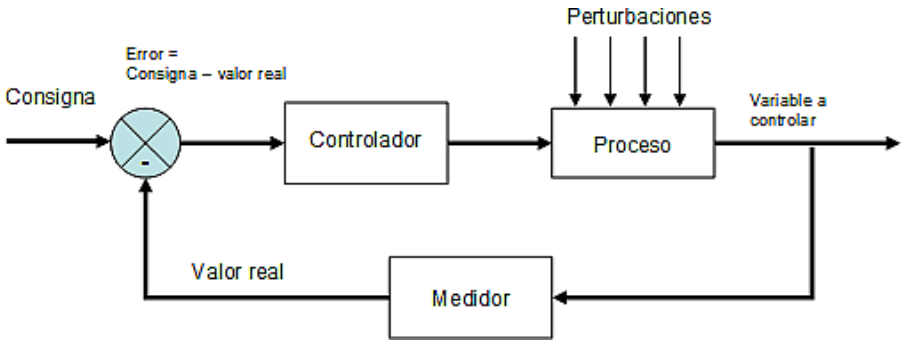


Figura 4.2.1.1: Diagrama de bloques

El controlador PD se utiliza principalmente para mejorar la respuesta transitoria de un sistema de control. La acción de control de un controlador proporcional-derivativo (PD) se define mediante:

$$u(t) = K_C e(t) + K_C T_D \frac{de(t)}{dt} \qquad (4.2.1.1)$$

Si lo pasamos a términos de Laplace nos quedaría:

$$U(s) = K_C E(s) + K_C T_D s E(s) \qquad (4.2.1.2)$$

$$U(s) = K_C E(s)(1 + T_D s) \qquad (4.2.1.3)$$

Donde la función de transferencia del controlador PD sería:

$$\frac{U(s)}{E(s)} = K_C\,(1 + T_D s) \qquad (4.2.1.4)$$

En la cual, T_D se definiría como tiempo derivativo y la K_C sería la ganancia del controlador. Como podemos observar, el controlador PD introduce un cero en el sistema que depende del valor de T_D (cero = $-1/T_D$). Entonces, en función del valor del cero que introduzcamos en el sistema haremos que sistema sea más rápido, estable o inestable, etcétera. Por lo tanto, lo primero que vamos a realizar, es definir un valor para esta variable. Para ello, lo primero que definiremos será el tiempo de establecimiento de un sistema de segundo orden, que, si utilizamos el criterio del 2%, tendríamos:

$$t_s = \frac{4}{\sigma} \qquad (4.2.1.5)$$

Donde t_s es el propio tiempo de establecimiento y σ son los diferentes valores del eje real en el lugar de las raíces. Sabiendo esto, si fijamos un valor de σ igual a 4, tendríamos un tiempo de establecimiento del sistema relativamente rápido y aproximadamente igual a 1 segundo. Con este valor fijado, ya podemos obtener el valor del cero del sistema y del tiempo derivativo. Ya que para que nuestro sistema corte en -4 en el eje real, debemos introducir un cero con un valor de -2 para así cumplir con las especificaciones de construcción del lugar de las raíces.

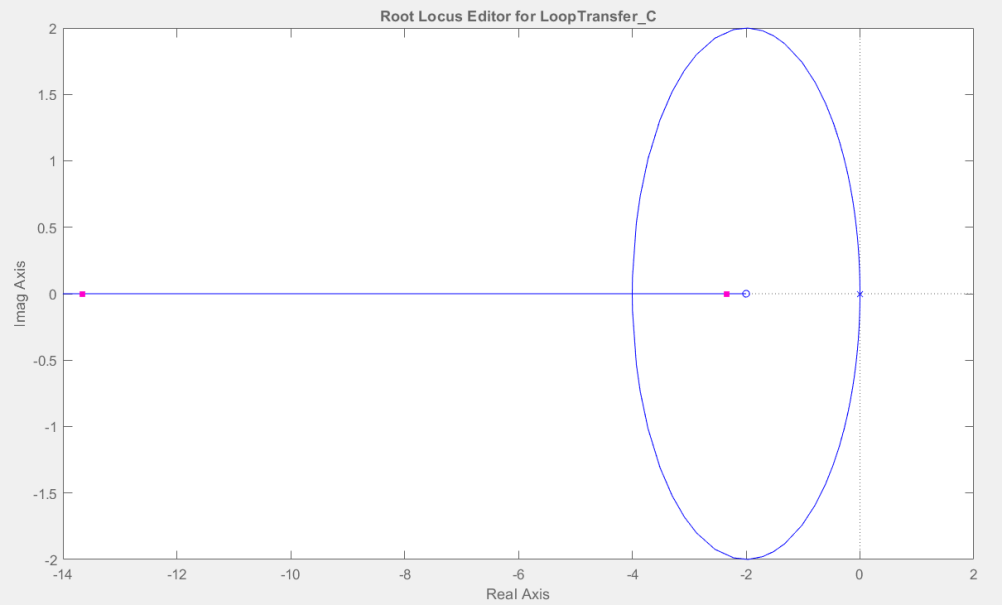


Figura 4.2.1.2: Lugar de las raíces aproximado del sistema

Con esto, ya sabríamos el valor del tiempo derivativo ya que, como he definido antes, valdría el valor negativo de la inversa del valor del cero, es decir, 0.5.

Por otra parte, el error que existe en un sistema es la referencia que se quiere obtener menos la medida que obtenemos mediante, por ejemplo, un sensor. En nuestro caso, como queremos que nuestra referencia sea igual a cero, quedaría:

$$e = r - \theta = -\theta \qquad (4.2.1.6)$$

Al definir cuánto vale el error podemos definir la derivada de este:

$$\frac{de}{dt} = -\dot{\theta} = -\omega \qquad (4.2.1.7)$$

Donde ω es la velocidad angular que tiene el dron al girar.

Si juntamos lo que hemos calculado anteriormente y lo juntamos con la ecuación 4.2.1, nos queda:

$$u(t) = -K_C \cdot \theta - (0.5 \cdot K_C \cdot \omega) \qquad (4.2.1.8)$$

Por otro lado, tenemos que nuestro sistema se define por la ecuación 4.1.1.5 que es la siguiente:

$$P(s) = \frac{K_P}{s^2} \qquad (4.1.1.5)$$

Donde la K_P es la ganancia de la planta. Como he comentado anteriormente, no ha sido posible calcular un valor exacto de la ganancia de la planta debido a los problemas que he tenido con respecto a la componente no lineal del sistema, por esto he calculado un valor aproximado de dicha ganancia cogiendo sola una parte de la gráfica del ensayo antes de que el dron cediera hacía uno de sus lados y obtuve lo siguiente:

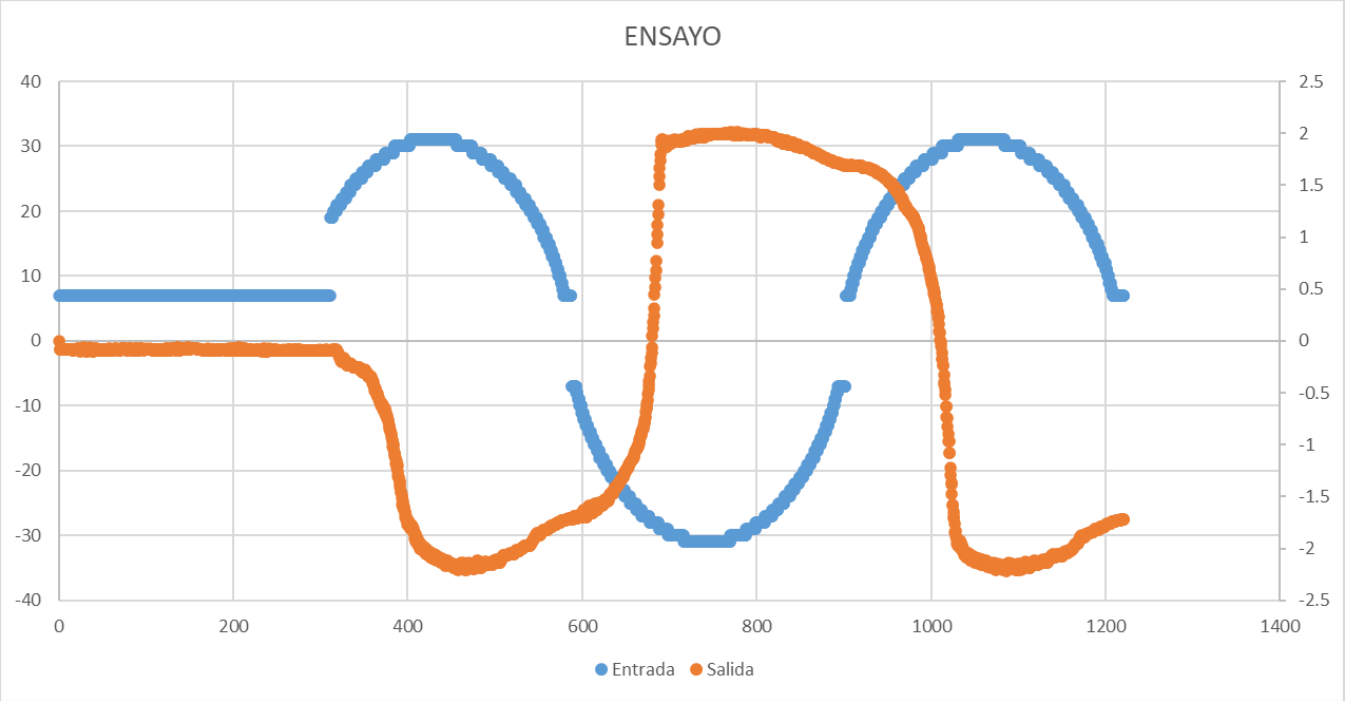


Figura 4.2.1.3: Ensayo para el cálculo de la Kp

$$K_P = \frac{\Delta A_s}{\Delta A_e} = \frac{2{,}01 - (-2{,}22)}{31 - (-31)} = 0.06823 \qquad (4.2.1.9)$$

Por lo tanto, nos quedaría:

$$P(s) = \frac{0.06823}{s^2} \qquad (4.2.1.10)$$

Por último, realizaremos el cálculo de la ganancia del controlador, ya que al tener la función de transferencia del controlador y de la planta, podemos calcular la función de lazo de cerrado del sistema y, sabiendo que, en el lugar de las raíces del sistema corta el eje real en s = -4, podemos calcular el valor de la ganancia del controlador.

Por lo tanto, la función de lazo abierto del sistema sería la siguiente:

$$L(s) = \frac{K_C K_P (1 + T_D s)}{s^2} \qquad (4.2.1.11)$$

Ahora, calculamos la función de lazo cerrado del sistema con la siguiente ecuación:

$$Gyr = \frac{L(s)}{1+L(s)} = \frac{K_C\,K_P(1+T_D s)}{s^2 + K_C\,K_P(1+T_D s)} = \frac{K_C\,K_P(1+T_D s)}{s^2 + K_C\,K_P T_D s + K_C\,K_P} \quad (4.2.1.12)$$

Realizando lo que he explicado anteriormente:

$$s^2 + K_C\,K_P T_D s + K_C\,K_P = 0, \quad para\ s = -4$$

Dónde Kp = 0.06823 y Td = 0.5, despejando la Kc obtenemos un valor de 234.5. Finalmente, ya podemos definir completamente nuestro controlador y haciendo una conversión para transformarlo al dominio de Laplace obtendríamos lo siguiente:

$$C(s) = 234.5 \cdot (s+2) \qquad (4.2.1.13)$$

Con este diseño de controlador tendríamos la siguiente respuesta del sistema ante una entrada escalón:

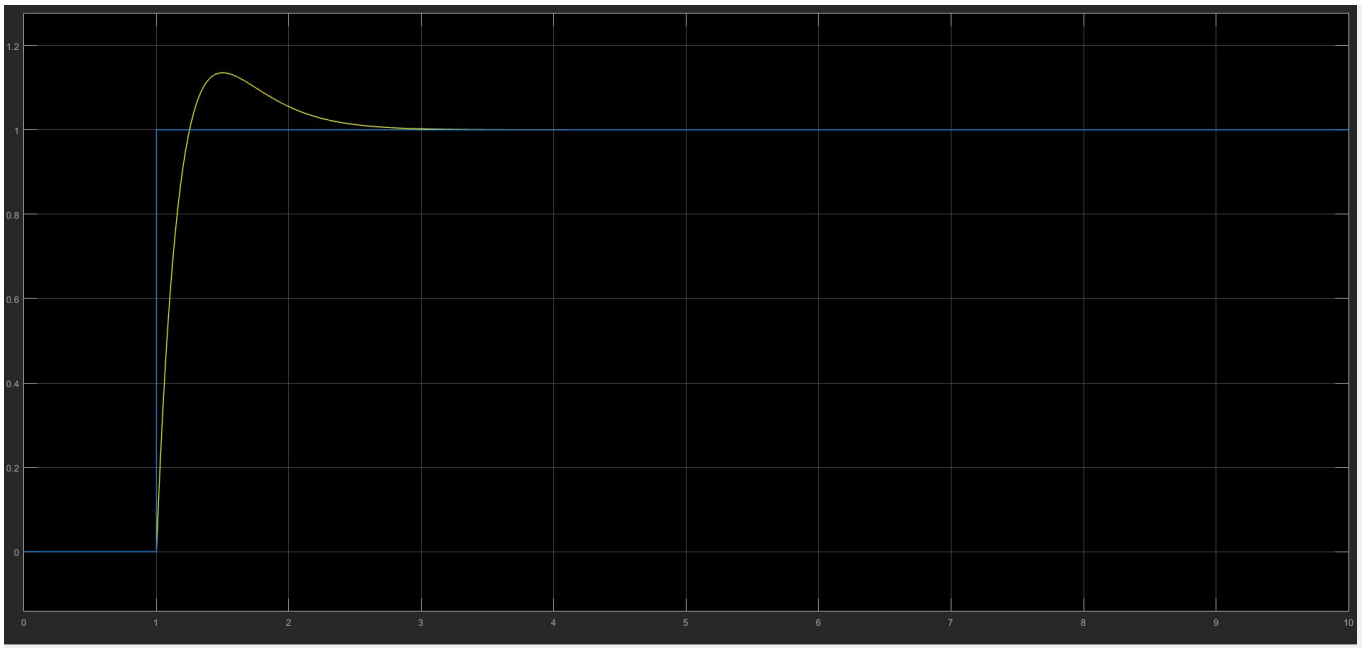


Figura 4.2.1.4: Respuesta del sistema ante una entrada escalón

Como podemos observar el sistema es relativamente rápido con un tiempo de establecimiento de, aproximadamente, 2 segundos y una sobreoscilación con un valor de alrededor de 0.12.

4.2.2 IMPLEMENTACIÓN DEL CONTROLADOR

Con esto y lo anteriormente descrito, podemos definir perfectamente una entrada al sistema que nos permita controlar el ángulo de giro de nuestro cuadricóptero. Haciendo uso de la ecuación 4.2.1.8 y sustituyendo valores, obtenemos:

$$u(t) = -234.5\theta - 117.5\omega \qquad (4.2.2.1)$$

Donde u(t) es la entrada a nuestro sistema, que como dije antes, es el momento que aplicamos, es decir, sería la potencia que aplicamos en los rotores para rectificar la perturbación que recibe el dron en el ángulo. Que como podemos observar en la ecuación dependería del ángulo que forma el dron con la horizontal (introduciremos el valor en grados, para así cuadrar unidades y que nos salga rpm) y de la velocidad angular del propio dispositivo.

Por último, veremos un ejemplo en una serie de gráficas de cómo el controlador diseñado es capaz de rectificar el valor de la perturbación que modifica la estabilidad del dron.

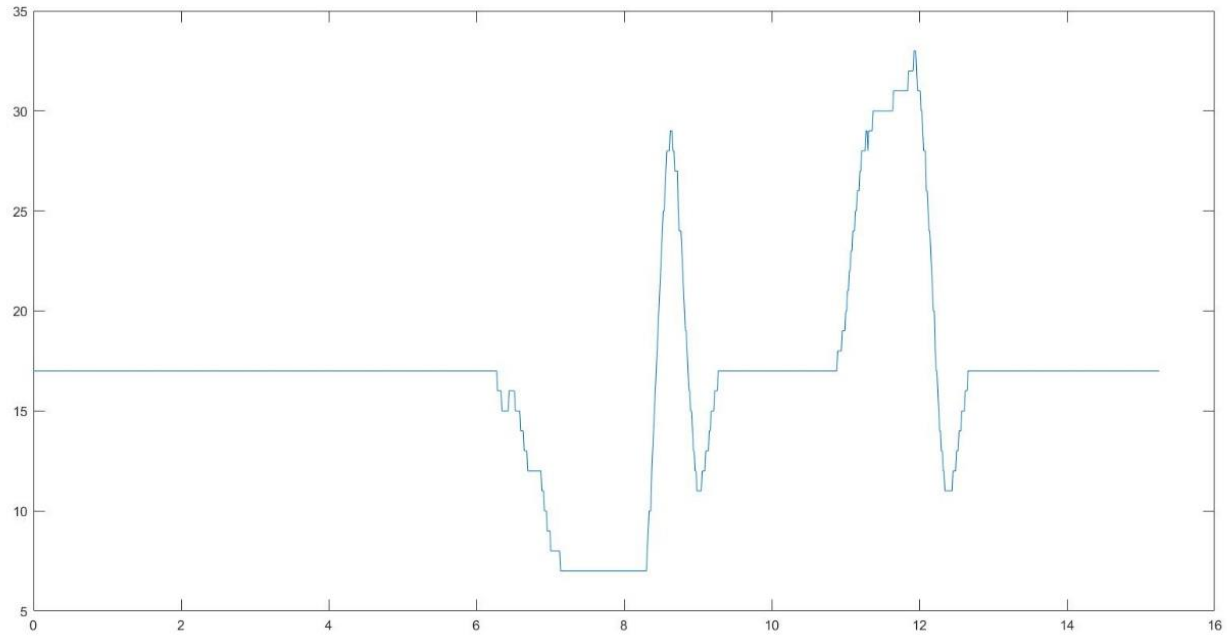


Figura 4.2.1.5: Ejemplo experimental del control del controlador PD gráfica de los rotores 1 y 3 con respecto al tiempo

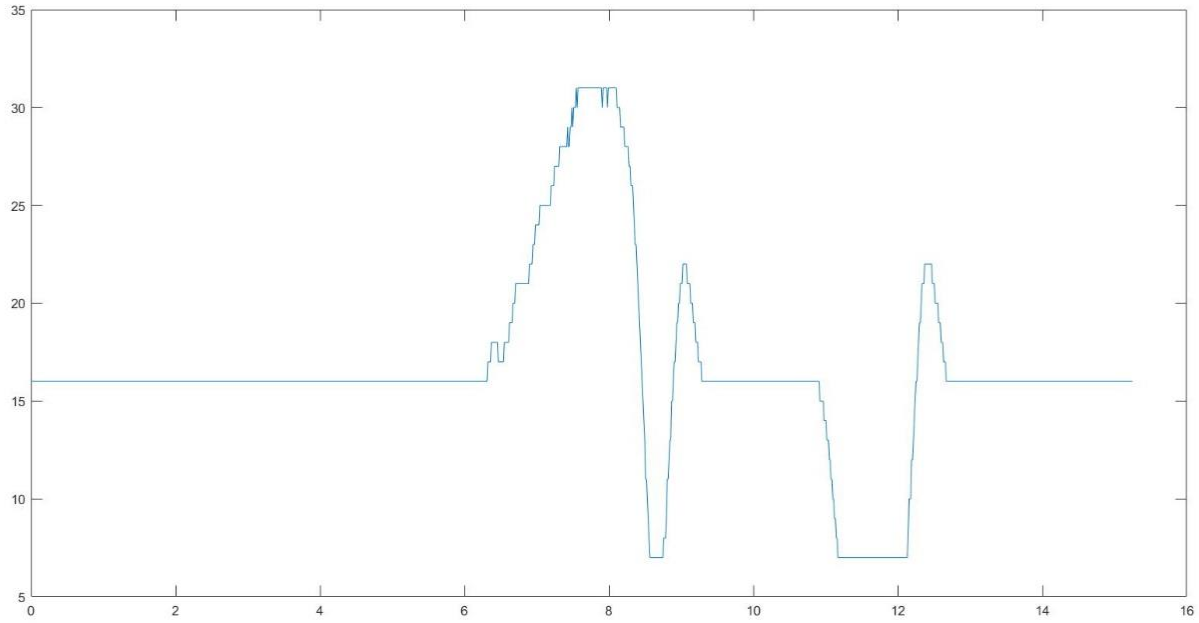


Figura 4.2.1.6: Ejemplo experimental del control del controlador PD gráfica de los rotores 2 y 4 con respecto al tiempo

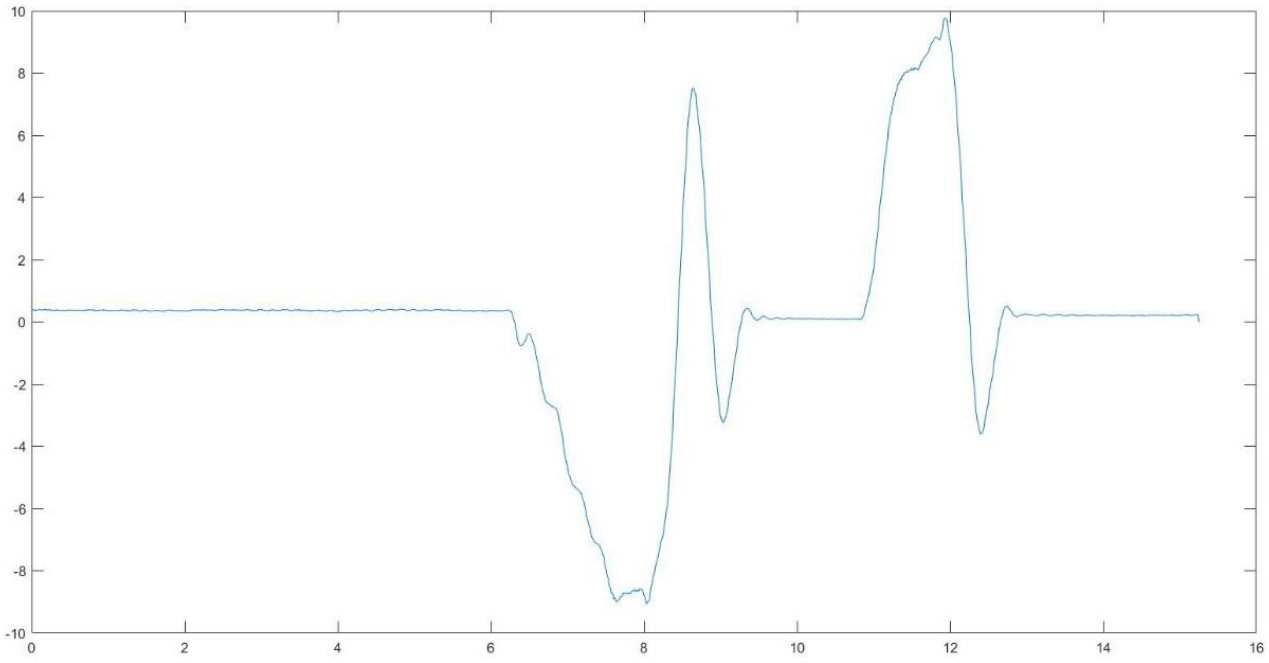


Figura 4.2.1.7: Ejemplo experimental del control del controlador PD gráfica del ángulo-tiempo

Cómo podemos ver en las gráficas adjuntas, cuando hay una modificación en la salida (ángulo que forma el dron con la horizontal y última gráfica), rápidamente actúan los rotores correspondientes para estabilizar el cuadricóptero. Es decir, en la última gráfica podemos ver como hemos introducido una perturbación haciendo que el dron forme un ángulo negativo con respecto al eje horizontal. En este caso y, cómo podemos ver en la primera y segunda gráfica, los rotores 2 y 4 actúan rápidamente incrementando el valor de las rpm de sus rotores y los rotores 1 y 3 decrementando su valor, para conseguir que el dron se estabilice lo más rápido posible dentro de las limitaciones del diseño del controlador.

4.3 CONTROLADOR AVANZADO

Cómo en la parte anterior, esta parte se va a dividir en dos partes, una primera parte de diseño del controlador avanzado y una segunda parte de implementación del mismo.

4.3.1 DISEÑO DEL CONTROLADOR AVANZADO

En general, el diseño de controladores en sistemas de control se puede ver como un problema de diseño de filtros, el controlador PD (implementado en el apartado anterior) sería un filtro pasa-alta que, también, se le conoce como controlador de adelanto (avance) de fase, ya que introduce fase positiva al sistema en un determinado intervalo de frecuencias. Este adelanto de fase produce, en esencia, un mejoramiento razonable en la respuesta transitoria del sistema y también, una mejora con respecto al corrección que introducía el PD. Cómo podemos observar, para el diseño de una red de avance siempre hablamos en

términos de frecuencia y para ello existe una herramienta muy útil que es el diagrama de Bode para ver cómo evoluciona un sistema en términos de la frecuencia.

En términos generales, una red de compensación tanto de adelanto como de atraso (un controlador PI) se define de la siguiente manera:

$$C(s) = \frac{K_C\,(\frac{1}{z}s + 1)}{(\frac{1}{p}s + 1)} \qquad (4.3.1.1)$$

Donde "z" y "p" son respectivamente el cero y el polo que introduce el controlador, siempre y cuando p sea mayor que z un número de veces especificada por el diseño de la red, estaremos hablando de una red de adelanto de fase.

$$p = \alpha \cdot z \qquad (4.3.1.2)$$

Por otra parte, el valor del cero y el polo también viene determinados por el valor de la frecuencia a la que el sistema corta con el valor de amplificación que aporta la red de avance, esto es:

$$\omega_m = \sqrt{z \cdot p} \qquad (4.3.1.3)$$

En el que α viene determinada por el margen de fase (ϕ) que queremos introducir en el sistema, es decir:

$$\sin \Phi = \frac{\alpha - 1}{\alpha + 1} \qquad (4.3.1.4)$$

Partiendo de lo anteriormente descrito, vamos a realizar el diseño de una red de avance para nuestro sistema que, recordemos que, tiene la siguiente pinta:

$$P(s) = \frac{0.06823}{s^2} \qquad (4.3.1.5)$$

Su lugar de las raíces tiene la siguiente forma:

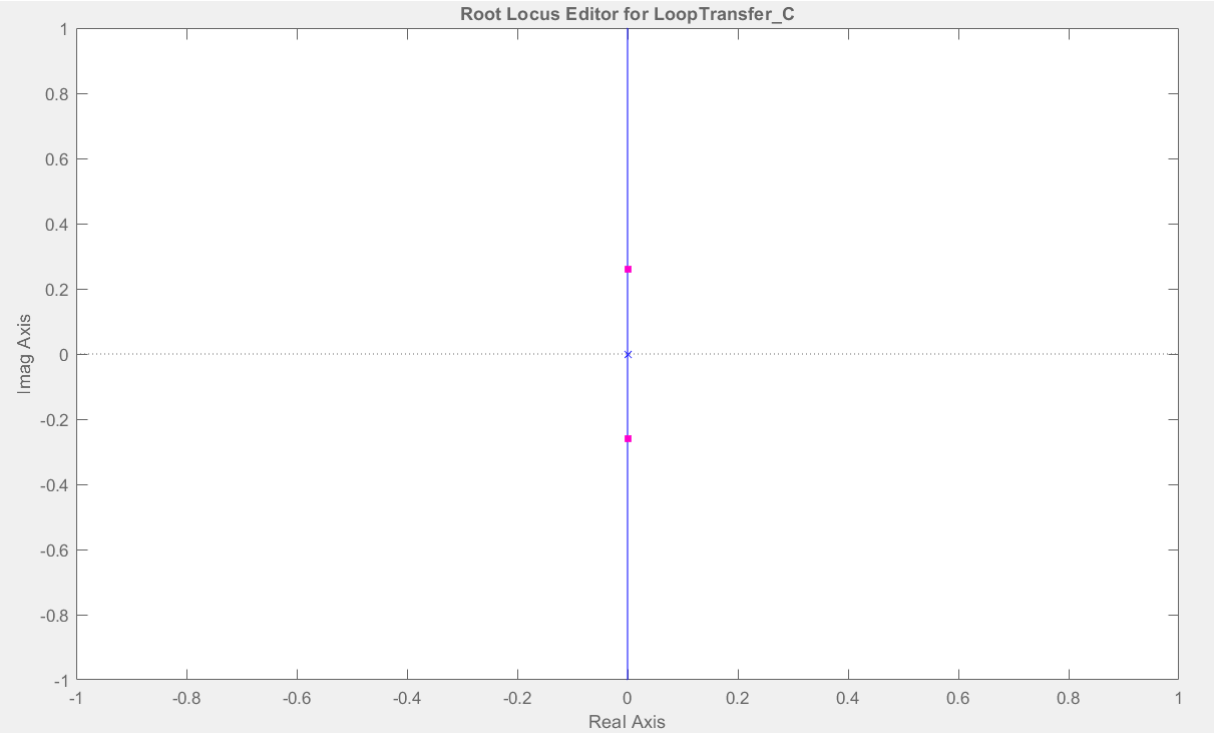


Figura 4.3.1.1: Lugar de las raíces de la planta del sistema

Y en términos de frecuencia:

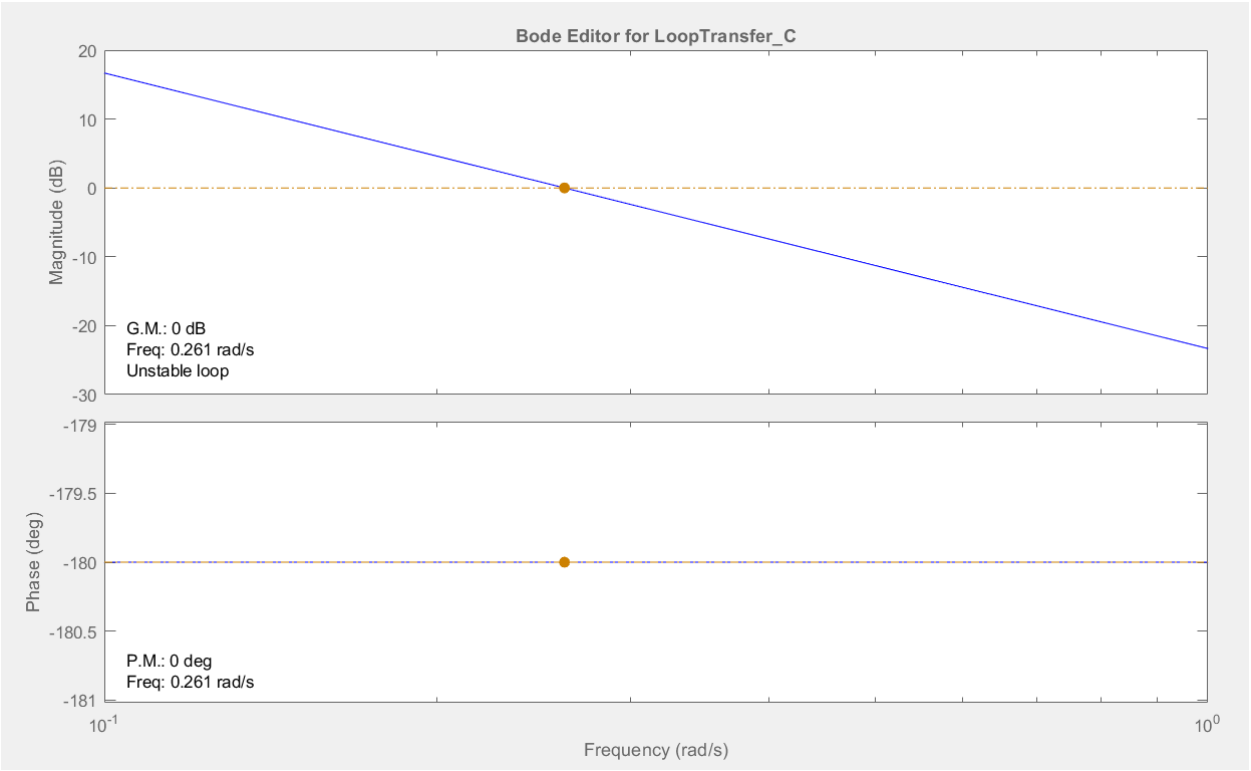


Figura 4.3.1.2: Diagrama de Bode de la planta del sistema

Conociendo ya nuestra planta, la finalidad del diseño de la red de avance es realizar un control sobre nuestra planta para que el dron (planta) a pesar de sufra alguna perturbación se mantenga estable formando un ángulo de aproximadamente de 0º, es decir, que sea capaz de corregir ese error que introduce dicha perturbación y volver a estabilizarse en el ángulo mencionado anteriormente.

Lo primero que vamos a realizar va a ser el cálculo de la ganancia de lazo de nuestro sistema para que tenga una respuesta relativamente rápida. Para esto tenemos que fijar una constante de tiempo del sistema para poder tener una condición para calcular dicha ganancia, para ello fijaremos un valor de la constante de tiempo a 0.25 segundos y por lo tanto obtendremos un valor de la frecuencia de 4 rad/s (ya que ω = 1/τ) y que ha dicha frecuencia, forzar a que corte en 0 dB, ya que, como podemos observar, nuestra planta por si sola

en 0 dB tiene un valor de 0.261 rad/s. Por otro lado, debemos fijar un margen de fase que debe tener el sistema. En término generales, el margen de fase

cuantifica el retardo de fase puro que debería agregarse para alcanzar la misma condición de estabilidad crítica. En nuestro caso, vamos a poner la condición de que el sistema tenga un margen de fase igual o mayor a 70º.

Ya sabiendo nuestra condición de que nuestro sistema corte en 4 rad/s en 0dB, realizaremos el cálculo de la ganancia de lazo abierto:

$$L(s) = \frac{|0.06823 \cdot K_C|}{|4j| \cdot |4j|} = 1 \qquad (4.3.1.6)$$

$$L(s) = \frac{0.06823 \cdot K_C}{16} = 1$$

$$K_C = 234.50$$

Sabiendo ya la ganancia de lazo abierto del sistema, el siguiente paso sería, si no hubiéramos fijado la frecuencia de corte con 0 dB, el cálculo de dicha frecuencia. Como ya la tenemos vamos a calcular el margen de fase actual que tenemos en el sistema para saber lo que debemos aportar para cumplir la especificación de margen de fase. Es decir, el margen de fase de un sistema se calcula como cero menos la cantidad de fase que aporte los polos (si es un polo en el origen aporta -90º), más la cantidad de fase que aporten los ceros (si es un cero en el origen aporta +90º), en este caso es lo siguiente:

$$\lfloor L(4j) = 0 - 180º = -180º \qquad (4.3.1.7)$$

$$MF_{actual} = 180 + \lfloor 4j \qquad (4.3.1.8)$$

$$MF_{actual} = 0º$$

Cómo queremos que nuestro sistema tenga un margen de fase igual o superior a los 70º, debemos de aportar +70º a nuestro sistema para cumplir dicha condición. También, sabemos que cuando un sistema necesita un margen de fase superior a los 60º, es recomendable diseñar una red de avance doble, es decir, dos redes de avance que te aporte cada una la mitad del margen total que quieres aportarle al sistema. En nuestro caso concreto, necesitamos diseñar dos redes de avance que cada una nos aporte 35º. Para ello, haciendo uso de la ecuación 4.3.3, sabiendo ya el valor de la fase que queremos añadir despejaremos el valor de α.

$$\sin 35 = \frac{\alpha - 1}{\alpha + 1}$$

$$\alpha = 3.7$$

Conociendo ya el valor de α, el próximo paso que debemos realizar es el cálculo de cuánto va a amplificar la red de avance nuestro sistema, esto lo calcularemos de siguiente forma:

$$2 \cdot 10 \log_{10} \alpha = 20 \log_{10} 3.7 = 11.4\ dB \qquad (4.3.1.9)$$

Multiplicamos por dos la fórmula ya que estamos diseñando una red doble de avance y entonces, amplifica el doble. Lo siguiente que realizaremos es el cálculo de la frecuencia a la corta el sistema en el valor que hemos calculado de amplificación, ya que esta frecuencia determinará el valor del polo y el cero
de nuestras redes de avance.

$$|L(j\omega)|_{dB} = -11.4\ dB \qquad (4.3.1.9)$$

$$\frac{16}{\omega_m^2} = 10^{\frac{-11.4}{20}}$$

$$\omega_m = 7.71\ rad/s$$

Haciendo uso de la ecuación 4.3.2 e introduciendo dicho valor en la ecuación 4.3.3, obtenemos:

$$\omega_m = \sqrt{\alpha \cdot z^2} \qquad (4.3.1.10)$$

Despejando el valor del cero, tenemos:

$$z = \frac{\omega_m}{\sqrt{\alpha}} = 4$$

Por lo tanto, el valor del polo será

$$p = \alpha \cdot z = 3.7 \cdot 4 = 14.8$$

Finalmente, haciendo uso de la ecuación 4.3.1 y sustituyendo valores, quedaría perfectamente definido nuestra red de avance doble y sería la siguiente:

$$C(s) = \frac{234.50 \cdot (\frac{1}{4}s + 1)^2}{(\frac{1}{14.8}s + 1)^2} \qquad (4.3.1.11)$$

Con el diseño del controlador finalizado, veamos cómo responde el sistema. Para esto, introduciremos una señal escalón a nuestro sistema completo y veremos cómo evoluciona la respuesta de nuestro sistema con respecto al tiempo:

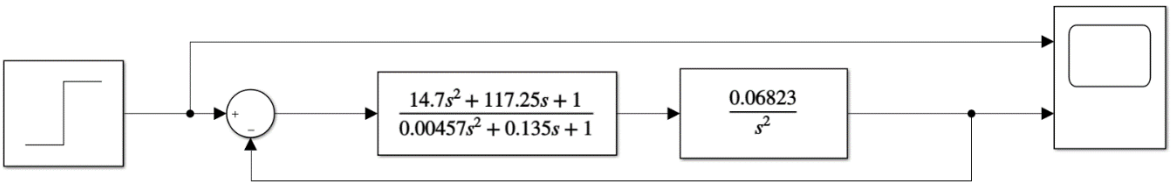


Figura 4.3.1.3: Diagrama de bloques del sistema

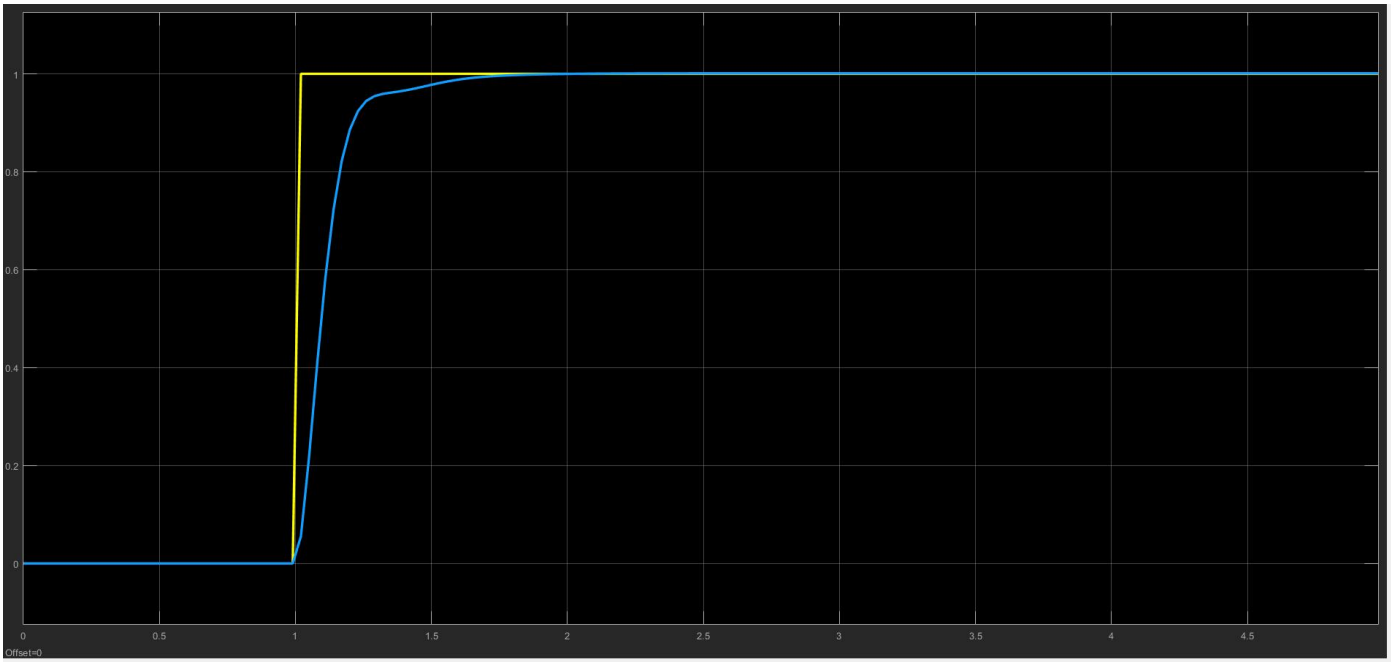


Figura 4.3.1.4: Respuesta del sistema con respecto al tiempo

Como podemos observar, nuestro sistema tiene una respuesta relativamente buena, teniendo un tiempo de establecimiento de alrededor de 1 segundo, es decir, nuestro sistema tardaría 1 segundo en rectificar una posible perturbación en el ángulo que forme el dron con respecto a la horizontal y sea distinto de cero.

Para terminar con el diseño de nuestra de red de avance doble, vamos a discretizar el sistema. Vamos a realizar dicha discretización ya que nuestra planta (dron) está controlada por nuestro controlador de vuelo, es decir, por un microcontrolador y, por lo tanto, dicho microcontrolador tiene una determinada tasa de muestreo y, por ende, tenemos que tener

en cuenta que la señal no es totalmente continua y debemos considerar realizar ciertos cambios para hacer una correcta implementación. Estos cambios serían la discretización del sistema, ya que lo que realmente tendríamos serían una señal discreta. Para realizar la discretización se pueden utilizar varios métodos, yo me he decantado por la transformación bilineal o de Tustin, que se define con la siguiente función:

$$s = \frac{2}{T_m} \cdot \frac{z-1}{z+1} \qquad (4.3.1.12)$$

Donde "s" es la variable del dominio de la Laplace, "z" es la variable de la función discreta y "T_m" es el periodo de muestreo que vamos a utilizar. Entonces, si desarrollamos la ecuación 4.3.9 y le introducimos el cambio mencionado, obtendríamos la siguiente expresión con un periodo de muestreo 0.05 segundos:

$$C(z) = \frac{2057z^2 - 3430z + 1373}{z^2 - 0.9207z + 0.2124} \qquad (4.3.1.13)$$

Como hemos hecho anteriormente, vamos a ver la respuesta del sistema para ahora con el controlador discretizado, para ver como varía la respuesta del sistema.

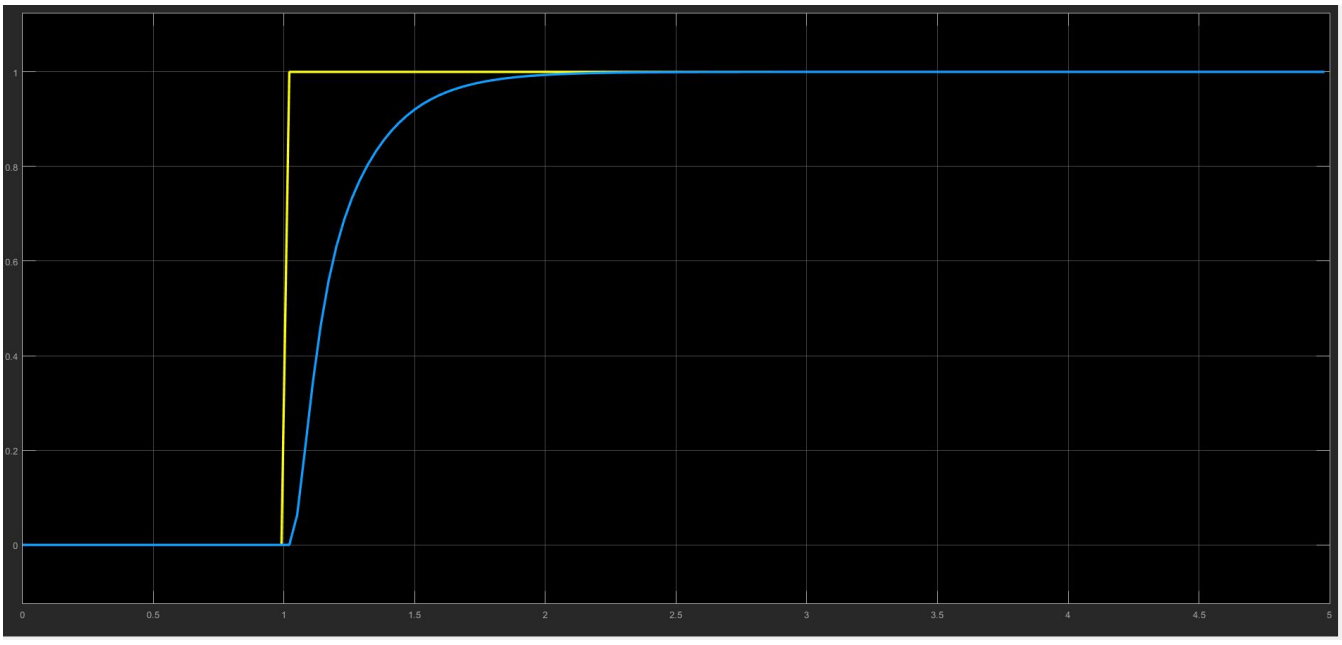


Figura 4.3.1.5: Respuesta del sistema con el controlador discreto

Como podemos observar en la gráfica, la respuesta se ha suavizado y la constante de tiempo sigue siendo la misma, alrededor de 1 segundo. Viendo que nuestro sistema responde bien de forma teórica, vamos a implementar dicho controlador en el código. Para ello, lo primero que debemos de hacer es poner toda función de transferencia en z^{-1}, para ir buscando la transformada inversa de la discretización, es decir:

$$C(z^{-1}) = \frac{2057 - 3430z^{-1} + 1373z^{-2}}{1 - 0.9207z^{-1} + 0.2124z^{-2}} \quad (4.3.1.14)$$

4.3.2 IMPLEMENTACIÓN DEL CONTROLADOR AVANZADO

En esta última parte se realizará la implementación del controlador avanzado, partiendo de la ecuación 4.3.1.14. Sabiendo que el controlador es igual a la entrada que tenga el sistema entre error del propio sistema, podríamos igualar la ecuación anteriormente mencionada a este cociente, es decir:

$$\frac{2057 - 3430z^{-1} + 1373z^{-2}}{1 - 0.9207z^{-1} + 0.2124z^{-2}} = \frac{u}{\varepsilon} \quad (4.3.1.15)$$

Desarrollando esta ecuación obtendríamos:

$$(2057 - 3430z^{-1} + 1373z^{-2})\varepsilon = (1 - 0.9207z^{-1} + 0.2124z^{-2})u$$

$$(2057\varepsilon - 3430\varepsilon z^{-1} + 1373\varepsilon z^{-2}) = (u - 0.9207uz^{-1} + 0.2124uz^{-2}) \quad (4.3.1.16)$$

Realizando la transformada inversa de la discretización

$$2057\varepsilon_K - 3430\varepsilon_{k-1} + 1373\varepsilon_{k-2} = u_K - 0.9207u_{K-1} + 0.2124u_{K-2} \quad (4.3.1.17)$$

Despejando u_K que es lo que vamos buscando, ya que sería nuestra entrada al sistema:

$$u_K = 0.9207u_{K-1} - 0.2124u_{K-2} + 2057\varepsilon_K - 3430\varepsilon_{k-1} + 1373\varepsilon_{k-2} \quad (4.3.1.18)$$

Donde:

- u_K es la entrada que van a recibir los rotores.
- u_{K-1} es la entrada que han recibido los rotores justamente en el instante anterior.

- u_{K-2} es la entrada que han recibido los rotores hace dos instantes.
- ε_K es el error del sistema en el instante actual.
- ε_{K-1} es el error del sistema justamente un instante anterior.
- ε_{K-2} es el error del sistema hace dos instantes.

Con esto hemos determinado que aspecto tendría nuestro controlador y con ello, la entrada que deberían de tener los rotores para poder rectificar una perturbación.

Como se ha realizado en el apartado anterior, terminaremos adjuntando una serie de gráficas de ejemplo de funcionamiento de la red de avance doble.

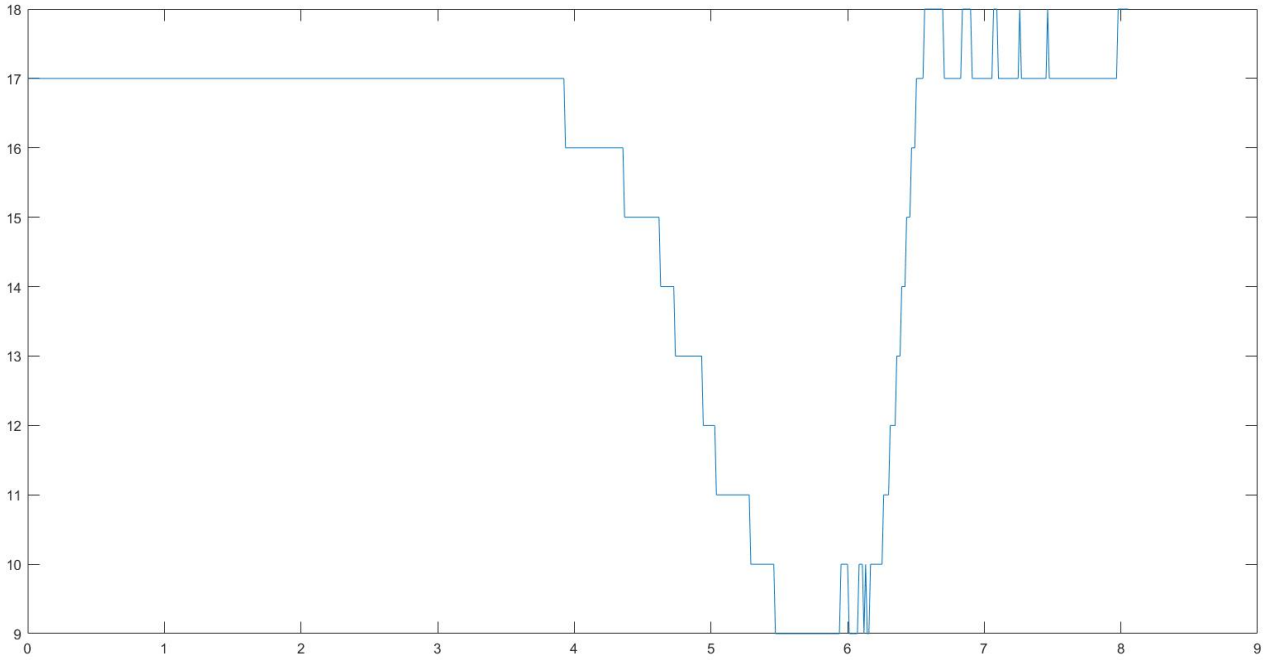


Figura 4.3.2.1: Gráfica experimental del control de la red de avance doble rotores 1 y 3 respecto al tiempo

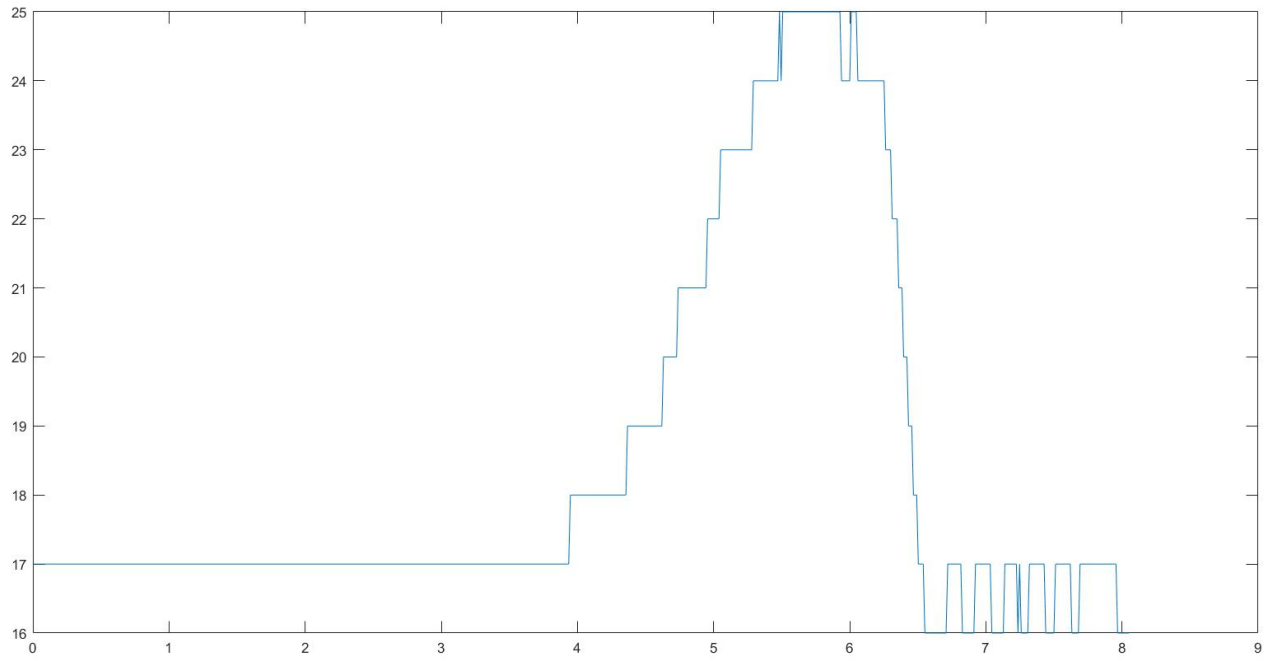


Figura 4.3.2.2: Gráfica experimental del control de la red de avance doble rotores 2 y 4 respecto al tiempo

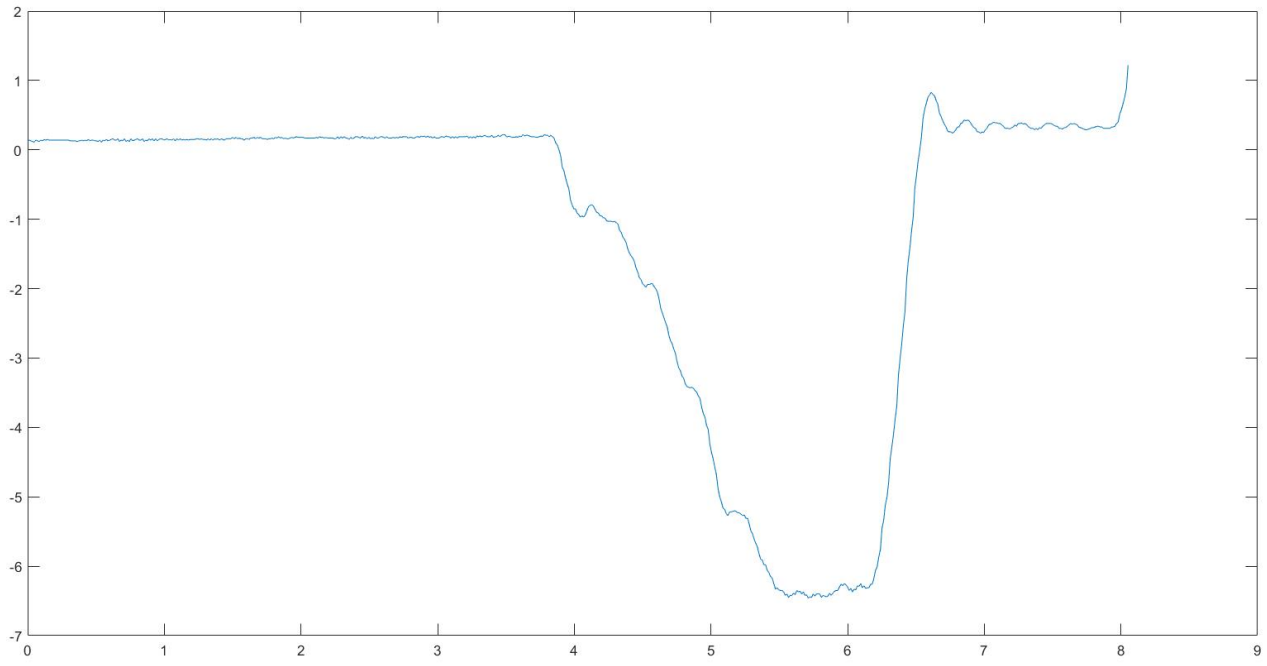


Figura 4.3.2.3: Gráfica experimental del control de la red de avance doble ángulo-tiempo

En términos generales, tiene el mismo funcionamiento que se ha explicado para el controlador PD. Las principales diferencias son que este diseño hace un control más exacto y el tiempo de establecimiento es menos con respecto al controlador proporcional-diferencial, como ya se ha comentado en apartados anteriores. Pero, en las gráficas los rotores hacen la misma función que en el apartado anterior, es decir, cuando introducimos una perturbación con un cierto ángulo, dependiendo de si este es positivo o negativo, unos rotores incrementan su valor y otros los decrementan.

5. HARDWARE

En esta parte, voy a explicar los diferentes componentes que forman nuestro cuadricóptero y, también, el "setup" de dicho vehículo aéreo no tripulado. Como he comentado anteriormente, este TFG parte de un TFM realizado con anterioridad y, por lo tanto, parto con los mismos componentes que se utilizaron en dicho Trabajo Fin de Máster y que son los siguientes:

- Motores: los motores que se utilizan para este tipo de vehículos son motores brushless. Este tipo de motores presenta una serie de ventajas frente a los motores DC con escobillas, entre ellas se pueden destacar una mayor relación velocidad-par, una mejor respuesta dinámica y una mayor eficiencia. Del mismo modo, este tipo de motores presentan también inconvenientes como son su elevado coste y una mayor dificultad a la hora de controlarlos ya que se debe realizar exclusivamente de forma electrónica.

Figura 5.1: Motor brushless

En el mercado se pueden encontrar una amplia gama motores brushless para este tipo de vehículos y se pueden clasificar según el valor de KV. Este valor representa el número de revoluciones por voltio, a menor valor de KV menor será la velocidad, pero aumentará el par generado y a mayor valor de KV ocurriría lo contrario. En general, los motores con mayor KV se suelen usar para vehículos pequeños.

El motor que se ha escogido para el proyecto es del fabricante Eachine, modelo MN2205 de 2300KV. Este motor tiene un empuje máximo de 7;65N, por lo que con los cuatro motores el empuje máximo será de 30,6N.

Figura 5.2: Motor Eachine MN2205 2300KV

Atributo	Valor
KV	2300
Max empuje	7, 65N
Longitud	32, 2mm
Nº de celdas	2 – 4S
Diámetro	27, 9mm
Marco	12*N* 14*P*
Eje	3mm
Hélice	5inch
Peso	0, 245N

Tabla 5.1: Características motor brushless Eachine MN2205 2300KV

- Variador de velocidad: como se ha comentado anteriormente el control de este tipo de motores se realizará de forma electrónica por medio de variadores de velocidad o ESC (del inglés Electric Speed Control). Estos variadores son controladores de Modulación por Ancho de Pulsos (PWM) y se caracterizan en función de la máxima corriente que pueden soportar, a mayor corriente mayor será el precio del ESC.

Figura 5.3: Variador de velocidad (ESC)

A la hora de escoger ESC no solo se ha tenido en cuenta la intensidad máxima a soportar, sino también si el variador debería ser OPTO o BEC. Los variadores BEC cuentan con un regulador de tensión, lo que permite alimentar otros componentes que no soportarían la tensión de la batería, por ejemplo, un servo. Sin embargo, en este caso no sería necesario ya que la placa de distribución de potencia, de la que se hablará más adelante, contará con un regulador de tensión. Por lo tanto, se ha escogido un ESC OPTO, del fabricante Eachine, que soportaría una corriente máxima de 20A.

Figura 5.4: Variador de velocidad Eachine 20A

Atributo	Valor
Corriente	20A
Pico de corriente(10s)	25A
Tensión de entrada	2 – 4S
BEC	NO
Programación	SI
Firmware	Blheli S
Peso	0, 042N
Dimensiones (PCB)	27x12mm

Tabla 5.2: Características del variador de velocidad 20ABLHELIS

- Controlador de vuelo: el controlador de vuelo es el componente principal en un cuadricóptero, ya que será encargado de coordinar al resto de componentes para que todo funcione correctamente. Este controlador recibirá las mediciones de los sensores y enviará las órdenes a los ESC, los cuales se encargarán de acelerar o desacelerar los motores.

  En este proyecto se ha optado por una placa de desarrollo libre. El modelo escogido es Teensy 3.2, aunque se ha programado mediante la plataforma de desarrollo de hardware libre Arduino, el lenguaje utilizado en esta plataforma es un conjunto de librerías las cuales permiten programar en lenguaje C++ de una forma más sencilla, ya que se realiza a un nivel más alto de abstracción.

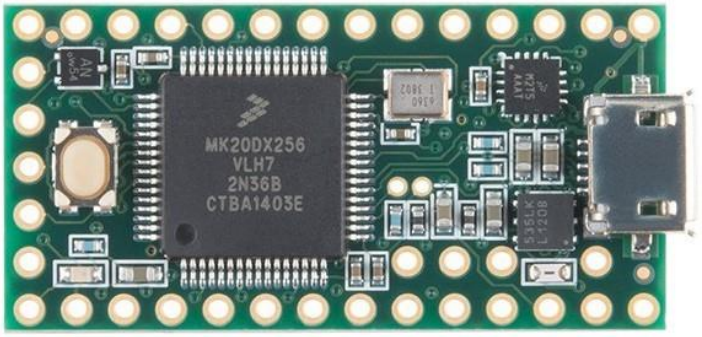


Figura 5.5: Teensy 3.2

La placa Teensy 3.2 cuenta con una conexión micro-USB para programarlo, un botón de reinicio y un microcontrolador ARM Cortex M4, cuyas características se encuentran en la siguiente tabla.

Atributo	Valor
Nombre de la Familia	Kinetis K2x
Tipo de Encapsulado	LWFFP
Tipo de Montaje	Montaje superficial
Conteo de Pines	64
Núcleo del Dispositivo	ARM Cortex M4
Tamaño de la Memoria del Programa	288kB
Frecuencia Máxima	72MHz
Tamaño RAM	66kB
Canales USB	1
Número de Unidades PWM	1
Canales de Convertidor Analógico-Digital	2
Tensión de Alimentación de Funcionamiento Típica	1, 71 - 3, 6V
Temporizadores	2(2 canales)
Dimensiones	10*x*10*x*1, 6mm
Modulación de Ancho de Pulso	1(8 canales)
Tipo de Memoria del Programa	Flash
Número de Canales UART	3
Número de Temporizadores	2
Número de Canales I2C	2
Ancho	10mm
Canales PWM	8
Altura	1, 6mm
Convertidores Analógico-Digital	2x16 bits
Longitud	10mm
Temperatura Máxima de Funcionamiento	+105C
Número de Unidades ADC	2
Número de Canales CAN	1
Arquitectura del Conjunto de Instrucciones	DSP
Temperatura de Funcionamiento Mínima	−40C
Resolución de Convertidor Analógico-Digital	16

Tabla 5.3: Características ARM Cortex M4

- Acelerómetro y giroscopio: se denomina acelerómetro al dispositivo capaz de medir las aceleraciones lineales y angulares de un cuerpo. Estos dispositivos captan las vibraciones convirtiéndolas en una señal eléctrica proporcional a la aceleración momentánea. Los giroscopios, sin embargo, son dispositivos capaces de medir la velocidad angular que

tiene un objeto.

Estos dos sensores se encuentran en dispositivos tan comunes como los smartphone, y, también, en dispositivos de navegación autónoma. En este proyecto se usarán para conocer las aceleraciones y velocidades de giro del cuadricóptero y de esta forma mantener en equilibrio al mismo. En los drones que se comercializan, estos sensores vienen incluidos en el propio controlador de vuelo, sin embargo, para este proyecto el dispositivo que se ha escogido es el sensor MPU6050. Esta unidad de medición inercial o IMU dispone de 6 grados de libertad, combinan- do un acelerómetro de 3 grados de libertad y un giroscopio de 3 GDL también.

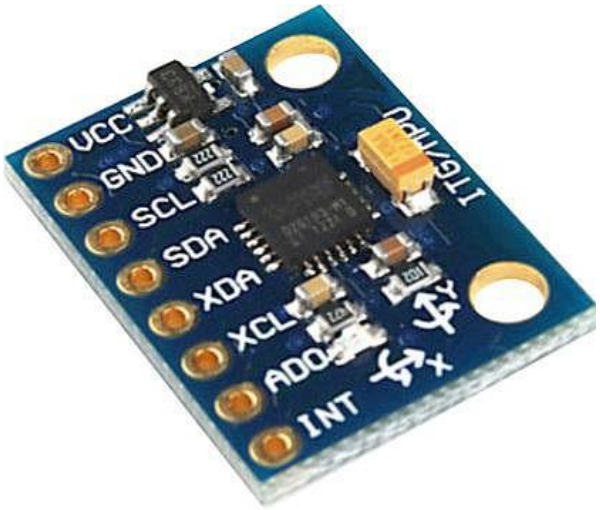


Figura 5.6: Acelerómetro y Giroscopio MPU6050

Atributo	Valor
Fabricante	TDK
Categoría de productor	IMU
Tipo de sensor	6 ejes
Eje sensor	X, Y, Z
Sensibilidad	340LSB/C
Aceleración	10000g
Tipo de salida	Digital
Tipo de interfaz	I2C
Corriente de suministro operativa	3, 9mA
Tensión del suministro Min	2, 375V
Tensión del suministro Max	3, 46V
Temperatura operativa mínima	−40C
Temperatura operativa máxima	+105C
Empaquetado	Cut Tape
Marca	TDK InvenSense
Sensibles a la humedad	SI
Peso	0, 029N

Tabla 5.4: Características acelerómetro y giroscopio MPU6050

- Placa de distribución de potencia: las placas de distribución de potencia o PDB tienen como objetivo alimentar a los distintos elementos que constituyen el dispositivo en el que está integrado, mediante el uso de este dispositivo se prescinde del uso de cables. Para este proyecto se ha seleccionado una placa de distribución de potencia con regulador de tensión, lo que permitirá alimentar el controlador de vuelo a 5V. Además, contará con un terminal XT60 para conectar de forma sencilla y segura la batería a la PDB.

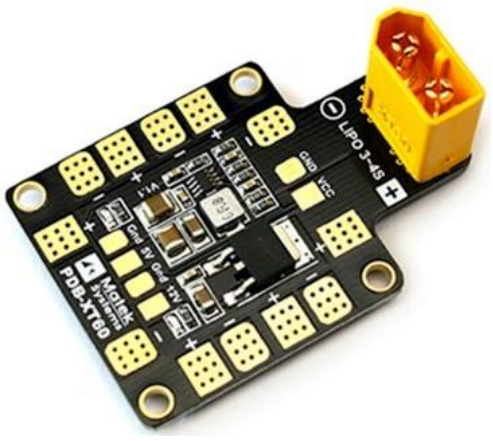


Figura 5.7: Placa de distribución de potencia con BEC y conector XT60

- Hélices: Las hélices tienen como objetivo generar impulso mediante su giro. Vienen determinadas por dos parámetros: el tamaño, el cual se mide de punta a punta, y el grado de inclinación. Ambas características serán directamente proporcionales al empuje, es decir a mayor diámetro e inclinación, mayor será el empuje. En contraposición, el incremento del tamaño y de la inclinación aumentará, también, el consumo de energía, por lo que se debe encontrar un punto de equilibrio entre estos parámetros.

  Por otro lado, se debe tener en cuenta el acabado en las puntas de las hélices, ya que, también, afectará al funcionamiento de las mismas. Se pueden distinguir tres tipos:

  - Acabadas en punta: más eficientes en relación consumo-empuje, proporcionará un mayor tiempo de vuelo, pero menor empuje.
  - Bullnose: son poco eficientes respecto al consumo, sin embargo, otorgan un gran empuje.
  - Hibridas: este tipo se encuentra entre las dos anteriores, consiguiendo un buen equilibrio entre consumo y empuje.

  Por último, se debe tener en cuenta el número de palas, se pueden encontrar de dos, tres y hasta cuatro palas. A un mayor número de palas mayor será el empuje y, por lo tanto, el consumo.

  Para este proyecto se ha escogido unas hélices 5040, es decir, un tamaño de 5” y una inclinación de 40. Además, las puntas tendrán un acabado híbrido y contarán con tres palas, lo que proporcionará una mayor estabilidad.

Figura 5.8: Hélices 5040

- Batería: las baterías utilizadas en drones son del tipo LiPo (Litio y polímero). Se tratan de baterías recargables capaces de almacenar una gran cantidad de energía, y ofrecen una tasa de descarga muy alta. Además, permiten fabricarse en medidas personalizadas.
  Se pueden distinguir tres cualidades en una batería LiPo: el número de celdas (S), la capacidad y la tasa de descarga (C).
  El número de celdas de una batería vendrá determinado por el número S, por ejemplo, una batería 4S dispondrá de cuatro celdas. Este tipo de baterías disponen de un mayor voltaje por celda, hasta 4, 2V cuando se encuentran completamente cargadas.
  La capacidad de la batería se mide en miliamperios-hora (mAh). A mayor mAh más capacidad de carga, sin embargo, cuanto mayor sea la capacidad, más grande y, por tanto, más pesada será la batería.
  La tasa de descarga indicara´ la rapidez con la que se puede descargarse una batería. Por ejemplo, con una capacidad de 1000mAh y una descarga de 1C, la batería descargaría 1A en una hora, si en fuera de 2C descargaría 2A en una hora, por lo que la batería se agotaría en media

hora. Un valor alto de C ofrece una mejor respuesta y rapidez del dron, sacrificando el tiempo de vuelo total.

Para este proyecto, la batería que se ha escogido es una batería LiPo de tres celdas (3S), con una capacidad de 1500mAh y una tasa de descarga de 60C.

Figura 5.9: Batería LiPo 3S 1500mAh 60C

- Módulo de control remoto: para el control remoto de un dron, o de cualquier otro dispositivo, se necesitarán dos componentes, el primero de ellos será un transmisor, el cual emitirá las ordenes, y el segundo será un receptor que se encontrará en el dron y recibirá las órdenes del transmisor. Los módulos de control remoto vendrán definidos por el rango de radiofrecuencia a la que operan y por su alcance de control. Por lo general, en estos tipos de vehículos, el transmisor suele ser un mando, como el que se muestra en la figura 5.10 y el receptor vinculado a dicho mando deberá ser un dispositivo pequeño y ligero, tal y como se observa en la figura 5.10.

Figura 5.10: Transmisor RC y receptor

Sin embargo, para este proyecto se ha optado por algo distinto, buscando un enfoque más didáctico, se ha decidido controlar al dron por medio de un ordenador. Entre las distintas tecnologías inalámbricas (WiFi, Bluetooth,...) se ha optado por la opción LoRa. Esta tecnología utiliza un tipo de modulación en radiofrecuencia como la AM o FM pero patentado por la empresa Semtech. La gran ventaja de este tipo de modulación es que permite la comunicación a largas distancias, manteniendo una gran solidez frente a las interferencias. El módulo que se ha escogido ha sido el modelo, desarrollado por CDSENET, E32-433T30D. Este módulo transmisor-receptor trabaja a una frecuencia de 433MHz y un alcance máximo de 8000m en aire abierto, con máxima potencia y ganancia de antena de 5dBi. Sin embargo, la antena escogida cuenta con una ganancia de 2, 5dBi por lo que alcance no serán los 8000m. En la tabla 4.5 se detallan las características de este módulo.

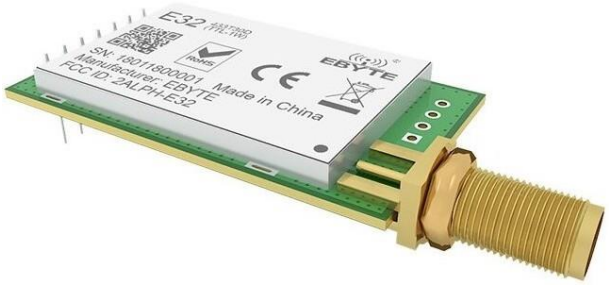


Figura 5.11: Módulo LoRa E32-433T30D

Atributo	Valor
Frecuencia de trabajo	410 ~ 441MHz
Potencia de transmisión	21 ~ 30dBm
Sensibilidad de recepción	−147dBm
Tasa de datos aéreos	0, 3k ~ 19, 2kbps
Distancia de prueba	8000m
Dimensiones	24x43mm
Tipo de antena	SMA-K
Interfaz de comunicación	UART
Encapsulado	DIP
Buffer	512bytes
Alimentación	3, 3 ~ 5, 2V
Nivel de comunicación	3, 3V
Corriente de transmisión	106mA
Corriente de recepción	15mA
Corriente de apagado	4μA
Temperatura de operación	−40 ~ +85C
Pot. max Tx	19 ~ 20dBm

Tabla 5.5: Parámetros E32-433T30D

Con todos los componentes descritos, el autor del TFM diseñó un diseño electrónico y mecánico. El diseño mecánico lo realizó mediante una tecnología de impresión 3D FDM y fue creando pieza por pieza, las piezas que conforman el dron. El resultado final sería:

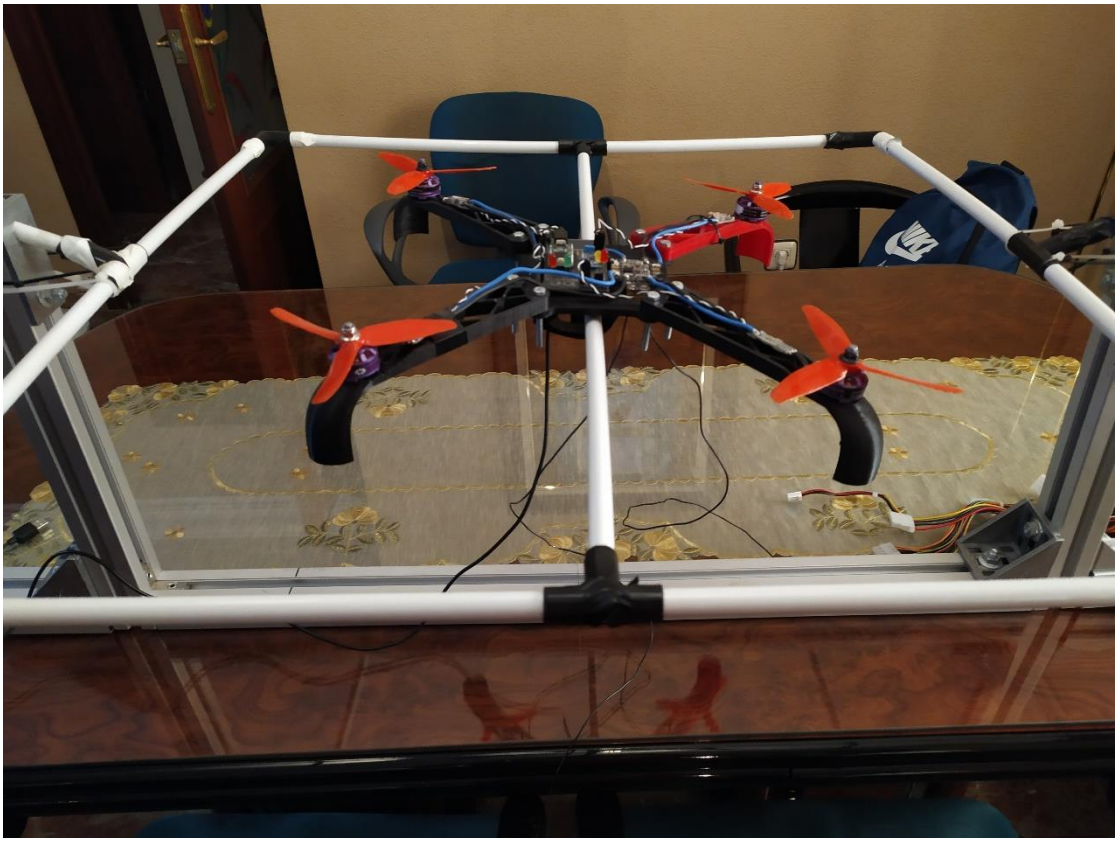


Figura 5.12: Figura del cuadricóptero

Como podemos observar en la foto adjunta, el dron está sobre una plataforma para poder realizar pruebas con respecto al giro del dron respecto a dos de sus ejes. Para este proyecto hemos tenido que restringir otro eje de giro para poder simplificar el cálculo de los diferentes puntos que contiene este documento. He realizado esta modificación en la estructura del cuadricóptero introduciendo un sistema de bridas, para realizar a sujeción e impedir que el dron girase en ese sentido.

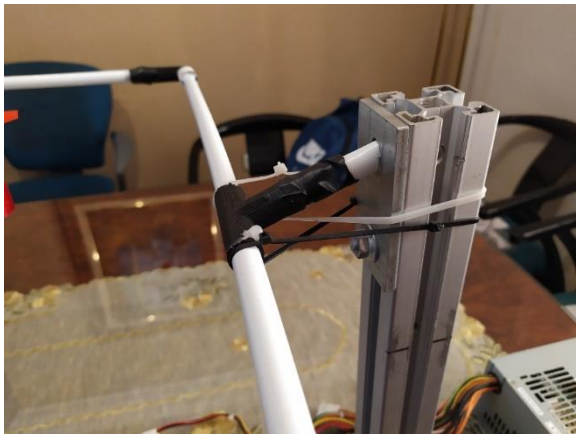


Figura 5.12: Figura del sistema de bridas que impide el giro

6. SOFTWARE

Con respecto al software del sistema lo he obtenido del TFM que he comentado con anterioridad. He obtenido las partes de inicialización de los diferentes componentes y la configuración que necesita cada dispositivo. Yo he realizado la programación de control que necesita el dron para poder estabilizarse. El software se ha desarrollado mediante el lenguaje Arduino, que está basado en C++.

Por otro lado, el software está dividido en diferentes ficheros dependiendo de su función. El programa está formado por cinco ficheros (Principal, Inicializar, Sensor, Control y PWM):

- Principal: encargado de aunar las funciones descritas en el resto de ficheros, se realizan las inicializaciones de todos los componentes (sensor y motores) y la calibración del sensor MPU6050, y se define el ciclo.

- Inicializar: se definen los pines de los leds y de cada uno de los controladores de velocidad, además, se define la función que iniciará los motores.

- Sensor: en este fichero se definen las funciones que inicializan y calibran el sensor MPU6050. También, se encuentran las funciones que realizan la lectura y procesado de los datos proporcionados por el sensor.

- Control: aplicando el control de cada parte, anteriormente diseñado, con la referencia enviada desde el PC y la posición del cuadricóptero, se obtienen las fuerzas y momentos necesarios para alcanzar dicha referencia.

- PWM: conocidas las fuerzas y momentos que deben generar los motores, se calcula el ancho de pulso que debe suministrarse a cada motor. Para ello, se calcula, en primer lugar, la velocidad a la que debe girar cada motor y posteriormente, se obtiene el ancho pulso mediante la caracterización experimental de los motores.

Con respecto a la estructura del programa se define de la siguiente manera:

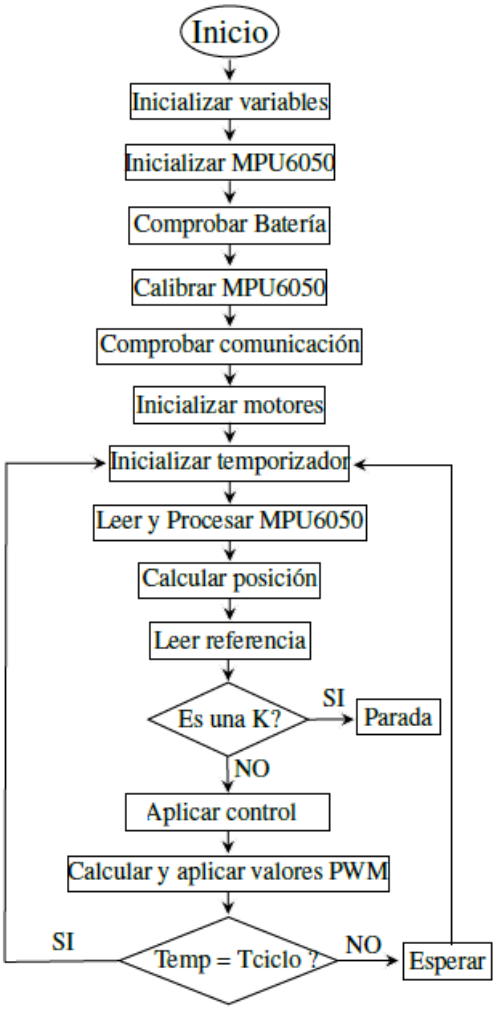


Figura 6.1: Figura de la estructura del programa

7. RESULTADO Y CONCLUSIÓN

7.1 RESULTADO

Como hemos visto en los aparatados de diseño de los controladores, el diseño del controlador avanzado (la red de avance doble), hemos obtenido una respuesta mucho mejor que la obtenida en el diseño del controlador proporcional derivativo, ya que hemos mejorado el tiempo de establecimiento (de 2 segundos a 1 segundo) y hemos suavizado la señal eliminando ese sobreoscilamiento que teníamos en el diseño del primer controlador. Lo que nos quiere decir esto, es que para eliminar el mismo error un controlador tardaría el doble de tiempo que el otro.

También decir, que evidentemente el diseño del primer controlador ha sido mucho más sencillo que del segundo, ya que tanto el proceso como en número de variables que intervienen son más simples. En general, en ambos diseños no he tenido dificultades.

Donde si tuve dificultades fue en el proceso de identificación de nuestro sistema, ya que, a pesar de nuestros intentos, por varios métodos de cálculo, no obtuvimos lo que queríamos ya que la componente no lineal que actúa debido a cómo está dispuesto el dron en la estructura, impide un correcto y sencillo cálculo del sistema. Porque cabe recordar que uno de los objetivos del TFG era diseñar un método de identificación sencillo para que los alumnos de 3º o 4º pudiesen calcular la forma del sistema sin complicaciones. Esto me involucra a que tuve que diseñar una interfaz en Processing que finalmente no he podido utilizarla para eso.

Figura 7.1: Interfaz desarrollada en Processing

En el que, como podemos observar, en tiempo real representaba tanto el valor de los rotores como el valor del ángulo que formaba el dron. Otra funcionalidad de la interfaz, era poder cambiar los valores de entrada de la función seno en tiempo, es decir, conforme se cambiaban dichos valores se recalculaban los valores y, por consiguiente, obtendríamos una gráfica diferente.

Por último, voy a insertar dos códigos QR donde demuestro el control tanto del controlador Proporcional-Diferencial como de la Red de Avance Doble.

Figura 7.2: QR controlador PD

Figura 7.3: QR red de avance doble

https://drive.google.com/file/d/10L9CWupuriPFaN-GeBWDkulCVJ8Ce-YK/view?usp=sharing(Controlador PD)
https://drive.google.com/file/d/1_SEYV9YAEbwfmD1vVpYtb765xY2TTnRL/view?usp=sharing (Red Avance)

7.2 CONCLUSIÓN

Este TFG me ha servido para avanzar y aprender mucho sobre un tema que era totalmente desconocido para mí. Al principio me costó bastante familiarizarme con este tipo de sistema, sus variables, una programación que no había realizad yo (que no es sencilla, ni mucho menos),… Pero con tiempo y dedicación fui entendiendo cómo funcionaba y para que servía cada parte de la programación y los componentes del sistema.

En mi opinión, este trabajo va a ayudar mucho a los alumnos y alumnas a comprender mucho mejor lo que es controlar un sistema, ya que lo pueden ver físicamente y aparte, que el tema de los drones y UAVs están a la orden del día y un gran parte de la población tiene un dron en su casa. Ver como funciona y como se pueden controlar este tipo de dispositivos con un controlador no muy complicado, ayudará a que amplíen su conocimiento en este campo y en campo de control de sistemas en general.

A. ANEXO

A.1 CÓGIDO PROGRAMA

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif
#include <SoftwareSerial.h>
#define usCiclo 12000  // Ciclo de ejecucion de software en microsegundos
#define PI 3.14159265358979323846264338327795
#define N 6
#define G 1670.1325//16384.0//1671.8367 // el valor es 16384/9.8
#define Gy 65.5
#define WR 250000 //161495558.567431
#include <PWMServo.h>

////////////////////////////////

int mEstable = 1;

//======== MOTORES ==========
PWMServo ESC1;
PWMServo ESC2;
PWMServo ESC3;
PWMServo ESC4;
int esc1, esc2, esc3, esc4;
float wr1_2, wr2_2, wr3_2, wr4_2, wr1, wr2, wr3, wr4;

//TIEMPOS
double tiempo_ejecucion, loop_timer, loop_timer1, esc_loop_timer, tiempo_ON, tiempo_1;
bool M_tiempo = false;

//========  VARIABLES ============
//float refx, refy, refz, refphi, reftheta, refpsi, refvx, refvy, refvz, refwphi, refwtheta, refwpsi;
String referencia = "";

float l_k[N] = {0, 0, 0, 0, 0, 0}; //lectura actual; va a contener ax ay az gx gy gz
float l_k1[N] = {0, 0, 0, 0, 0, 0}; //lectura en instante anterior
float lf_k[N] = {0, 0, 0, 0, 0, 0}; //lectura filtrada actual
float lf_k1[N] = {0, 0, 0, 0, 0, 0}; //lectura filtrada anterior

float lff_k[N] = {0, 0, 0, 0, 0, 0}; //lectura filtrada filtrada actual
float lff_k1[N] = {0, 0, 0, 0, 0, 0}; //lectura filtrada filtrada anterior

float x[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
float x_old[12] = {0,0,0,0,0,0,0,0,0,0,0,0};

float xs[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
float xs_old[12] = {0,0,0,0,0,0,0,0,0,0,0,0};

int16_t ax, ay, az, gx, gy, gz;

//===== MPU6050 =====
#define gyro_address 0x68
float angulo_pitch, angulo_roll, angulo_yaw, yawGyro, pitchGyro, rollGyro, angle_pitch_acc, angle_roll_acc, temperature;
int emerg, cal_int;

MPU6050 mpu;
#define OUTPUT_READABLE_YAWPITCHROLL
#define OUTPUT_READABLE_REALACCEL
#define OUTPUT_READABLE_WORLDACCEL

#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
```

```
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };


// ========= INTERRUPT DETECTION ROUTINE ===============

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}

//======== PARADA =========

bool KO = false;
char interrupcion;
String parada;

////////////////////////////////

void setup() {
  inicializar();
  init_MPU6050();
  init_Motores();
  for (int i = 0; i < 18; i++){  // This is where I want to flush any remaing data
   byte discard = Serial.read();
  }
}

////////////////////////////////

void loop() {
  M_tiempo = false;
  loop_timer = micros();
  MPU_6050();
  PWM();
  if (micros() - loop_timer > usCiclo + 50) Serial.println("Max Time"), digitalWrite(13,HIGH);
  //if (M_tiempo == true) Serial3.println(micros() - loop_timer);
  while (micros() - loop_timer < usCiclo);
  tiempo_ejecucion = (micros() - loop_timer) / 1000000;
}
```

Figura A.1.1: Código programa principal

```
void inicializar(){
//  TWBR = 24;
  KO = false;

  //LEDS

  pinMode(A2, OUTPUT);  // Led naranja (bateria)
  pinMode(11, OUTPUT);  // Led verde
  pinMode(10, OUTPUT);  // Led rojo (IMU error)
  pinMode(13, OUTPUT);  // Led rojo (tiempo excedido)

  // Modulo LORA

  Serial.begin(115200);
  //Serial.println("Inicializando");

  Serial.println("Introduce S para iniciar");
    char GO = Serial.read();
    while (GO != 'S'){
      GO = Serial.read();
      delay(100);
    }

    // Motores

  ESC1.attach(3,1000,2000); //(pin,min,max)
  ESC2.attach(5,1000,2000); //(pin,min,max)
  ESC3.attach(4,1000,2000); //(pin,min,max)
  ESC4.attach(6,1000,2000); //(pin,min,max)

  digitalWrite(13, LOW);
  digitalWrite(11, HIGH);
}
void init_Motores(){
  Serial.println("Introduce M para iniciar motores");
  char MOT = Serial.read();
  while (MOT != 'M'){
    MOT = Serial.read();
    delay(100);
  }

  int potValue = map(1023,0,1023,0,180);
  ESC2.write(potValue);
  delay(10);
  ESC1.write(potValue);
  delay(10);
  ESC3.write(potValue);
  delay(10);
  ESC4.write(potValue);
  delay(1500);

  int potValue2 = map(0,0,1023,0,180);
  ESC1.write(potValue2);
  delay(10);
  ESC2.write(potValue2);
  delay(10);
  ESC3.write(potValue2);
  delay(10);
  ESC4.write(potValue2);
  delay(3000);

 Serial.println("Introduce C para comenzar");
  char comienzo = Serial.read();
   while (comienzo != 'C'){
    comienzo = Serial.read();
    delay(100);
   }

}
```

Figura A.1.2: Código inicializar

```
void init_MPU6050() {
   // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // Calibration Time: generate offsets and calibrate our MPU6050
        mpu.CalibrateAccel(6);
        mpu.CalibrateGyro(6);
        mpu.PrintActiveOffsets();
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
        Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
        Serial.println(F(")..."));
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}
```

```
void MPU_6050() {
  emerg = 0;

 // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
//    while (!mpuInterrupt && fifoCount < packetSize) {
//        if (mpuInterrupt && fifoCount < packetSize) {
//          // try to get out of the infinite loop
//          fifoCount = mpu.getFIFOCount();
//        }
//    }
    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();
  if(fifoCount < packetSize){
        //Lets go back and wait for another interrupt. We shouldn't be here, we got an interrupt from another event
      // This is blocking so don't do it   while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
  }
    // check for overflow (this should never happen unless our code is too inefficient)
    else if ((mpuIntStatus & (0x01 << MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) || fifoCount >= 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
      //  fifoCount = mpu.getFIFOCount();  // will be zero after reset no need to ask
        //Serial.println(F("FIFO overflow!"));

    // otherwise, check for DMP data ready interrupt (this should happen frequently)
    } else if (mpuIntStatus & (0x01 << MPU6050_INTERRUPT_DMP_INT_BIT)) {
        // read a packet from FIFO
  while(fifoCount >= packetSize){ // Lets catch up to NOW, someone is using the dreaded delay()!
    mpu.getFIFOBytes(fifoBuffer, packetSize);
    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
    fifoCount -= packetSize;
  }

  mpu.dmpGetQuaternion(&q, fifoBuffer);
  mpu.dmpGetGravity(&gravity, &q);
  mpu.getRotation(&gx, &gy, &gz); // obtener velocidades de giro
  mpu.dmpGetYawPitchRoll(ypr, &q, &gravity); // obtener angulos Pitch Roll Yaw
  mpu.dmpGetAccel(&aa, fifoBuffer);
  mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity); //obtener aceleraciones
  mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
    }

  //asignar_lectura(l_k,gx,gy,gz,aaReal); // asignar aceleraciones y velocidades de giro al vector l_k
  asignar_lectura(l_k,gx,gy,gz,aaWorld); // asignar aceleraciones y velocidades de giro al vector l_k
  filtra_lecturas(lf_k, lf_kl, l_k, l_kl); //filtramos los valores
//  filtraHigh_lecturas_Acc(lff_k, lff_kl, lf_k, lf_kl); // filtramos los valores de aceleracion alta frecuencia

  integrar_lecturas(xs, lf_k, ypr, tiempo_ejecucion); // integramos valores leidos
  filtraHigh_lecturas(x, x_old, xs, xs_old); // filtramos los valores de velocidad
  filtraHigh_lecturas_2(x, x_old, xs, xs_old); // filtramos los valores de posicion
 // copia_vector(xs,x,12);
  asignar_variables_estado(x, xs, tiempo_ejecucion); // obtenemos las variables de estado

  // ===== PRINTS =======

  //velocidades_filt_sinfil(); // comparamos los valores de velocidad filtrados y no filtrados
  //aceleraciones_filt_sinfil(); // comparamos los valores de aceleracion filtrados y no filtrados
  //mostrar_aceleraciones_xyz(); // mostramos las aceleraciones ax ay az
  //mostrar_posicion_xyz(); // mostramos las posiciones x y z
  //mostrar_angulos();  //loop_timer; // mostramos los angulos pitch roll yaw
  //mostrar_velocidades_xyz(); // mostramos las velocidades vx vy vz
  //mostrar_velocidad_giro(); // mostramos las velocidades gx gy gz
  //mostrar_angulos_giro();
  mostrar_angulo_velocidad_angular();
  //Serial.println();
```

```
  // ===== ACTUALIZACIONES =======

  copia_vector(l_k, l_kl, N); //actualizamos los valores -> vector_pasado = vector_actual
  copia_vector(lf_k, lf_kl, N);
  copia_vector(xs, xs_old, 12);
  copia_vector(x, x_old, 12);

  //delay(100);


 //Serial.println(micros() - loop_timer);
}

//////////////////////////////////////////
// ============= FUNCIONES ==============//
//////////////////////////////////////////

void asignar_lectura(float* l,int16_t gx,int16_t gy,int16_t gz, VectorInt16 aaReal)
{
  // funcion que captura los valores del sensor, elimina los offsets y los escala

  l[0] =  aaReal.x * (9.81/16384.0);
  l[1] =  aaReal.y * (9.81/16384.0);
  l[2] =  aaReal.z * (9.81/16384.0);
  l[3] =  gx * (250.0/32768.0)*(M_PI/180);
  l[4] =  gy * (250.0/32768.0)*(M_PI/180);
  l[5] =  gz * (250.0/32768.0)*(M_PI/180);
}

void actualizar (float* v_new, float* v_old,int len){
  for (int i = 0; i < len; i++){
    v_old[i] = v_new[i];
  }
}

void copia_vector(float* src, float* dst, int len) {
  // funcion auxiliar para copiar un vector en otro
   memcpy(dst, src, sizeof(src[0])*len);
}

float filtra_lowlevel(float lectura_filtrada_anterior, float lectura_actual, float lectura_anterior){
  // funcion para implementar un filtro que se aplica directamente a escalares
  float b[2] = {0.863271264002680,  0.863271264002680};//{0.6625, 0.6625}; //{0.62, 0.62};
  float a[2] = {1.0,  0.726542528005361}; //{1.0, 0.3249}; //{1.0, 0.2401};
  float lectura_filtrada_actual = b[0] * lectura_actual + b[1] * lectura_anterior - a[1] * lectura_filtrada_anterior;
  return lectura_filtrada_actual;
}

void filtra_lecturas(float* vector_filtrado_actual, float* vector_filtrado_pasado, float* vector_lecturas_actual, float* vector_lecturas_pasadas){
  // funcion para filtrar todas las componentes de los vectores
  for(int i=0; i<N; i++)
  {
    vector_filtrado_actual[i] = filtra_lowlevel(vector_filtrado_pasado[i], vector_lecturas_actual[i], vector_lecturas_pasadas[i]);
  }
}

void integrar_lecturas(float* x, float* l, float* ypr, double tiempo_ejecucion){
  x[6] += 0; //l[0]*tiempo_ejecucion; // velocidad x
  x[7] += 0; //l[1]*tiempo_ejecucion; // velocidad y
  x[8] += 0; //l[2]*tiempo_ejecucion; // velocidad z

  x[0] += 0; //x[6]*tiempo_ejecucion; // posicion x
  x[1] += 0; //x[7]*tiempo_ejecucion; // posicion y
  x[2] += 0; //x[8]*tiempo_ejecucion; // posicion z

  x[3] = ypr[2];//*(180/PI); // angulo pitch
  x[4] = ypr[1];//*(180/PI); // angulo roll
  x[5] = ypr[0];//*(180/PI); // angulo yaw

  x[9] = l[3]; // velocidad gx
  x[10] = l[4]; // velocidad gy
  x[11] = l[5]; // velocidad gz
}
```

```
float filtra_highlevel(float lectura_filtrada_anterior, float lectura_actual, float lectura_anterior){
  // funcion para implementar un filtro que se aplica directamente a escalares
  float b[2] = {0.9992,  -0.9992,}; //{0.62, 0.62};
  float a[2] = {1,  -0.9984}; //{1.0, 0.2401};
  float lectura_filtrada_actual = b[0] * lectura_actual + b[1] * lectura_anterior - a[1] * lectura_filtrada_anterior;
  return lectura_filtrada_actual;
}

void filtraHigh_lecturas(float* vector_filtrado_actual, float* vector_filtrado_pasado, float* vector_lecturas_actual, float* vector_lecturas_pasadas){
  // funcion para filtrar todas las componentes de los vectores
  for(int i=6; i<9; i++) // de 6 a 9 para que unicamente filtre vx vy vz
  {
    vector_filtrado_actual[i] = filtra_highlevel(vector_filtrado_pasado[i], vector_lecturas_actual[i], vector_lecturas_pasadas[i]);
  }
}

float filtra_highlevel_2(float lectura_filtrada_anterior, float lectura_actual, float lectura_anterior){
  // funcion para implementar un filtro que se aplica directamente a escalares
  float b[2] = {0.9984,  -0.9984}; //{0.62, 0.62};
  float a[2] = {1,  -0.9969}; //{1.0, 0.2401};
  float lectura_filtrada_actual = b[0] * lectura_actual + b[1] * lectura_anterior - a[1] * lectura_filtrada_anterior;
  return lectura_filtrada_actual;
}

void filtraHigh_lecturas_2(float* vector_filtrado_actual, float* vector_filtrado_pasado, float* vector_lecturas_actual, float* vector_lecturas_pasadas){
  // funcion para filtrar todas las componentes de los vectores
  for(int i=0; i<3; i++) // de 6 a 9 para que unicamente filtre vx vy vz
  {
    vector_filtrado_actual[i] = filtra_highlevel_2(vector_filtrado_pasado[i], vector_lecturas_actual[i], vector_lecturas_pasadas[i]);
  }
}

float filtra_highlevel_Acc(float lectura_filtrada_anterior, float lectura_actual, float lectura_anterior){
  // funcion para implementar un filtro que se aplica directamente a escalares
  float b[2] = {0.9999,  -0.9999}; //{0.62, 0.62};
  float a[2] = {1,  -0.9998}; //{1.0, 0.2401};
  float lectura_filtrada_actual = b[0] * lectura_actual + b[1] * lectura_anterior - a[1] * lectura_filtrada_anterior;
  return lectura_filtrada_actual;
}

void filtraHigh_lecturas_Acc(float* vector_filtrado_actual, float* vector_filtrado_pasado, float* vector_lecturas_actual, float* vector_lecturas_pasadas){
  // funcion para filtrar todas las componentes de los vectores
  for(int i=0; i<3; i++) // de 6 a 9 para que unicamente filtre vx vy vz
  {
    vector_filtrado_actual[i] = filtra_highlevel_Acc(vector_filtrado_pasado[i], vector_lecturas_actual[i], vector_lecturas_pasadas[i]);
  }
}


void asignar_variables_estado(float* x, float* xs, double tiempo_ejecucion){
//  x[0] = 0.9992*xs[0]+ (-0.9992)*xs_old[0] - (-0.9984)*x_old[0] ;// + x[6]*tiempo_ejecucion; // posicion x
//  x[1] = 0.9992*xs[1]+ (-0.9992)*xs_old[1] - (-0.9984)*x_old[1];// + x[7]*tiempo_ejecucion; // posicion y
//  x[2] = 0.9992*xs[2]+ (-0.9992)*xs_old[2] - (-0.9984)*x_old[2];// + x[8]*tiempo_ejecucion; // posicion z
  x[0] = x[0];// + x[6]*tiempo_ejecucion; // posicion x
  x[1] = x[1];// + x[7]*tiempo_ejecucion; // posicion y
  x[2] = x[2];// + x[8]*tiempo_ejecucion; // posicion z
  x[3] = xs[3]*(180/PI); // angulo pitch
  x[4] = xs[4]*(180/PI); // angulo roll
  x[5] = xs[5]*(180/PI); // angulo yaw
  x[6] = x[6]; // velocidad x
  x[7] = x[7]; // velocidad y
  x[8] = x[8]; // velocidad z
  x[9] = xs[9]; // velocidad gx
  x[10] = xs[10]*(60/(2*PI)); // velocidad gy
  x[11] = xs[11]; // velocidad gz
}
```

```
void aceleraciones_filt_sinfil(){
  Serial.print(lf_k[0]); Serial.print("\t");
  Serial.print(lf_k[1]); Serial.print("\t");
  Serial.print(lf_k[2]); Serial.print("\t");
  Serial.print(l_k[0]); Serial.print("\t");
  Serial.print(l_k[1]); Serial.print("\t");
  Serial.print(l_k[2]); Serial.print("\t");
}

void velocidades_filt_sinfil(){
  Serial3.print(x[6]); Serial3.print("\t");
  Serial3.print(x[7]); Serial3.print("\t");
  Serial3.print(x[8]); Serial3.print("\t");
  Serial3.print(xs[6]); Serial3.print("\t");
  Serial3.print(xs[7]); Serial3.print("\t");
  Serial3.print(xs[8]); Serial3.print("\t");
}


void mostrar_aceleraciones_xyz(){
  Serial.print(lf_k[0]); Serial.print("\t");
  Serial.print(lf_k[1]); Serial.print("\t");
  Serial.print(lf_k[2]); Serial.print("\t");
}

void mostrar_velocidades_xyz(){
  Serial3.print(x[6]); Serial3.print("\t");
  Serial3.print(x[7]); Serial3.print("\t");
  Serial3.print(x[8]); Serial3.print("\t");
}

void mostrar_posicion_xyz(){
  Serial.print(x[0]); Serial.print("\t");
  Serial.print(x[1]); Serial.print("\t");
  Serial.print(x[2]); Serial.print("\t");
}
void mostrar_angulos(){
  Serial.print(x[3]); Serial.print("\t");
  Serial.print(x[4]); Serial.print("\t");
  Serial.print(x[5]); Serial.print("\t");
  //Serial.print(loop_timer); Serial.print("\t");
}

void mostrar_velocidad_giro(){
  Serial.print(x[9]); Serial.print("\t");
  Serial.print(x[10]); Serial.print("\t");
  Serial.print(x[11]); Serial.print("\t");
}

void mostrar_angulo_velocidad_angular(){
  Serial.print(x[4]); Serial.print('\t');
  Serial.print(x[10]); Serial.print('\t');
}
```

Figura A.1.3: Código sensor MPU6050

A.2 CÓGIDO CONTROLADOR PD

La estructura y el programa en sí es el mismo, lo único que cambia es el código desarrollado en la parte PWM.

```
void PWM(){
  tiempo_1 = micros();

  wr1 = 2500+(234.5*x[4])-(117.25*x[10]);
  wr3 = wr1;
  wr2 = 2500-(234.5*x[4])-(117.25*x[10]);
  wr4 = wr2;

  esc1 = (wr1-71.989)/25.162;
  if (esc1<40){
    esc1 = 40;
  }
  if (esc1>1000){
    esc1 = 1000;
  }
  esc2 = (wr2-71.989)/25.162;
  if (esc2<40){
    esc2 = 40;
  }
  if (esc2>1000){
    esc2 = 1000;
  }
  esc3 = (wr3-71.989)/25.162;
  if (esc3<40){
    esc3 = 40;
  }
  if (esc3>1000){
    esc3 = 1000;
  }
  esc4 = (wr4-71.989)/25.162;
  if (esc4<40){
    esc4 = 40 ;
  }
  if (esc4>1000){
    esc4 = 1000;
  }
```

Figura A.2.1: Código PWM controlador PD 1

```
  esc1 = map(esc1,0,1023,0,180);
  esc2 = map(esc2,0,1023,0,180);
  esc3 = map(esc3,0,1023,0,180);
  esc4 = map(esc4,0,1023,0,180);

  ESC1.write(esc1);
  ESC2.write(esc2);
  ESC3.write(esc3);
  ESC4.write(esc4);

  mostrar_wr();
  mostrar_esc();


  //ONled();
  //VBat();

  tiempo_ON = micros() - tiempo_1;
  if (tiempo_ON > 900) digitalWrite(13, HIGH);
}
void mostrar_wr(){
  Serial.print(wr1);Serial.print('\t');
  Serial.print(wr2);Serial.print('\t');
  Serial.print(wr3);Serial.print('\t');
  Serial.print(wr4);Serial.print('\t');
}

void mostrar_esc(){
  Serial.print(esc1);Serial.print('\t');
  Serial.print(esc2);Serial.print('\t');
  Serial.print(esc3);Serial.print('\t');
  Serial.print(esc4);Serial.println('\t');
}
```

Figura A.2.2: Código PWM controlador PD 2

A.3 CÓGIDO RED DE AVANCE DOBLE

En esta parte ocurre lo mismo que en el caso anterior que el apartado PWM cambia y que en el programa principal se hace una inicialización de variables que son las siguientes:

```
//======== DISCRETIZACIÓN =========

float uk_1_1 = 0;
float uk_2_1 = 0;
float ek_1_1 = 0;
float ek_2_1 = 0;

float ek = 0;

float uk_1_2 = 0;
float uk_2_2 = 0;
float ek_1_2 = 0;
float ek_2_2 = 0;
```

Figura A.3.1: Código parte programa principal de la red de avance doble

```
void PWM(){
  tiempo_1 = micros();

  ek = -x[4];

  wr1 = 2500 + (((0.9207*uk_1_1) - (0.2124*uk_2_1) - (2057*ek) - (3430*ek_1_1) + (1373*ek_2_1)) / 25);
  wr3 = wr1;
  wr2 = 2500 + (((0.9207*uk_1_2) - (0.2124*uk_2_2) + (2057*ek) - (3430*ek_1_2) + (1373*ek_2_2)) / 25);
  wr4 = wr2;

  ////////////////////////////////////// VARIABLES PRIMERA FÓRMULA ////////////////////////////
  uk_2_1 = uk_1_1;
  uk_1_1 = wr1;
  ek_2_1 = ek_1_1;
  ek_1_1 = ek;
  ////////////////////////////////////////////////////////////////////////////////////////////

  ////////////////////////////////////// VARIABLES SEGUNDA FÓRMULA ////////////////////////////
  uk_2_2 = uk_1_2;
  uk_1_2 = wr2;
  ek_2_2 = ek_1_2;
  ek_1_2 = -ek;
  ////////////////////////////////////////////////////////////////////////////////////////////

  esc1 = (wr1-71.989)/25.162;
  if (esc1<40){
    esc1 = 40;
  }
  if (esc1>1000){
    esc1 = 1000;
  }
  esc2 = (wr2-71.989)/25.162;
  if (esc2<40){
    esc2 = 40;
  }
  if (esc2>1000){
    esc2 = 1000;
  }
```

```
  if (esc3>1000){
    esc3 = 1000;
  }
  esc4 = (wr4-71.989)/25.162;
  if (esc4<40){
    esc4 = 40 ;
  }
  if (esc4>1000){
    esc4 = 1000;
  }

  escl = map(escl,0,1023,0,180);
  esc2 = map(esc2,0,1023,0,180);
  esc3 = map(esc3,0,1023,0,180);
  esc4 = map(esc4,0,1023,0,180);

  ESC1.write(escl);
  ESC2.write(esc2);
  ESC3.write(esc3);
  ESC4.write(esc4);

  mostrar_wr();
  mostrar_esc();


  //ONled();
  //VBat();

  tiempo_ON = micros() - tiempo_1;
  if (tiempo_ON > 900) digitalWrite(13, HIGH);
}
void mostrar_wr(){
  Serial.print(wr1);Serial.print('\t');
  Serial.print(wr2);Serial.print('\t');
  Serial.print(wr3);Serial.print('\t');
  Serial.print(wr4);Serial.print('\t');
}

void mostrar_esc(){
  Serial.print(escl);Serial.print('\t');
  Serial.print(esc2);Serial.print('\t');
  Serial.print(esc3);Serial.print('\t');
  Serial.print(esc4);Serial.println('\t');
}
```

Figura A.3.2: Código PWM red de avance doble

BIBLIOGRAFÍA

- [1] https://www.arduino.cc/ (ONLINE)
- [2] https://processing.org/ (ONLINE)
- [3] TUTORIAL MPU6050, ACELERÓMETRO Y GIROSCOPIO (ONLINE) https://naylampmechatronics.com/blog/45_tutorial-mpu6050-acelerometro-y-giroscopio.html
- [4] HISTORIA DE LOS DRONES (ONLINE) https://es.digitaltrends.com/drones/la-historia-de-los-drones/
- [5] DRONES MÁS INNOVADORES (ONLINE) https://culture-lab.es/los-9-drones-mas-grandes-del-mundo-en-2020/
- [6] INTRODUCCIÓN A IDENTIFICACIÓN DE SISTEMAS. ENERO 2009 JAVIER SEDANO FRANCO Y JOSÉ RAMÓN VILLAR FLECHA
- [7] INGENIERÍA DE CONTROL. MODELADO, ANÁLISIS Y CONTROL DE SISTEMAS. OCTUBRE 2003. LUIS MORENO, SANTIAGO GARRIDO Y CALOR BALAGUER.
- [8] DRONES. MODELADO Y CONTROL DE CUADRICÓPTEROS. FEBRERO 2020. ROGER MIRANDA, ALEJANDRO GARRIDO, LUIS TUPAK AGUILAR Y JOSÉ ERNESTO HERRERO.