UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS FÍSICAS

DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

TRABAJO DE FIN DE GRADO

Código de TFG: ACA01

Creación de Simulaciones Físicas Interactivas
con Easy JavaScript Simulations

Building Interactive Physics Simulations
with Easy JavaScript Simulations

Supervisor/es: Jesús Chacón Sombría

Carlos Romera de Blas

Grado en Física

Curso académico 2021-22
Convocatoria de julio

Resumen: El modelado y control de vehículos aéreos no tripulados (UAV) es un problema desafiante debido a la no linealidad inherente. Estos exigen un control ideal para generar un movimiento preciso. Este trabajo tiene como objetivo el modelado y control de un cuadrirrotor. Para ello se describe la formulación de la dinámica del sistema y el espacio de estados para el diseño del controlador. Se discute una estrategia de control, basada en un controlador PD simple. Se diseña un simulador en *Easy JavaScript Simulations* para verificar fácilmente la estrategia de control y evaluar su eficacia. Finalmente, se discuten las posibles aplicaciones del sistema. Todos los resultados de la simulación se muestran en las partes del documento donde resulten necesarias. Todas las figuras están íntegramente elaboradas por el autor de esta memoria.

Abstract: Modelling and control of Unmanned Aerial Vehicles (UAV) is a challenging problem owing to the inherent non-linearity. They demand ideal control for generating precise motion. This work aims at modelling and controlling a quadrotor. Therefore, it describes the dynamics of the system and the state-space formulation for the controller design. A control strategy is discussed, based on a simple PD controller. A simulator is designed in *Easy JavaScript Simulations* to easily verify the control strategy and evaluate its effectiveness. In the end, possible applications of the system are discussed. All the simulation results are shown wherever they are needed. All the figures are entirely made by the author of this document.

Índice

1. Introducción

Día a día vemos como es cada vez más frecuente la automatización de todo lo que nos rodea. Tal y como sucedió con la revolución agrícola o la industrial, los empleos van desapareciendo, dando lugar a nuevas profesiones. En la antigüedad cualquier proceso necesitaba ser controlado por una persona, ya fuese con un carro de caballos o incluso con un coche. Sin embargo, continuamente observamos más vehículos que no necesitan un conductor, como es el caso de muchos trenes de metro que no tienen ningún conductor *in situ* o de coches como los Tesla que no requieren de un humano para desplazarse. En el sector de la aviación tenemos el famoso caso del autopiloto de los aviones que permiten a los pilotos reposar durante buen parte del trayecto, siendo su trabajo exclusivamente el aterrizaje, el despegue y las posibles zonas con turbulencias.

Un vehículo aéreo no tripulado se conoce como un UAV, por sus siglas en inglés: Unmanned Aerial Vehicle. Este tipo de vehículos se basan en el cálculo de su propia trayectoria para llegar a su destino, sin la necesidad de un piloto que gire a la izquierda o a la derecha para lograrlo. En la actualidad podemos observar como algunos de estos UAV son lo suficientemente pequeños y ligeros como para moverse entre obstáculos, como por ejemplo los edificios de una ciudad. En en un futuro este hecho puede brindar oportunidades de envío de paquetes y mensajería sin precisar de personas que se trasladen hasta cada domicilio para introducir una carta en el buzón. Estamos hablando de los populares drones y, en nuestro caso, de un cuadrirrotor, es decir, un dron con cuatro motores/rotores.

En este documento vamos a tratar de modelar, simular y controlar un cuadrirrotor, siendo este capaz de llegar a las coordenadas establecidas sin necesidad de interferencia humana. Asimismo, será capaz de perseguir un objetivo e interceptarlo en caso de ser necesario. El único motivo por el que estos UAVs no están sumamente extendidos a día de hoy es la corta duración de las baterías, que no pueden ser de mayor tamaño por su elevado peso, lo que va en contra de la filosofía de un vehículo ultraligero.
Objetivo: Simulación de un cuadrirrotor alcanzando una posición XYZ eficientemente.

- Modelado del cuadrirrotor.
- Control del cuadrirrotor.
- Implementación en EJSS.
- Programación del algoritmo de Pure Pursuit.

2. Modelado

Un cuadrirrotor como bien indica su nombre cuenta con cuatro rotores, los cuales van a permitir que el dron se desplace en el espacio. Para entender bien el modelo con el que vamos a trabajar, veremos primero un esquema del dispositivo con el centro del cuadrirrotor $\{V\}$ como sistema de referencia (Figura 1). La velocidad angular de los rotores es respectivamente $\omega_1, \omega_2, \omega_3$ y ω_4 con giro horario *Front* y *Back*, y antihorario *Left* y *Right* tal y como se indica en la figura. Cada uno de los cuatro rotores genera un empuje en el sentido negativo de z, es decir,

$$T_i = -b \cdot \omega_i^2$$

donde b es una constante de elevación que depende la densidad del aire, el radio de la pala del rotor al cubo, el número de palas y la longitud de cuerda de la pala.

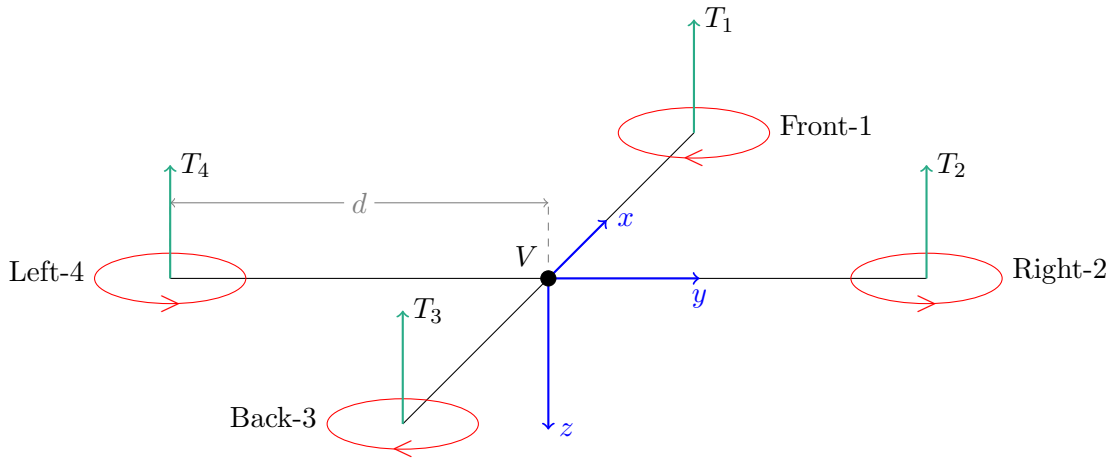


Figura 1: Esquema del Cuadirrotor

El empuje total del vehículo es

$$T = \sum_{i=1}^{4} T_i = -b \sum_{i=1}^{4} \omega_i^2 \tag{2.1}$$

Los cuatro rotores funcionan por motores eléctricos alimentados por voltajes dados por señales de control.

La dinámica de traslación del cuadrirrotor en el sistema de referencia del suelo $\{0\}$ con la misma orientación de ejes, viene dado por la segunda ley de Newton:

$$\mathbf{F} = m \cdot \ddot{\mathbf{x}} = \mathbf{F}_{\text{grav}} + \mathbf{F}_{\text{UAV}} = m \cdot \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} + \mathcal{R}_V^0 \cdot \begin{pmatrix} 0 \\ 0 \\ T \end{pmatrix} \tag{2.2}$$

siendo $\mathcal{R}_V^0$ la matriz de cambio de coordenadas de $\{V\}$ a $\{0\}$, m la masa del vehículo y g la gravedad (positiva por los ejes). La matriz de cambio de coordenadas viene dado precisamente por la inclinación del vehículo en los tres ejes, que viene dada por:

- *Roll* o Alabeo. Giro horario respecto al eje x; lo representaremos como θ_r.
- *Pitch* o Cabeceo. Giro antihorario respecto al eje y; lo representaremos como θ_p.
- *Yaw* o Guiñada. Giro horario respecto al eje z; lo representaremos como θ_y.

Por ello, la matriz de cambio de coordenadas viene dada por

$$\begin{aligned} \mathcal{R}_V^0 &= \mathcal{R}_z(\theta_y) \cdot \mathcal{R}_y(\theta_p) \cdot \mathcal{R}_x(\theta_r) \\ &= \begin{pmatrix} \cos(\theta_y) & -\sin(\theta_y) & 0 \\ \sin(\theta_y) & \cos(\theta_y) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta_p) & 0 & \sin(\theta_p) \\ 0 & 1 & 0 \\ -\sin(\theta_p) & 0 & \cos(\theta_p) \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_r) & -\sin(\theta_r) \\ 0 & \sin(\theta_r) & \cos(\theta_r) \end{pmatrix} \end{aligned}$$

siendo la matriz de cambio de coordenadas de $\{V\}$ a $\{0\}$:

$$\mathcal{R}_V^0 = \begin{pmatrix} \cos(\theta_y) \cdot \cos(\theta_p) & -\sin(\theta_y) \cdot \cos(\theta_r) + \cos(\theta_y) \cdot \sin(\theta_p) \cdot \sin(\theta_r) & \sin(\theta_y) \cdot \sin(\theta_r) + \cos(\theta_y) \cdot \sin(\theta_p) \cdot \cos(\theta_r) \\ \sin(\theta_y) \cdot \cos(\theta_p) & \cos(\theta_y) \cdot \cos(\theta_r) + \sin(\theta_y) \cdot \sin(\theta_p) \cdot \sin(\theta_r) & -\cos(\theta_y) \cdot \sin(\theta_r) + \sin(\theta_y) \cdot \sin(\theta_p) \cdot \cos(\theta_r) \\ -\sin(\theta_p) & \cos(\theta_p) \cdot \sin(\theta_r) & \cos(\theta_p) \cdot \cos(\theta_r) \end{pmatrix} \tag{2.3}$$

Su inversa, que es igual a la traspuesta, es la matriz de cambio de coordenadas de $\{0\}$ a $\{V\}$.

Utilizando la expresión de la matriz rotacional y despejando las aceleraciones en los tres ejes, podemos reescribir la ecuación (2.2) de la siguiente manera:

$$\begin{cases} \ddot{x} = \dfrac{T}{m} \underbrace{(\sin(\theta_{\mathrm{y}}) \cdot \sin(\theta_{\mathrm{r}}) + \cos(\theta_{\mathrm{y}}) \cdot \sin(\theta_{\mathrm{p}}) \cdot \cos(\theta_{\mathrm{r}}))}_{u_x} \\ \ddot{y} = \dfrac{T}{m} \underbrace{(-\cos(\theta_{\mathrm{y}}) \cdot \sin(\theta_{\mathrm{r}}) + \sin(\theta_{\mathrm{y}}) \cdot \sin(\theta_{\mathrm{p}}) \cdot \cos(\theta_{\mathrm{r}}))}_{u_y} \\ \ddot{z} = g + \dfrac{T}{m} \cdot \underbrace{\cos(\theta_{\mathrm{p}}) \cdot \cos(\theta_{\mathrm{r}})}_{u_z} \end{cases} \tag{2.4}$$

Podemos observar que el sistema de las aceleraciones espaciales tiene sentido físico.

1. Para $\ddot{x}$, observamos que si $\theta_p = 0$ y $\theta_y = 0$, entonces $\ddot{x} = 0$, es decir, el ángulo *roll* (θ_r) no afecta en solitario a la aceleración en el eje x. En cambio, si el *roll* y el *yaw* son nulos, la acelaración depende del seno del *pitch*, lo cual concuerda con la idea de que al bajar la cabeza (*Front*) entonces se acelera positivamente en el eje x.

2. Para $\ddot{y}$, observamos que si $\theta_r = 0$ y $\theta_y = 0$ entonces $\ddot{x} = 0$, es decir, el ángulo *pitch* (θ_p) no afecta en solitario a la aceleración en el eje y. En cambio si el *pitch* y el *yaw* son nulos, la acelaración depende del menos seno del *roll*, lo cual concuerda con la idea de que al bajar *Right* (θ_r negativo al ser contrasentido) entonces se acelera positivamente en el eje y.

3. Para $\ddot{z}$, observamos que es completamente independiente del ángulo θ_y. En particular, observamos que si el *pitch* y *roll* son nulos entonces la aceleración depende de la gravedad, el empuje y la masa de forma que
$$\ddot{z} = g + \frac{T}{m}$$
Si se quiere hacer constante la altura de vuelo es necesario que la aceleración en z sea nula, por lo que la velocidad angular de los rotores (siendo en todos la misma al mantenerse fijo en el plano XY) cumple que
$$T_0 = -m \cdot g \Longleftrightarrow -b \cdot \sum_{i=1}^{4} \omega_i^2 = -m \cdot g \Longleftrightarrow \omega_0 = \sqrt{\frac{mg}{4b}}$$
Igualmente para mantener la aceleración nula en z, sin ser *pitch* y *roll* nulos, es necesario que el empuje sea:
$$T_0 = -\frac{m \cdot g}{\cos(\theta_{\mathrm{p}}) \cdot \cos(\theta_{\mathrm{r}})} \tag{2.5}$$

Vamos a formalizar las observaciones que hemos hecho estudiando los torques en los tres ejes, creados por la diferencia en las velocidades angulares de los rotores, o lo que es lo mismo, la diferencia en los empujes.

- El torque alrededor del eje x es la diferencia entre los empujes de *Right* y *Left* por la distancia d del centro a los rotores
$$\tau_x = d \cdot (T_4 - T_2) = -d \cdot b \cdot (\omega_4^2 - \omega_2^2)$$

- El torque alrededor del eje y depende de la diferencia entre los empujes *Front* y *Back*

$$\tau_y = d \cdot (T_1 - T_3) = -d \cdot b \cdot (\omega_1^2 - \omega_3^2)$$

- El torque aplicado a cada hélice por el motor se ve opuesto por una resistencia aerodinámica $Q_i = k \cdot \omega_i^2$, donde k depende de los mismos factores que b. Por lo tanto, el torque alrededor del eje z viene dado por

$$\tau_z = Q_1 - Q_2 + Q_3 - Q_4 = k \cdot (\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2)$$

  donde los signos se deben por el sentido de giro de los rotores, indicados en el esquema inicial.

Utilizando las ecuaciones de los torques y la ecuación del empuje total (2.1) se pueden escribir las velocidades angulares de los rotores en función del empuje y los torques ($\Gamma = (\tau_x, \tau_y, \tau_z)$) como:

$$\begin{pmatrix} T \\ \Gamma \end{pmatrix} = \underbrace{\begin{pmatrix} -b & -b & -b & -b \\ 0 & -d \cdot b & 0 & d \cdot b \\ d \cdot b & 0 & -d \cdot b & 0 \\ k & -k & k & -k \end{pmatrix}}_{B^{-1}} \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} \Longrightarrow \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} = B \cdot \begin{pmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix}$$

Por tanto, se tiene que las velocidades angulares de cada rotor, teniendo en cuenta su sentido de giro son precisamente

$$\begin{cases} \omega_1 = \sqrt{-\dfrac{T}{4b} + \dfrac{\tau_y}{2bd} + \dfrac{\tau_z}{4k}} \\ \omega_2 = -\sqrt{-\dfrac{T}{4b} - \dfrac{\tau_x}{2bd} - \dfrac{\tau_z}{4k}} \\ \omega_3 = \sqrt{-\dfrac{T}{4b} - \dfrac{\tau_y}{2bd} + \dfrac{\tau_z}{4k}} \\ \omega_4 = -\sqrt{-\dfrac{T}{4b} + \dfrac{\tau_x}{2bd} - \dfrac{\tau_z}{4k}} \end{cases} \tag{2.6}$$

La ecuación rotacional viene dada por la ecuación de movimiento de Euler

$$\mathcal{I} \cdot \dot{\mathbf{w}} = -\mathbf{w} \times (\mathcal{I} \cdot \mathbf{w}) + \Gamma \tag{2.7}$$

Siendo $\mathcal{I} \equiv$ matriz de inercia del UAV, $\mathbf{w} \equiv (\dot{\theta_r}, \dot{\theta_p}, \dot{\theta_y})$, vector velocidad angular de $\{V\}$ frente a $\{0\}$, y $\Gamma \equiv$ torque aplicado al sistema $\{V\}$. Nosotros vamos a trabajar con un UAV con una matriz de inercia diagonal, es decir, $\mathcal{I} = diag(I_r, I_p, I_y)$, por lo podemos reescribir la ecuación de movimiento de Euler (2.7), despejando las aceleraciones angulares como

$$\begin{cases} \ddot{\theta_r} = \underbrace{\dfrac{I_p - I_y}{I_r}}_{a_1} \cdot \dot{\theta_p} \cdot \dot{\theta_y} + \dfrac{\tau_x}{I_r} \\ \ddot{\theta_p} = \underbrace{\dfrac{I_y - I_r}{I_p}}_{a_2} \cdot \dot{\theta_y} \cdot \dot{\theta_r} + \dfrac{\tau_y}{I_p} \\ \ddot{\theta_y} = \underbrace{\dfrac{I_r - I_p}{I_y}}_{a_3} \cdot \dot{\theta_r} \cdot \dot{\theta_p} + \dfrac{\tau_z}{I_y} \end{cases} \tag{2.8}$$

Si juntamos las ecuaciones que rigen la rotación (2.8) y traslación (2.4) del UAV. Podemos escribir un sistema de ecuaciones diferenciales de primer orden con 12 variables de estado. Dichas variables de estado son

$$\begin{cases} x_1 = \theta_r & x_4 = \dot{\theta_r} \\ x_2 = \theta_p & x_5 = \dot{\theta_p} \\ x_3 = \theta_y & x_6 = \dot{\theta_y} \end{cases} \qquad \begin{cases} x_7 = x & x_{10} = \dot{x} \\ x_8 = y & x_{11} = \dot{y} \\ x_9 = z & x_{12} = \dot{z} \end{cases}$$

Las cuales conforman el sistema de EDOs de primer Orden

$$\begin{cases} \dot{x_1} = x_4 \\ \dot{x_2} = x_5 \\ \dot{x_3} = x_6 \\ \dot{x_4} = a_1 \cdot x_5 \cdot x_6 + \dfrac{\tau_x}{I_r} \\ \dot{x_5} = a_2 \cdot x_6 \cdot x_4 + \dfrac{\tau_y}{I_p} \\ \dot{x_6} = a_3 \cdot x_4 \cdot x_5 + \dfrac{\tau_z}{I_y} \\ \dot{x_7} = x_{10} \\ \dot{x_8} = x_{11} \\ \dot{x_9} = x_{12} \\ \dot{x_{10}} = \dfrac{T}{m} \cdot u_x \\ \dot{x_{11}} = \dfrac{T}{m} \cdot u_y \\ \dot{x_{12}} = g + \dfrac{T}{m} \cdot u_z \end{cases} \tag{2.9}$$

Teniendo el sistema de las ecuaciones que rigen el modelo del cuadrirrotor, veremos en la siguiente sección como podemos controlar dicho modelo.

3. Control

Una vez obtenido el modelo físico que tenemos que manejar, debemos observar cómo podemos interaccionar con el sistema. En el esquema inicial del modelo observábamos que lo que realmente podíamos controlar son las velocidades angulares de los cuatro rotores del UAV. Estas velocidades angulares las podíamos expresar a su vez en función del torque y empuje como veíamos en la ecuación (2.6). Por lo tanto, nuestras señales de control van a ser precisamente el empuje y torques que queremos, pues resulta mucho más sencillo interaccionar con el sistema (2.9) de esta manera.

Vamos a tener, por tanto, un vector de cuatro señales de control

$$u = (u_0, u_1, u_2, u_3) = (u_T, u_r, u_p, u_y) = (T, \tau_x, \tau_y, \tau_z)$$

Estas señales de control van a basarse en un controlador PD (Proporcional, Derivativo). Por ejemplo, para una variable a, siendo a^* el punto establecido o que desea alcanzar, el control PD viene dado por el siquiente esquema.

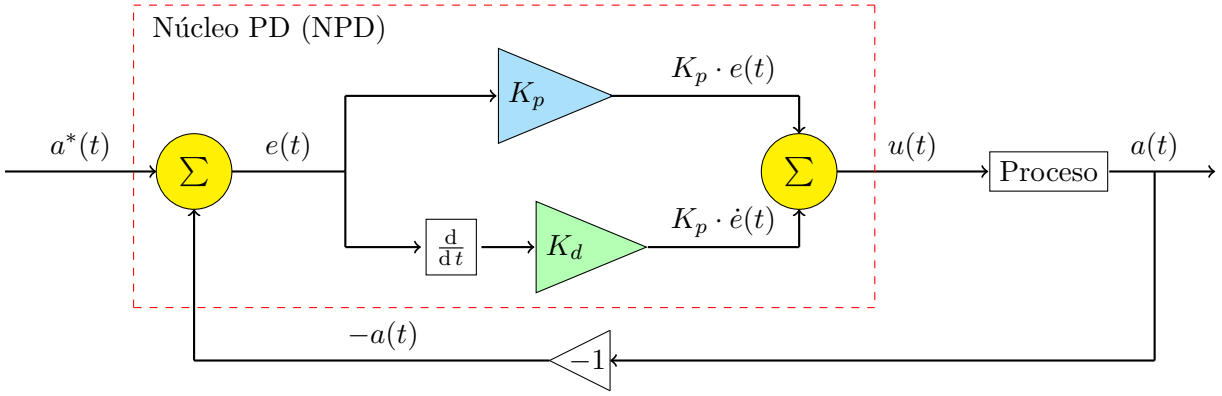


Figura 2: Control PD

En el esquema observamos que el núcleo del controlador tiene dos entradas y una salida.

Si observamos el sistema (2.9), las señales u_1, u_2, u_3 nos van a permitir controlar los ángulos *roll*, *pitch* y *yaw*. Por otro lado, la señal u_0, que es el empuje nos va a permitir controlar la altura del sistema. Asimismo, tenemos que ver como controlar la posición XY mediante las cuatro señales de control disponibles.

3.1. Attitude Control

Para el control de la inclinación del UAV respecto al plano terrestre vamos a utilizar un controlador PD (Figura 2) para cada ángulo $i = r, p, y$ de la forma

$$u_i(t) = K_p^i \cdot e_{\theta_i}(t) + K_d^i \cdot \dot{e}_{\theta_i}(t)$$

siendo $e_i(t) = \theta_i^* - \theta_i$, es decir, el error entre el punto establecido θ_i^* y el punto actual θ_i. La derivada del error es $\dot{e}_i(t) = \dot{\theta}_i^* - \dot{\theta}_i$, donde la velocidad del *setpoint* es despreciable, por lo que se va a omitir. Las constantes de proporcionalidad K_p y derivativa K_d se ajustan manualmente mediante la simulación. Por lo tanto, las señales de *attitude* control son

$$\begin{cases} u_1 = K_p^r \cdot (\theta_r^* - \theta_r) - K_d^r \cdot \dot{\theta}_r \\ u_2 = K_p^p \cdot (\theta_p^* - \theta_p) - K_d^p \cdot \dot{\theta}_p \\ u_3 = K_p^y \cdot (\theta_y^* - \theta_y) - K_d^y \cdot \dot{\theta}_y \end{cases} \tag{3.1}$$

3.2. Altitude-Thrust Control

Para el control de la altura del AUV se va a utilizar también un control PD pero añadiendo un término de estabilidad, es decir, un término que permita mantener la altura en todo momento aunque el punto establecido de altura no cambie. De esta forma vamos a tener

$$u_0 = K_p^T \cdot (z^* - z) - K_d^T \cdot \dot{z} + T_0 \tag{3.2}$$

donde T_0 viene dado por la ecuación (2.5).

3.3. Position Control

Para el control de la posición XY vamos a utilizar respectivos controles PD de los ángulos *roll* y *pitch* que se quieren establecer, utilizando para ello la posición XY en el sistema $\{V\}$. En nuestro caso el *yaw* no se va a utilizar.

De esta forma el *roll* y el *pitch* que vamos a establecer en función de los puntos x^* e y^* son:

$$\begin{cases} \theta_p^* = K_p^X(x_V^* - x_V) - K_d^X \cdot \dot{x}_V \\ \theta_r^* = K_p^Y(y_V^* - y_V) - K_d^Y \cdot \dot{y}_V \end{cases} \tag{3.3}$$

Donde $x_V, y_V, ...$ son las variables en el sistema de referencia V, para lo que basta multiplicar por la matriz de rotación $\mathcal{R}_0^V = (\mathcal{R}_V^0)^T$, es decir, $\mathbf{x}_V = \mathcal{R}_0^V\,\mathbf{x}$ e igual para $\mathbf{x}^*$ y $\dot{\mathbf{x}}$. De esta forma se obtiene

$$\begin{cases} x_V = (x,y,z)\cdot \begin{pmatrix} \cos(\theta_{\mathrm{y}})\cdot\cos(\theta_{\mathrm{p}}) \\ \sin(\theta_{\mathrm{y}})\cdot\cos(\theta_{\mathrm{p}}) \\ -\sin(\theta_{\mathrm{p}}) \end{pmatrix} \\ y_V = (x,y,z)\cdot \begin{pmatrix} -\sin(\theta_{\mathrm{y}})\cdot\cos(\theta_{\mathrm{r}}) + \cos(\theta_{\mathrm{y}})\cdot\sin(\theta_{\mathrm{p}})\cdot\sin(\theta_{\mathrm{r}}) \\ \cos(\theta_{\mathrm{y}})\cdot\cos(\theta_{\mathrm{r}}) + \sin(\theta_{\mathrm{y}})\cdot\sin(\theta_{\mathrm{p}})\cdot\sin(\theta_{\mathrm{r}}) \\ \cos(\theta_{\mathrm{p}})\cdot\sin(\theta_{\mathrm{r}}) \end{pmatrix} \end{cases} \tag{3.4}$$

En la simulación veremos como este control funciona a la perfección.

3.4. Control del sistema

Los controladores del *roll*, *pitch* y *yaw* son los más simples y se basan en la Figura 2. El control de altura también, pero además se le añade un término a la señal $u(t)$ antes del proceso. Sin embargo, los controles de X e Y tienen un proceso más complejo, como podemos ver en la ecuación (3.4), pues requieren de más calculos al hacer un cambio de coordenadas (CC). Además el proceso incluye el control de los ángulos. Los cambios de coordenadas requieren de 6 entradas y tienen 3 salidas. Los procesos vienen dados por las ecuaciones del sistema (2.9), pero para no sobrecargar el esquema se omiten.

De esta manera, nuestro esquema completo de control usando controladores según la figura 2 y entendiendo NPD como el núcleo del controlador PD viene dado por la figura 3.

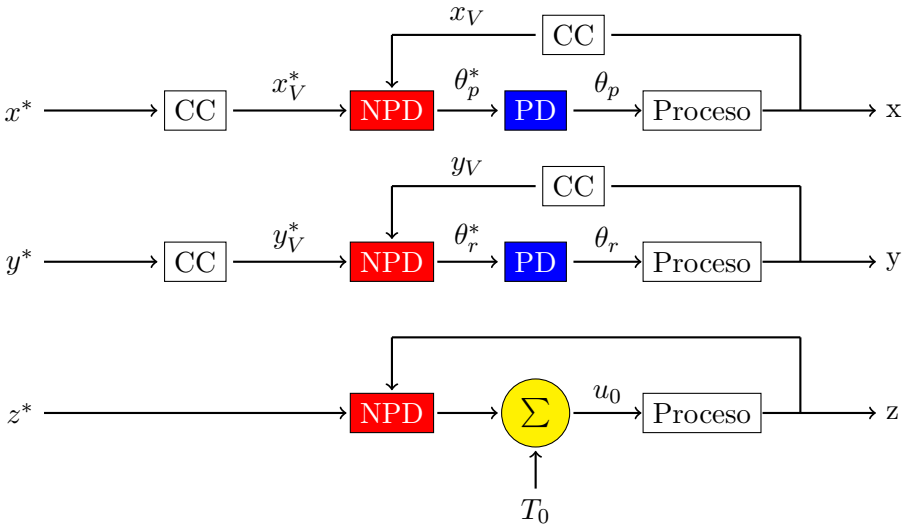


Figura 3: Control del sistema

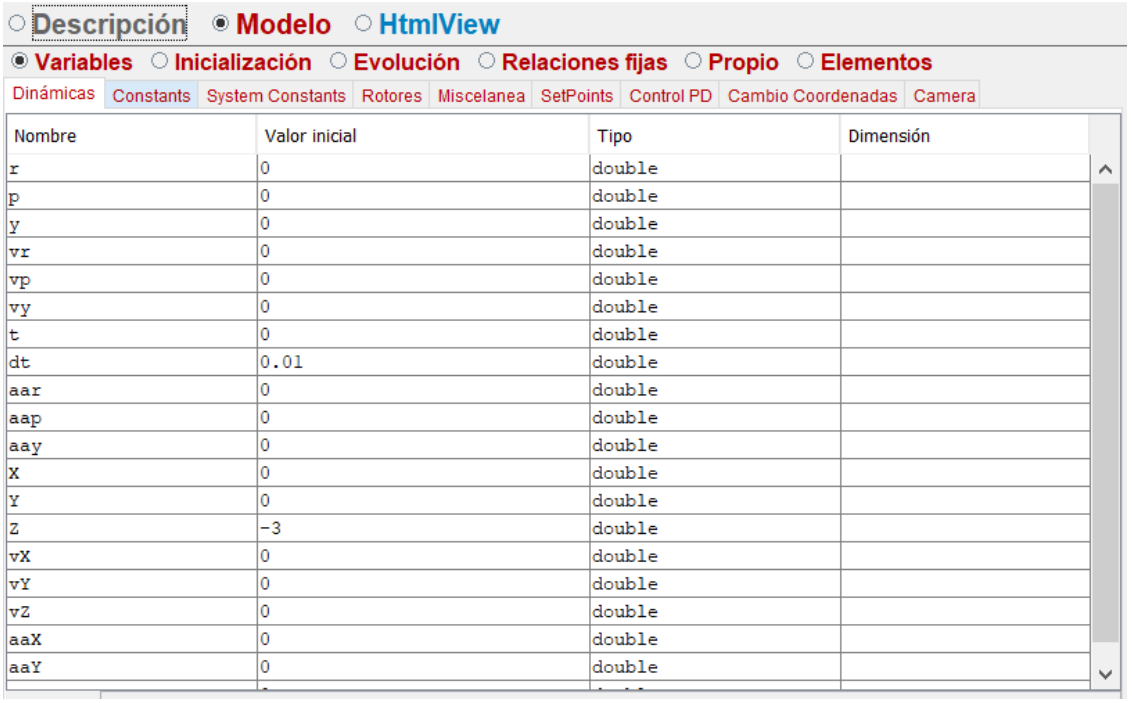
4. Simulación

Para la simulación del modelo y control del cuadrirrotor se utiliza *Easy JavaScript Simulations* . Los parámetros que vamos a utilizar se han extraído de un cuadrirrotor real.

Parámetro	Valor	Descripción
I_r	$0.01335\ kg \cdot m^2$	Inercia del *roll*
I_p	$0.01335\ kg \cdot m^2$	Inercia del *pitch*
I_y	$0.02465\ kg \cdot m^2$	Inercia del *yaw*
m	$2\ kg$	Masa del cuadrirrotor
d	$0.18\ m$	Longitud de brazo
g	$9.81\ m \cdot s^{-2}$	Aceleración gravitatoria
b	$8{,}54 \cdot 10^{-6}\ kg \cdot m$	Coeficiente aerodinámico
k	$1{,}36 \cdot 10^{-7}\ kg \cdot m^2$	Coeficiente de arrastre

Cuadro 1: Parámetros del cuadrirrotor usados en la simulación.

Para la implementación se definen todas las variables y constantes que se van a utilizar en el apartado Modelo : Variables. Para ello se han utilizado 9 páginas diferentes donde se distingue entre las variables de estado, constantes del cuadrirrotor, constantes del sistema (2.9), variables de los rotores, puntos establecidos, variables del control PD, variables de cambio de coordenadas, misceláneas y control de cámara.



Nombre	Valor inicial	Tipo	Dimensión
r	0	double	
p	0	double	
y	0	double	
vr	0	double	
vp	0	double	
vy	0	double	
t	0	double	
dt	0.01	double	
aar	0	double	
aap	0	double	
aay	0	double	
X	0	double	
Y	0	double	
Z	-3	double	
vX	0	double	
vY	0	double	
vZ	0	double	
aaX	0	double	
aaY	0	double	

Figura 4: Algunas variables de la simulación

Para la resolución del sistema (2.9) se incluyen las EDOs en el apartado Modelo: Evolución.

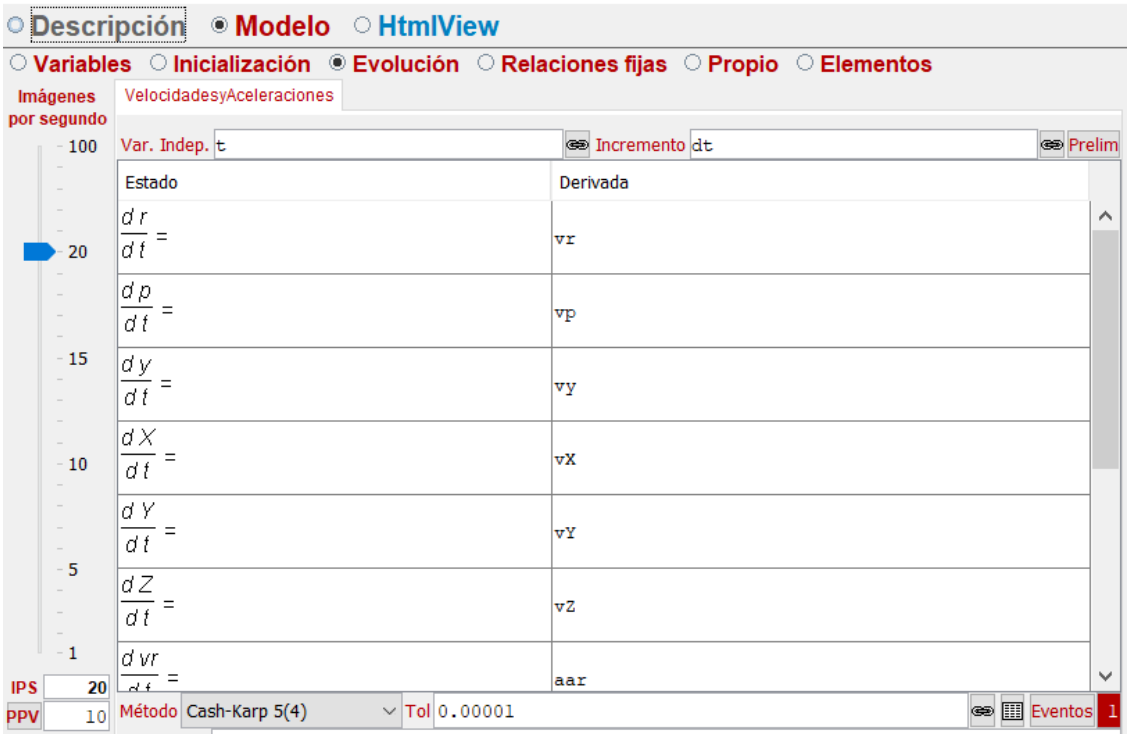


Figura 5: Cálculo de la evolución de la simulación

Previo a dicho cálculo se incluye el control de *attitude* (ec. 3.1), altura (ec. 3.2) y posición (ec. 3.3) en preliminares, al igual que se calcula la velocidad angular de los rotores (ec. 2.6).

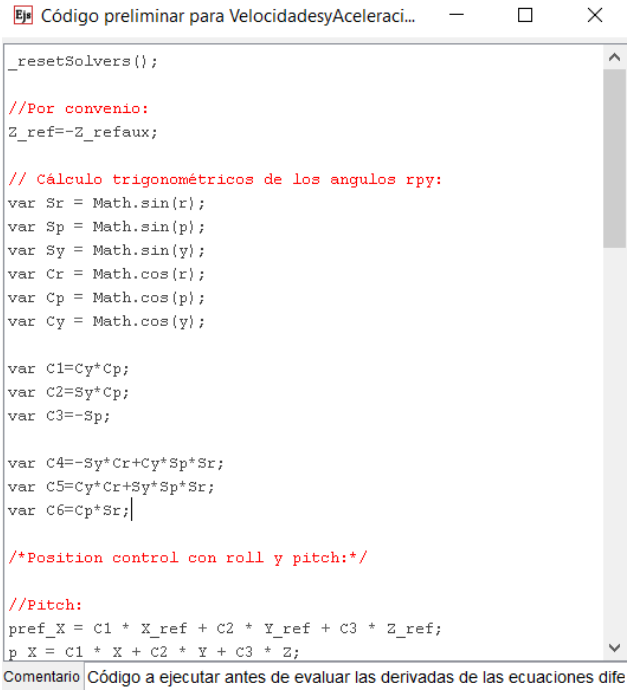


Figura 6: Cálculos preliminares de la simulación

En eventos, se establece la colisión del dron contra el suelo al alcanzar la altura $z = 0$.

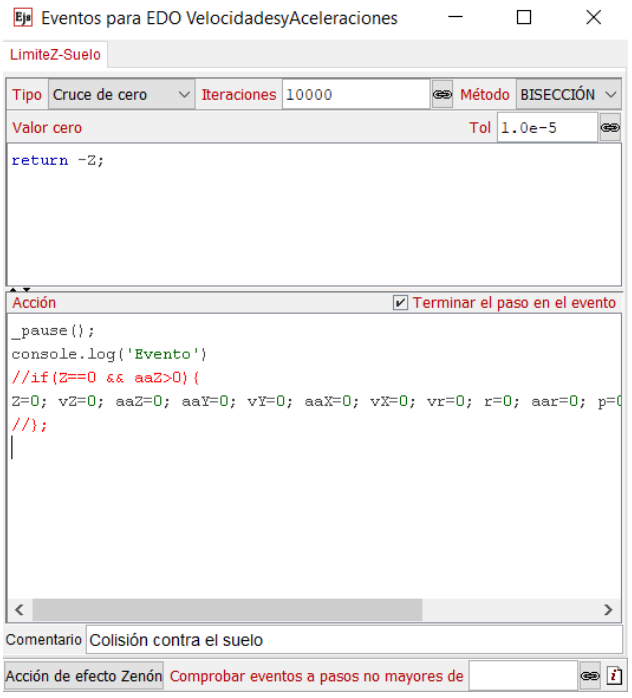


Figura 7: Eventos de la simulación

Por último, la interfaz se diseña en el apartado HtmlView:

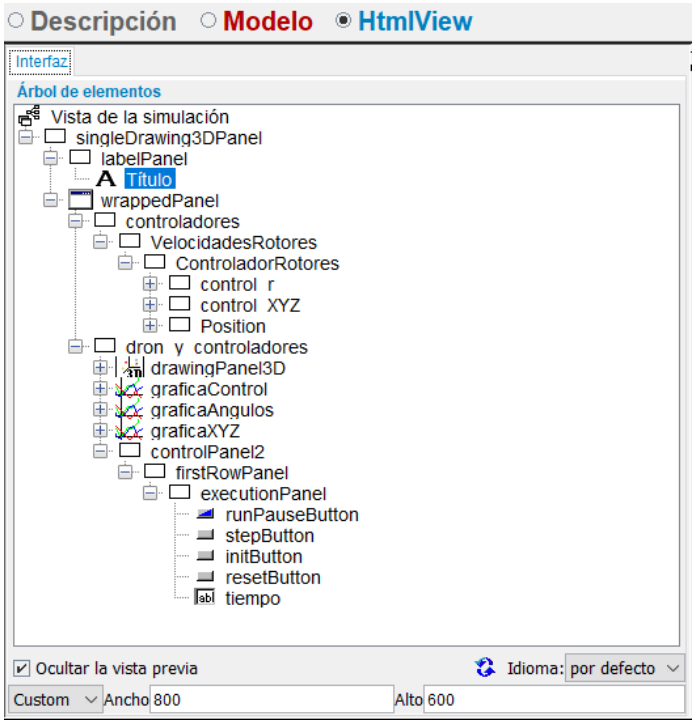


Figura 8: Elementos de la interfaz simulación

Al ejecutar la simulación nos encontramos con la siguiente interfaz, donde se pueden establecer las coordenadas espaciales y, de manera automática, el UAV llegará hasta ellas:

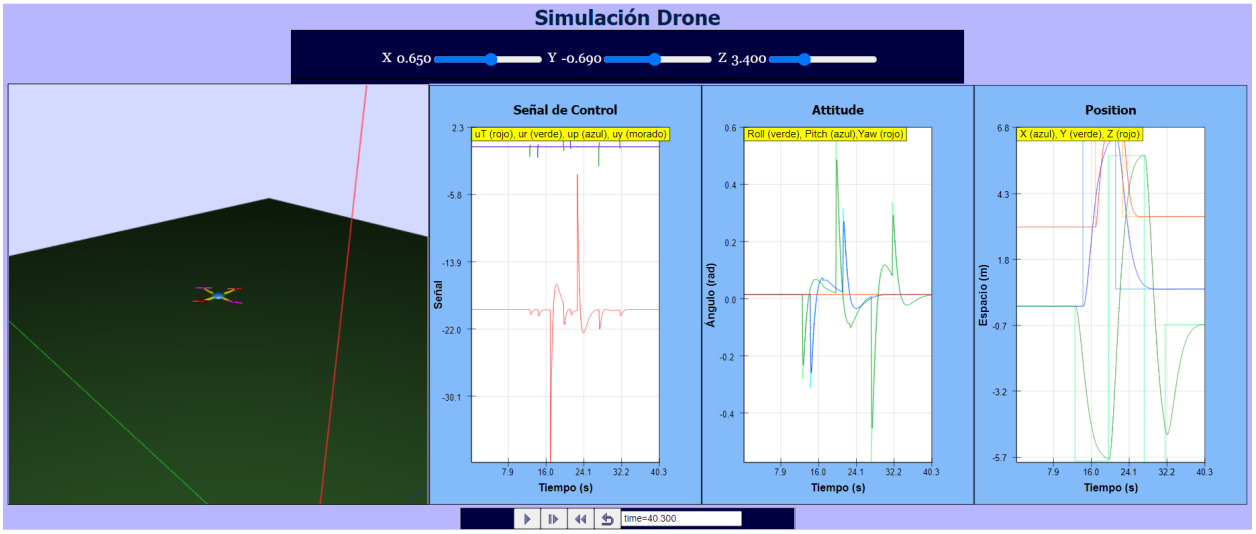


Figura 9: Interfaz de la simulación

En esta figura podemos observar tres gráficas: señales de control, *attitude* y posición. En las dos últimas podemos observar los puntos establecidos frente a los del sistema. En la parte superior podemos establecer coordenadas escpaciales y en la parte inferior tenemos un panel de control.

Tras implementar el *pure pursuit* la interfaz añade un botón para habitarlo y establecer las coordenadas sin límites.

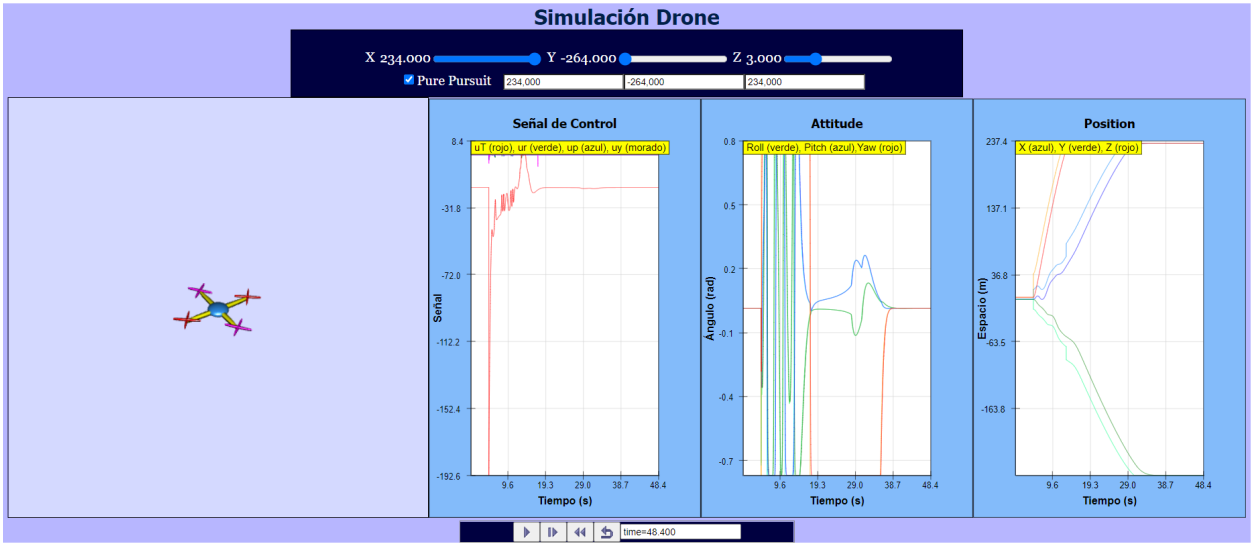


Figura 10: Interfaz de la simulación

5. Resultados

Aunque en la interfaz queda oculto, se pueden controlar los ángulos *roll*, *pitch* y *yaw* de manera directa. De esta forma, es posible establecer los parámetros K_p y K_d de cada uno de los ángu-

los, obteniendo un control muy bueno. Para cada uno los ángulos se obtienen gráficas libres de oscilaciones, además de necesitar poco tiempo para alcanzar el ángulo deseado.

- *Roll*:

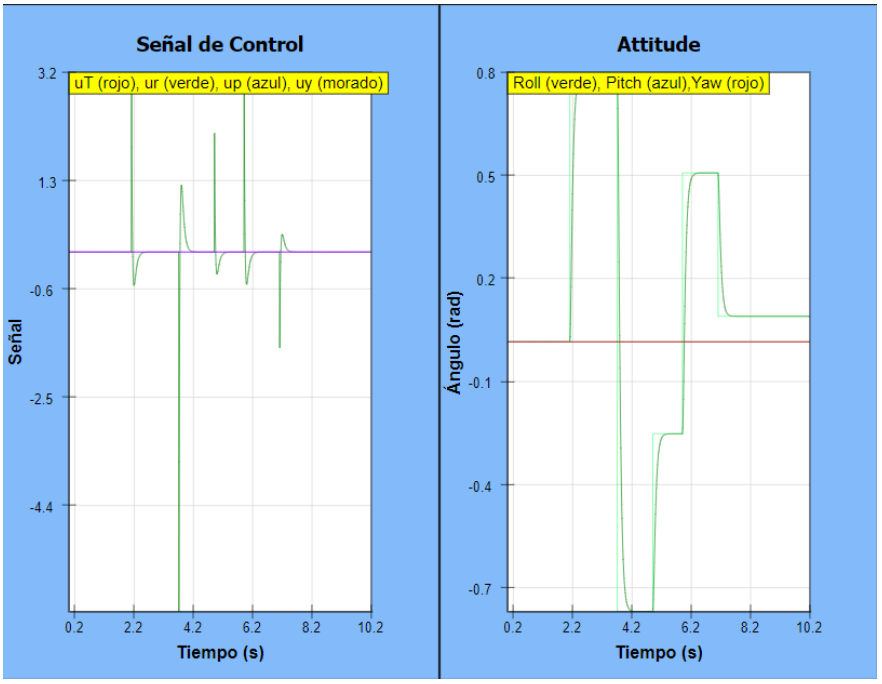


Figura 11: Control del *Roll*

- *Pitch*:

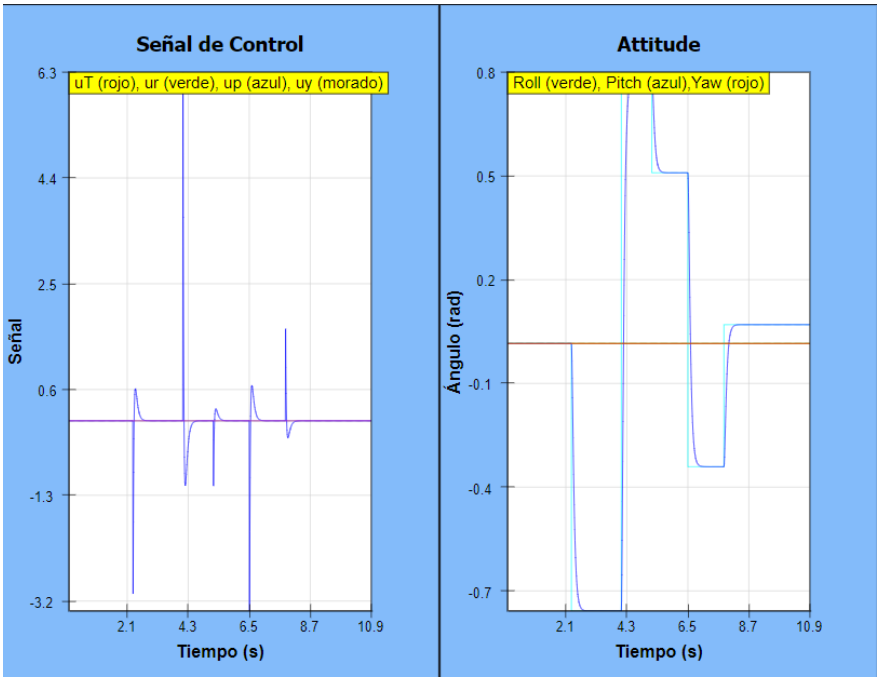


Figura 12: Control del *Pitch*

- *Yaw*:

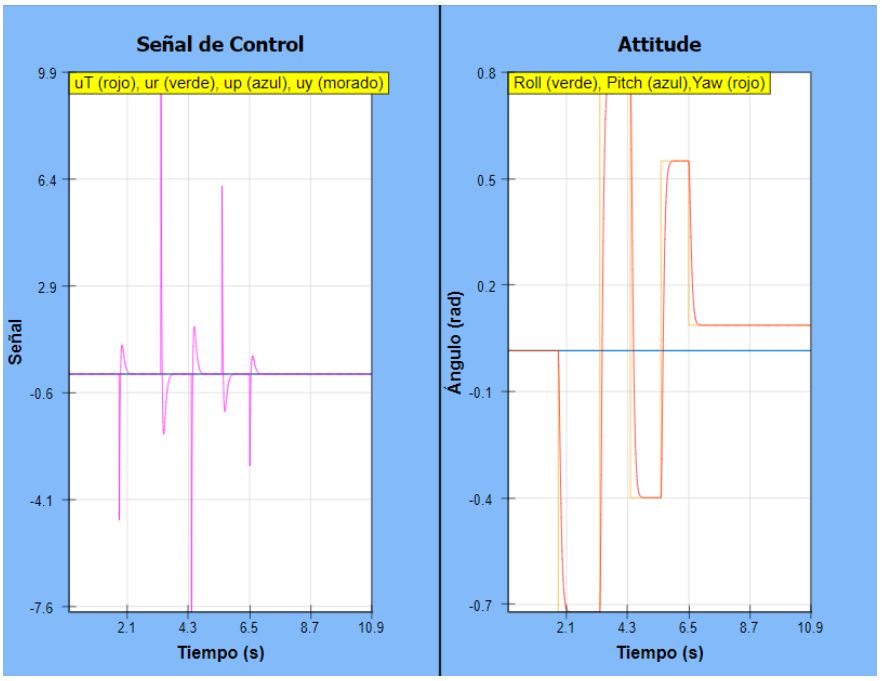


Figura 13: Control del *Yaw*

Para hacer el análisis de la eficiencia de control es necesario anular las ecuaciones de posición y el control de posición. De esta forma, podemos observar para cada ángulo las señales de control y qué el control es preciso.

Para obtener un control tan preciso se han establecido los parámetros del controlador PD como se muestran en la siguiente tabla:

Ángulo	K_p	K_d
Roll	6	0.6
Pitch	6	0.6
Yaw	9	0.9

Cuadro 2: Constantes del Attitude Control PD

Para el control de la altura se habilita el control de z^* en la interfaz.

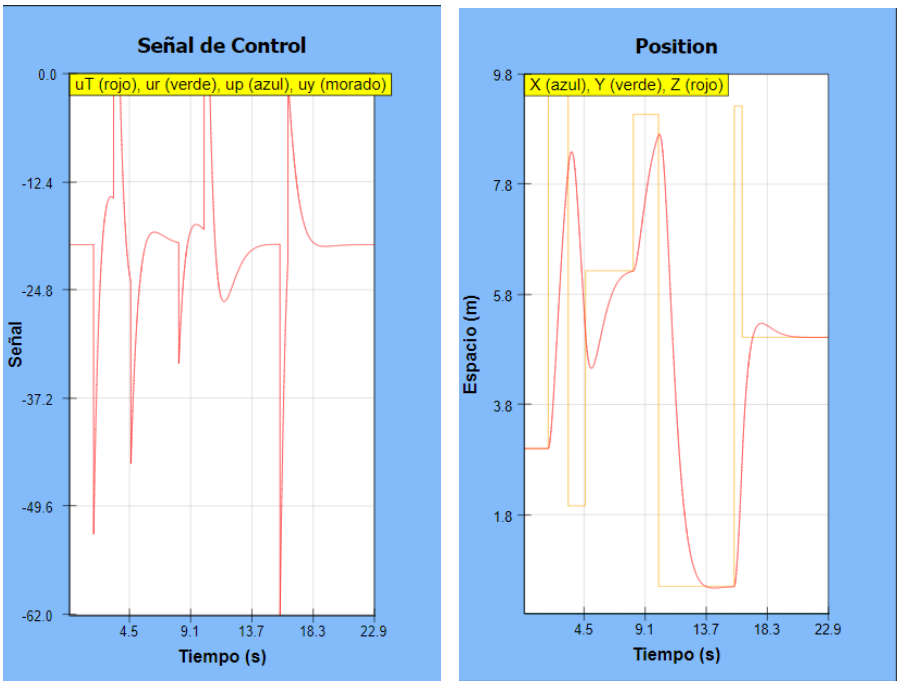


Figura 14: Señal de Control y Control de la altura

Se puede observar que se obtienen señales de control mucho más grandes, sin embargo $u_T \leq 0$ siempre, pues es una limitación física del sistema al no poderse generar un empuje hacia el suelo. El control es eficiente y sin oscilaciones. En este caso, se puede observar que es más lento, ya que se están recorriendo distancias considerables. Para obtener este control se han establecido los parámetros del controlador PD de la altura como sigue:

Empuje	K_p	K_d
T	5	5.5

Cuadro 3: Constantes del Altitude Control PD

El control de posición se realiza una vez que los controles de postura y altura son eficientes. Se establece de nuevo la evolución del sistema sin cancelar las ecuaciones de posición y activando el control de posición. Se obtiene un control eficiente, sin permitir el establecimiento de ángulos desorbitados para los propósitos. La consecuencia de esto es que se necesita más tiempo para alcanzar la posición.

En las figuras 15, 16 y 17 la posición se encuentra limitada de -10 a 10 para los ejes X,Y y entre 0.5 y 10 para el eje Z. De esta manera, el máximo cambio que puede haber es 20 y 9.5 respectivamente, evitando señales demasiado grandes que den lugar a oscilaciones. En el proceso de alcanzar la posición deseada se puede ver de nuevo la gran eficacia en el *attitude* control.

- Posición X:

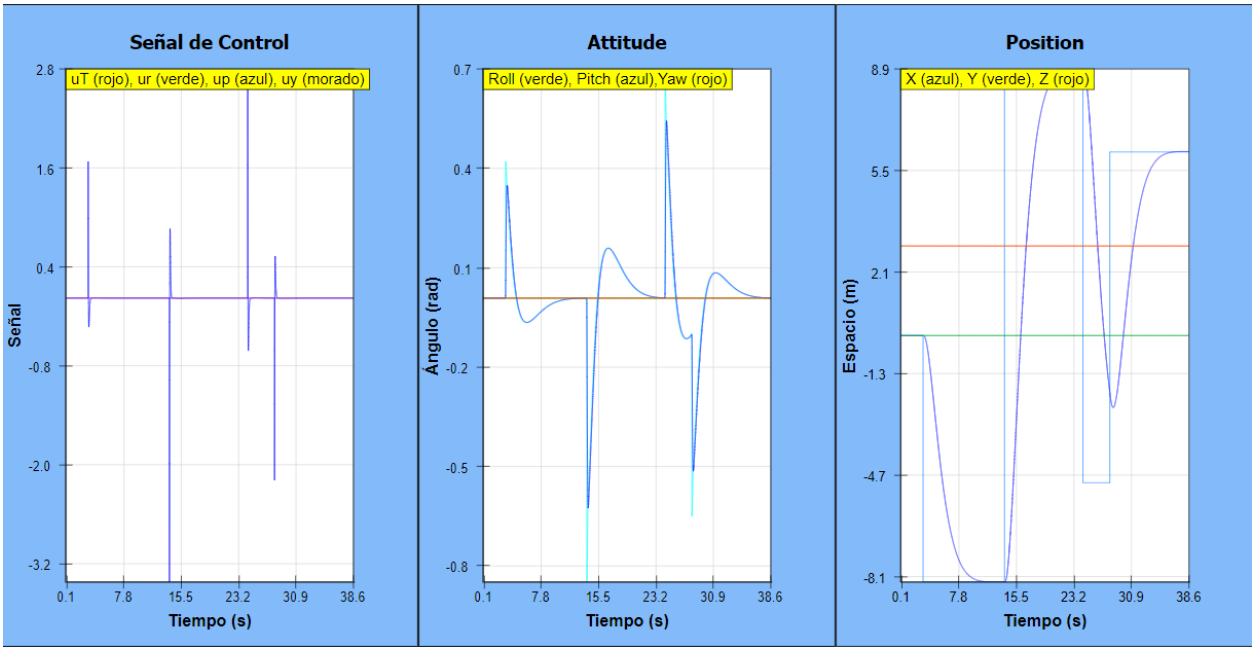


Figura 15: Control de la posición en el eje x

- Posición Y:

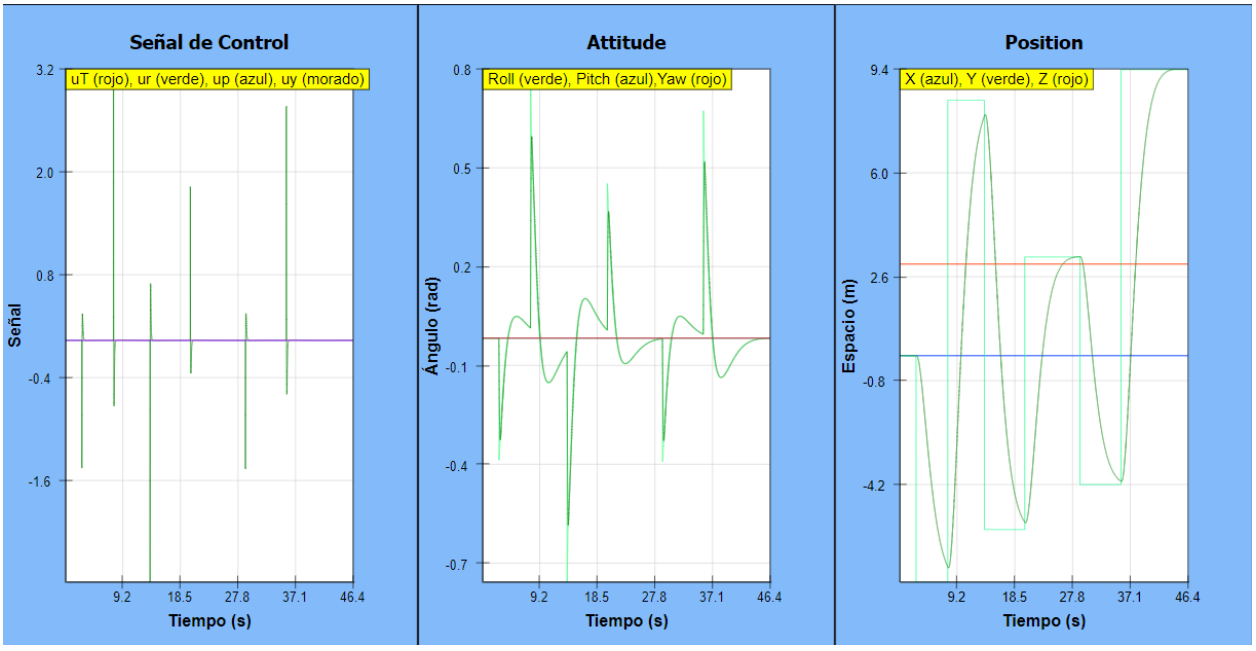


Figura 16: Control de la posición en el eje y

- Posición XYZ:

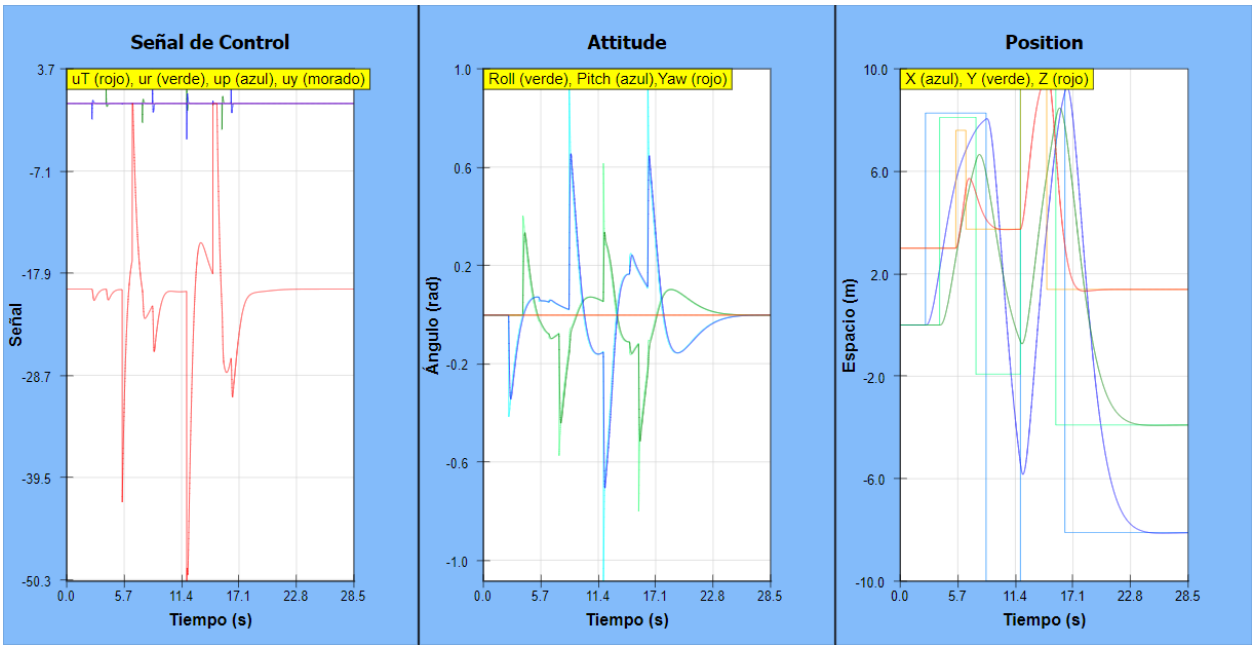


Figura 17: Control de la posición espacial

Para obtener este control se han establecido los parámetros del controlador PD de la posición como sigue:

Eje	K_p	K_d
x	0.05	0.125
y	0.05	0.125

Cuadro 4: Constantes del Position Control PD

En este caso, observamos como la constante de proporcionalidad es muy pequeña precisamente para que los ángulos que se establecen no sean excesivamente grandes dando lugar a un descontrol en todo el sistema.

5.1. Pure Pursuit

Realizando unas cuentas modificaciones en el código se ha implementado un algoritmo que permite desplazarse en grandes distancias sin límites. En las figuras anteriores se muestran posiciones en el intervalo $[-10, 10]$ para X e Y y en el intervalo $[0{,}5, 10]$ en Z. Con este algoritmo se pueden recorrer grandes distancias sin oscilaciones. Este viene dado por el siguiente código:

```
1 // PurePursuit:
2 if (PurePursuit){
3 eX_pursuit=X_pursuit-X;
```

```
eY_pursuit=Y_pursuit-Y;
eZ_pursuit=Z_pursuit-Z;

var Rlim=1;
Rz = Math.max(getBaseLog(10, Math.abs(eZ_pursuit)),1);

if (Rz<=1){
Rx = Math.max(getBaseLog(10, Math.abs(eX_pursuit)),1);
}
else {Rx=Rlim;}


if (Math.abs(eX_pursuit)  > LA * Rx){
  X_ref= X + Math.sign(eX_pursuit) * LA * Rx;
  }
else {
X_ref=X_pursuit;
};


if (Rz<=1){
Ry = Math.max(getBaseLog(10, Math.abs(eY_pursuit)),1);
}
else {Ry=Rlim;}
if (Math.abs(eY_pursuit) > LA * Ry){
  Y_ref= Y + Math.sign(eY_pursuit) * LA * Ry;
  }
else {
Y_ref=Y_pursuit;
};


if (Math.abs(eZ_pursuit)  > LA * Rz){
Z_ref= Z + Math.sign(eZ_pursuit) * LA * Rz;
}
else {
Z_ref=Z_pursuit;
};
}
```

Se puede observar como el código está plagado de condiciones que buscan optimizar el rendimiento del desplazamiento del dron intentando evitar las oscilaciones y sobrepasar la marca. Esto se hace con la motivación de un posible futuro desarrollo que debe evitar objetos de mayor o menor tamaño en la trayectoria.

A continuación, se muestran una serie de destinos establecidos y la evolución de las gráficas para ellos. Se ha añadido además una señal para el ángulo *yaw* de manera análoga a los de *pitch* y *roll* para intentar mitigar oscilaciones producidas por una gran velocidad en el eje z.

- Gran recorrido, $(X, Y, Z) = (-300, 2300, 150)$:

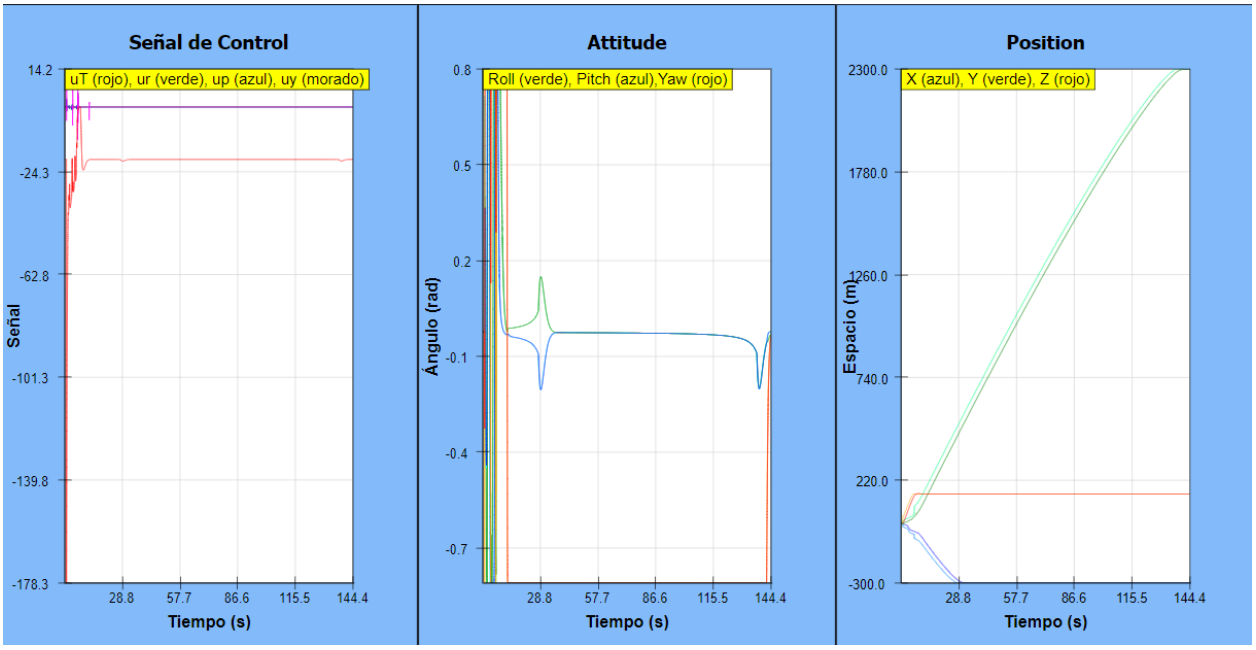


Figura 18: Pure Pursuit para $(x, y, z) = (-300, 2300, 150)$

- Gran subida, $(X, Y, Z) = (10, -24, 320)$:

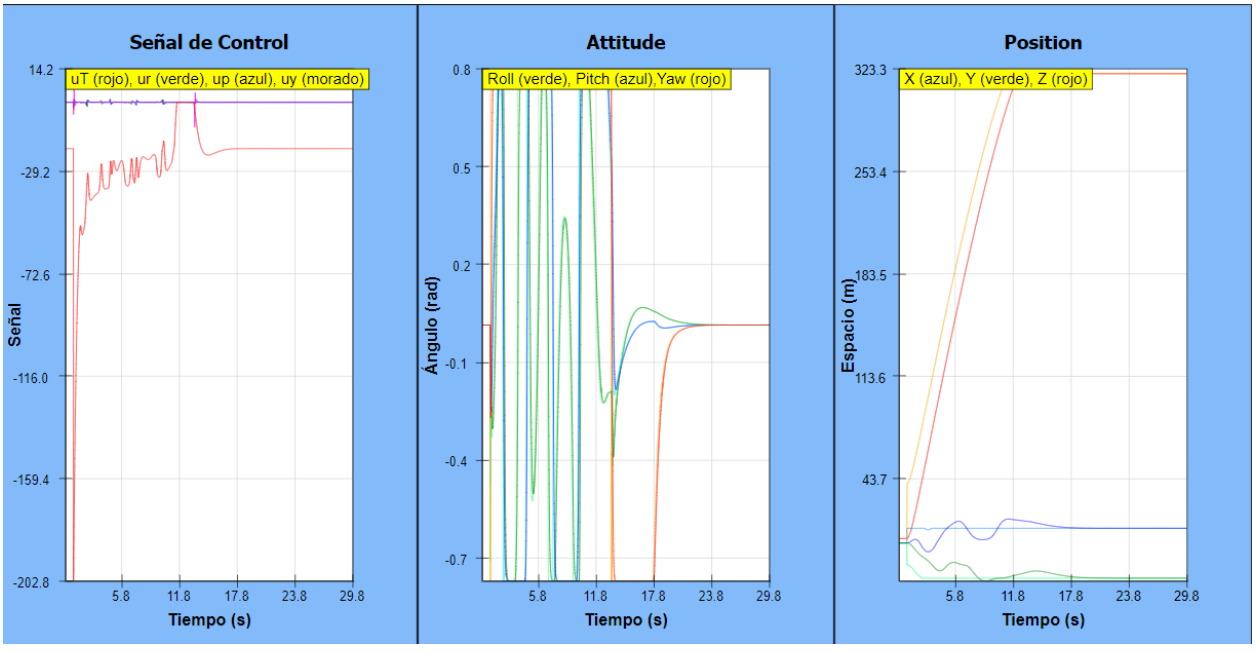


Figura 19: Pure Pursuit para $(x, y, z) = (10, -24, 320)$

- Gran desplazamiento en el plano XY, $(X, Y, Z) = (-1523, 120, 24)$;

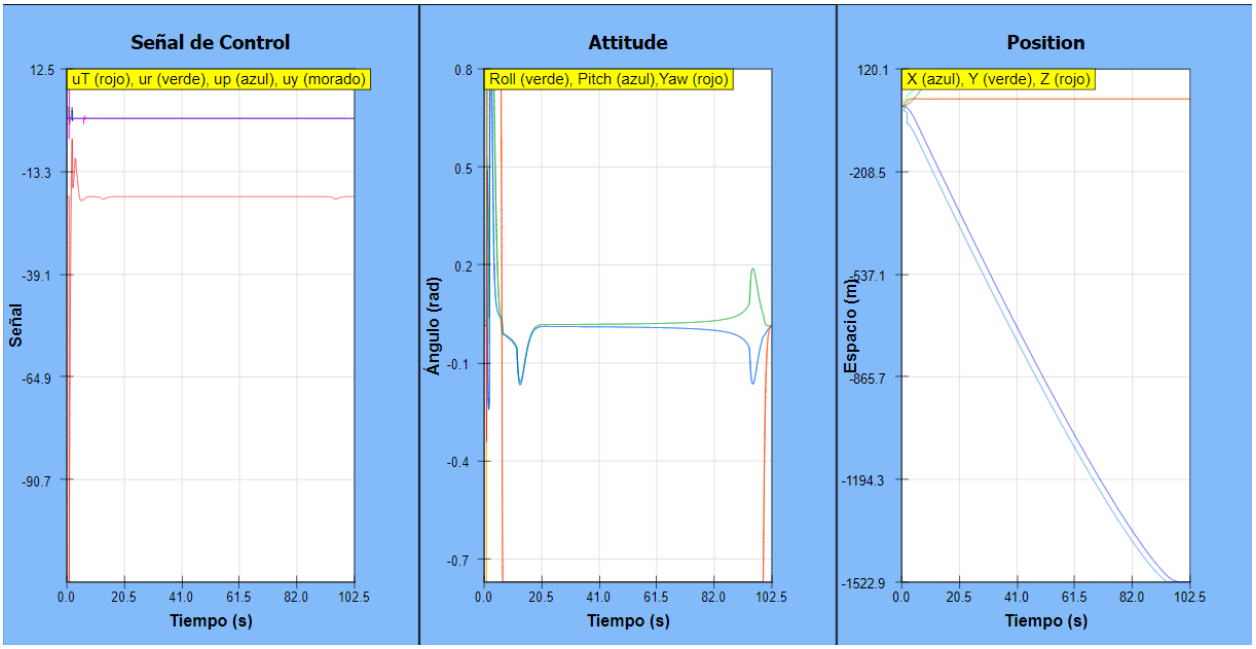


Figura 20: Pure Pursuit para $(x, y, z) = (-1523, 120, 24)$

Observamos que funciona francamente bien y de manera muy suave, salvo en el caso de la gran subida. Una gran subida con escaso desplazamiento para x o y provoca unas señales incorrectas, produciendo un movimiento innecesario, sin embargo, las oscilaciones en XY cesan al alcanzar la altura deseada. Estas oscilaciones podrían ser más fuertes, pero para ello se ha introducido la señal para el ángulo *yaw*, además de un *Look Ahead* diferente para cada una de las coordenadas espaciales. Además, se han acotado los ángulos deseados entre $[-\frac{\pi}{4}, \frac{\pi}{4}]$ pues resulta más natural y evita un descontrol en el modelo.

6. Conclusiones

En este trabajo se ha estudiado la dinámico de un caudrirrotor mediante la segunda ley de Newton y la ecuación de movimiento de Euler. Mediante controles PD se ha conseguido controlar el movimiento del dron sin interferencia humana convirtiendo un simple cuadrirrotor en un UAV. De esta manera se ha conseguido que el dispositivo llegue a su destino en un tiempo aceptable y sin oscilaciones innecesarias. Se ha implementado de manera sencilla el método de Pure Pursuit que establece puntos de referencia a través del Look Ahead, hasta llegar al punto deseado o persiguiendo de manera infinita a un objetivo. En el estudio no se ha planteado la posibilidad de obstáculos, pero bastaría implementar el pure pursuit con ciertas condiciones para que evitase los obstáculos con holgura, además de habilitar en la interfaz la creación de obstáculos y establecer como los detecta el dron para evitar la colisión.

Tomando como base este documento y con muchas más horas de trabajo, se podría desarrollar el modelo permitiendo que un dron circulase por una ciudad de manera eficiente y autónoma. Cualquier estudio futuro exigiría el desarrollo de algoritmos más complejos.

Referencias

[1] Shi, Di Wu, Zhong Chou, Wusheng. *(2018). Harmonic Extended State Observer Based Anti-Swing Attitude Control for Quadrotor with Slung Load.* Electronics. 7. 83. 10.3390/electronics7060083. 1974.

[2] Bouabdallah, S. and Siegwart, R., *2007. Full control of a quadrotor.* 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems,.

[3] Manjunath, Abhishek *"Path Following by a Quadrotor Using Virtual Target Pursuit Guidance"(2016).* All Graduate Theses and Dissertations. 4990. https://digitalcommons.usu.edu/etd/4990

[4] Mohammad Ashraf, M., Zainal Abidin, M., Azimi Mahmud, M., Abd Rahman, M. and Nasser Saleh Motea, Z., *2018. Modelling and Parameters Identification of a Quadrotor Using a Custom Test Rig.* International Journal of Power Electronics and Drive Systems (IJPEDS), 9(2), p.865.

[5] Jayakrishnan, H., *2016. Position and Attitude control of a Quadrotor UAV using Super Twisting Sliding Mode.* IFAC-PapersOnLine, 49(1), pp.284-289.

[6] Corke, P., *2011. Robotics, Vision and Control. Springer Tracts in Advanced Robotics,.*

[7] Héctor Pérez León and Jesús Iván Maza Alcañiz, *Trabajo Fin de Máster. Máster en Ingenería Electrónica, Robótica y Automática. Generación y seguimiento de trayectorias para un vehículo aéreo multi-rotor*

[8] Wolfgang Christian and Francisco Esquembre *Easy Java/Javascript Simulations Manual.* November 4, 2015