

Índice general

1. Introducción	5
1.1. Objetivo	5
1.2. Antecedentes	5
1.3. Estado de la técnica	7
2. Modelado del sistema	9
2.1. Modelo	9
2.2. Parámetros	13
3. Descripción del controlador y simulaciones	17
3.1. Control PID y simulación	17
3.2. Modos deslizantes	24
3.3. Modos deslizantes de orden superior	28
3.3.1. Algoritmo supertwisting	28
3.3.2. Simulación	30
3.4. Derivador numérico	33
4. Arquitectura del cuadricóptero	37
4.1. Comunicación WiFi con el AR.Drone	38
4.2. Modelo 3D	40
4.3. Interfaz gráfica	43

Índice general

4.3.1. Indicador de altitud	44
4.3.2. Brújula	46
4.3.3. Indicador de horizonte	46
4.3.4. Indicador de la velocidad de los motores	46
4.3.5. Display de mensajes	47
5. Implementación, pruebas y resultados	49
5.1. Implementación con Python	49
5.1.1. Control de postura	49
5.2. Implementación con C	52
5.2.1. Control de postura	53
5.2.2. Control de altitud	57
5.2.3. Comparación con el controlador de fábrica	59
6. Conclusiones	65
6.1. Trabajo futuro	66
6.2. Código	67
A. Pseudocódigo	69
A.1. Algoritmo Supertwisting	69
A.2. Derivador numérico	70
A.3. Controlador PID	71

1 Introducción

1.1. Objetivo

El objetivo de este trabajo es implementar un controlador externo en un cuadricóptero comercial utilizando una computadora para la interconexión con el dispositivo vía *WiFi* y como plataforma de programación de una interfaz de usuario.

1.2. Antecedentes

Desde sus inicios a principios del siglo pasado, la aviación fue una actividad de riesgo debido a la peligrosidad de que los aviones sufrieran una falla en pleno vuelo, pues en la mayoría de los casos no se tenía la oportunidad de detenerse a corregir la falla. Dado que gran parte de su crecimiento se ha dado en torno a los desarrollos militares, además de que las tecnologías necesarias, como de posicionamiento, procesamiento y sensado han avanzado a grandes pasos en las últimas décadas, se ha dado origen a los vehículos aéreos no tripulados, por un lado buscando evitar pérdidas humanas en el ámbito militar, pero por otro, aprovechando las capacidades y flexibilidad que pueden tener estos vehículos, ya que al no transportar personas, pueden ser más pequeños, maniobrables, o tener mayor autonomía [1]. Existen fundamentalmente dos categorías dentro de los vehículos aéreos: de despegue y aterrizaje horizontal, y de despegue y aterrizaje vertical. Entre los primeros se encuentran los aviones; entre los segundos hay una mayor variedad, como son los globos aerostáticos, cohetes, helicópteros e incluso algunos aviones que también tienen la capacidad de vuelo

1 Introducción

estacionario. Una de las ventajas principales de los segundos, es el poco espacio requerido para las operaciones de aterrizaje y despegue, y también la capacidad de mantenerse en vuelo estático. Los helicópteros, tal vez los vehículos más comunes dentro de esta categoría, tienen aún más variedades, pensando en el número de hélices dispuestas de manera vertical, como los de una hélice, doble hélice en un solo eje, doble hélice en ejes diferentes, cuatro hélices, etc. Ésta última variedad, que se conoce como cuadricóptero, es una de las que tiene más ventajas, ya que permite capacidades de carga relativamente grandes debido a sus cuatro motores, alta maniobrabilidad, simplicidad mecánica ya que no requieren mecanismos complejos en las hélices, capacidad de vuelo estacionario, etc [2]. Por esto, es una de las configuraciones que más popularidad ha ganado en los últimos años, ya sea para fines de vigilancia, mapeo, grabación de video, investigación, o simplemente entretenimiento.

Generalmente, la disposición de un cuadricóptero cumple con que son cuatro hélices con eje vertical dispuestas en forma de cruz, teniendo todas la misma distancia al centro y separados los brazos que las sostienen por un ángulo recto.

A diferencia de un helicóptero tradicional, el mecanismo de un cuadricóptero es bastante simple. Mientras que el helicóptero necesita variar la velocidad de la hélice, también se debe modificar el ángulo de ataque de las palas, así como el eje de giro, para así poder moverse en distintas direcciones en el espacio, es decir, adelante, atrás o a los lados. Además cuenta con una hélice de estabilización para contrarrestar el par aplicado a la hélice central; dicha hélice de estabilización también necesita un mecanismo complejo que le permita modificar el ángulo de ataque de las palas, cuyo efecto además debe ser contrarrestado por la hélice principal.

Un cuadricóptero tiene cuatro hélices de las que sólo es necesario variar la velocidad, pues los efectos del par de cada una de las hélices se contrarrestan con el de las dos hélices contiguas, y combinando el empuje en todas ellas, se obtiene la inclinación que le permite moverse en el espacio. Así, con todas las hélices montadas sobre un eje de giro fijo y sin necesidad de modificar el ángulo de ataque de las palas de las hélices, se tiene un sistema mecánicamente simple, pero que como desventaja tiene el mayor gasto de energía y la inestabilidad inherente que necesita de un buen sistema de control que le permita volar.

Existe una gran cantidad de métodos de control que pueden usarse para controlar un cuadricóptero, entre los que se cuentan, por ejemplo, control PID, control adaptable, control robusto, control inteligente, entre otros.

1.3. Estado de la técnica

Un cuadricóptero es un sistema inherentemente inestable y además es un sistema complejo, pues cuenta con múltiples entradas y múltiples salidas, además de no linealidades. Es por eso que el modelado del sistema es una parte fundamental si se desea controlarlo. En [2] se desarrolla un modelo mediante ambos métodos, de Euler-Lagrange y de Newton-Euler, aunque se obtiene un modelo más realista mediante el método de Newton-Euler, pues además se toman en cuenta las fuerzas aerodinámicas. Este modelo se retoma en [3], sin embargo, para el diseño del controlador se simplifica, omitiendo estas fuerzas aerodinámicas, por lo que quedan sólamente los efectos giroscópicos del movimiento del propio cuadricóptero, del giro de las hélices y las fuerzas de entrada al sistema, que es una forma muy utilizada en los modelos presentados en la literatura [5, 7, 8, 9].

Un método de control para sistemas lineales es el control óptimo, que minimiza una función de costo diseñada para el sistema, de manera que para llevar al sistema de un punto inicial a un punto final, el costo sea mínimo, ya sea en tiempo o en energía, o de la forma que se haya diseñado dicha función [4]. En [5] se explica un método de control no lineal para un cuadricóptero basado en un controlador óptimo. Para esto se factoriza el modelo matemático del sistema no lineal, de modo que tenga la forma de un sistema lineal en ecuaciones de estado y se pueda aplicar la técnica de control óptimo, pues éste método funciona para sistemas lineales. El problema en este caso es el hecho de que el sistema no lineal, factorizado para tener la forma de sistema lineal, varía en el tiempo, por lo que hay que calcular las ganancias para el control óptimo a cada instante.

El uso de métodos de inteligencia artificial como redes neuronales también ha tenido lugar en la implementación de controladores para vehículos aéreos no tripulados. En [6] se presenta la técnica de control mencionada anteriormente, presentada en [5], ahora asistida por una red neuronal de una capa oculta, cuyo propósito es resolver un sistema de cinco ecuaciones no lineales con cuatro incógnitas. Las cinco ecuaciones están definidas por las entradas al sistema, que son las fuerzas y pares generados por la combinación de la acción de las hélices del cuadricóptero, además de la suma aritmética de las velocidades de las hélices, para tomar en cuenta el efecto giroscópico de las cuatro hélices como una señal de entrada. Las cuatro incógnitas son las velocidades de cada una de las hélices, que son en realidad las cuatro entradas al sistema.

Además existen métodos de control robusto, diseñados para ser capaces de controlar un sistema a pesar de perturbaciones externas reales o provocadas por ruido en los sensores, o a pesar de incertidumbres paramétricas y/o inexactitudes en el modelo matemático. Uno

1 Introducción

de ellos se presenta en [8], que implementa un controlador PID con integrador y derivador de orden no entero emulado a través de una red neuronal de una capa oculta que asiste en el cálculo de coeficientes para un impulso de respuesta finita, de modo que es capaz de calcular la salida del controlador en un tiempo menor que con un controlador PID de orden fraccional real, es decir, implementa un método de cálculo computacionalmente simple.

Otro método de control robusto es el de modos deslizantes, abordados en [10], donde se utilizan modos deslizantes de orden superior con el fin de que el controlador sea capaz de contrarrestar perturbaciones externas e incertidumbres paramétricas, y en cuyo trabajo está basada esta tesis.

Para lograr un control en la posición espacial del cuadricóptero, y posteriormente el seguimiento de rutas, es necesario también lograr una medición precisa de la posición dentro de un marco de referencia. Esto puede ser logrado con un dispositivo de posicionamiento global, como se presenta en [11], donde se utiliza un observador por modos deslizantes de orden superior, basado en el derivador numérico por modos deslizantes de orden superior presentado en [20], cuyo trabajo es utilizado en esta tesis. El observador se utiliza debido a la incapacidad de un derivador numérico convencional de entregar valores útiles, pues amplifica el ruido en los sensores. El observador, a partir de los datos obtenidos por un sistema de posicionamiento global diferencial (DGPS, por sus siglas en inglés), un sonar y una brújula digital, estima las velocidades, aceleraciones, ángulos y velocidades angulares, y estos son alimentados a un controlador con linealización por retroalimentación, que junto con el observador, logra la robustez y estabilidad deseada.

Uno de los impedimentos de los sistemas de posicionamiento global es el pobre o nulo desempeño en interiores, y si se desea aprovechar la capacidad de un cuadricóptero de volar en interiores, es necesario implementar un sistema diferente. La manera más obvia es el uso de cámaras fijas en el sistema de referencia que identifiquen al vehículo y lo ubiquen en el espacio. En [12] se utilizan un par de cámaras web con resolución VGA, que son capaces de identificar las marcas brillantes montadas en el cuadricóptero a una distancia de hasta cinco metros. El cuadricóptero también cuenta con sensores a bordo, como acelerómetros y giroscopios, gracias a los cuales el cuadricóptero puede ser estabilizado en cuanto a los ángulos de ladeo y cabeceo, pero no los otros cuatro grados de libertad, es decir, posición espacial y orientación en el eje vertical. Para este propósito son usadas las cámaras, que identifican marcas de colores montadas en el cuadricóptero para luego calcular las correspondencias de una cámara con la otra, necesarias para la reconstrucción 3D.

2 Modelado del sistema



Figura 2.1 Cuadricóptero.

2.1. Modelo

Para poder diseñar un sistema de control, es necesario tener un modelo matemático del sistema físico que se desea controlar, que sea lo suficientemente acertado al menos al rededor de la zona de operación del sistema. Es por eso que se revisó la literatura correspondiente y se eligió el modelo matemático que se explica a continuación.

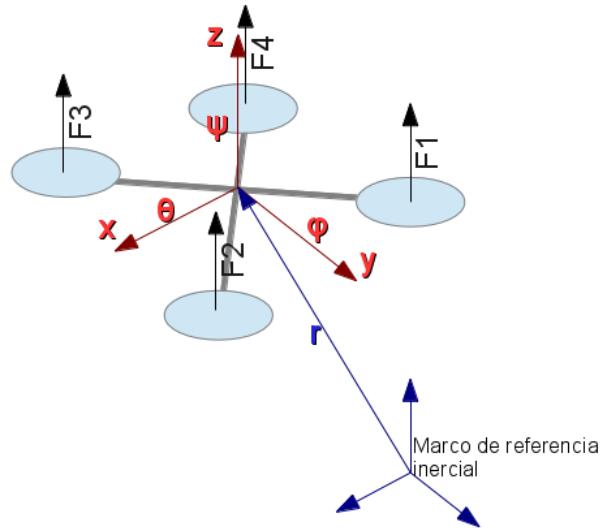


Figura 2.2 Modelo simplificado y marco de referencia.

En la figura 2.1 se puede ver una imagen del cuadricóptero utilizado, con indicaciones de la numeración de los motores y su sentido de giro, mientras que en la Figura 2.2 se muestra un diagrama del cuadricóptero, en el que se ilustran las fuerzas de empuje de los motores y las coordenadas del sistema. Se considera un marco de referencia inercial y un marco de referencia fijo en el cuadricóptero, que se asume como un cuerpo rígido. La orientación del cuadricóptero está dada por los tres ángulos de Euler que son θ para el cabeceo, ϕ para el ladeo y ψ para el guiñado. Las transformaciones que se aplican a los vectores dentro del marco de referencia fijo en el cuerpo del cuadricóptero para tenerlos en el marco de referencia inercial, están dadas por la matriz de rotación R , donde c y s significan \cos y \sin respectivamente:

$$R = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi \\ s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \quad (2.1)$$

El modelo del cuadricóptero se puede dividir en dos subsistemas; uno de ellos es la posición del cuadricóptero en el espacio, que depende del segundo, su orientación. Para controlarlos se diseña un lazo cerrado de control interno y uno externo, donde el interno controla la postura y recibe los ángulos deseados del lazo externo, que controla la posición espacial [3,5,6,11]. Esto significa que no son independientes uno del otro, pues aunque la posición no

interviene en el sistema que incluye la orientación, la orientación sí influye en cómo cambia la posición.

El vector r que indica la posición del cuadricóptero dentro del marco de referencia inercial es:

$$r = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (2.2)$$

Dado que las únicas fuerzas que afectan la posición son aquellas que surgen por el efecto de empuje de los motores y de la gravedad, la ecuación diferencial que modela dicho fenómeno queda de la siguiente manera:

$$\ddot{r} = R \cdot b/m \sum \omega_i^2 \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - g \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad (2.3)$$

donde b es el coeficiente de empuje de los motores, m es la masa del cuadricóptero, ω_i es la velocidad angular de cada motor y g es la aceleración de la gravedad.

Para la orientación, se pueden separar los momentos que afectan a cada una de las coordenadas.

Los momentos relacionados con el ladeo son la actuación de ladeo, que se da al combinar el empuje de los motores uno con cuatro y dos con tres, y que es igual a $lb(-\omega_1^2 - \omega_4^2 + \omega_2^2 + \omega_3^2)$, donde l es la longitud del brazo de palanca, es decir, la distancia de cada motor al eje de giro del cuadricóptero; el efecto giroscópico del cuadricóptero: $\dot{\theta}\dot{\psi}(I_y - I_z)$, donde I_y e I_z son los momentos de inercia, con respecto al eje y y al eje z , respectivamente; y el efecto giroscópico de los motores: $J_R\dot{\theta}\omega_r$, donde J_R es el momento de inercia de las hélices con respecto al eje de giro de éstas y ω_r es la suma aritmética de las velocidades angulares de las hélices.

Los momentos relacionados con el cabeceo son la actuación de cabeceo, que se da al combinar el empuje de los motores tres con cuatro y uno con dos, y es igual a $l \cdot b(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2)$; el efecto giroscópico del cuadricóptero: $\dot{\phi}\dot{\psi}(I_z - I_x)$; y el efecto giroscópico de los motores: $J_R\dot{\phi}\omega_r$.

Los momentos relacionados con el guiñado son la actuación de guiñado, que se obtiene de la combinación del contrapar de los motores uno con tres (que giran en el mismo sentido) y dos con cuatro (que giran en sentido contrario a uno y tres), y se calcula con $d(\omega_1^2 +$

2 Modelado del sistema

$\omega_3^2 - \omega_2^2 - \omega_4^2$), donde d es el coeficiente de arrastre de las hélices; el efecto giroscópico del cuadricóptero $\dot{\theta}\phi(I_x - I_y)$; y por último el par para vencer la inercia de las hélices $J_R \sum \omega_i$.

En el modelo obtenido de la literatura, se considera un sistema de referencia inercial y un sistema de referencia fijo en el cuadricóptero, cuyo origen coincide con su centro de masa [1].

$$\dot{w} = \begin{bmatrix} w_2 \\ (\cos w_7 \sin w_9 \cos w_{11} + \sin w_7 \sin w_{11}) u_1 / m \\ w_4 \\ (\cos w_7 \sin w_9 \cos w_{11} - \sin w_7 \cos w_{11}) u_1 / m \\ w_6 \\ -g + (\cos w_7 \cos w_9) u_1 / m \\ w_8 \\ w_{12} w_{10} I_1 - \frac{J_R}{I_x} w_{10} \omega_d + \frac{l}{I_x} u_2 \\ w_{10} \\ w_{12} w_8 I_2 + \frac{J_R}{I_y} w_8 \omega_d + \frac{l}{I_y} u_3 \\ w_{12} \\ w_{10} w_8 I_3 + \frac{1}{I_z} u_4 \end{bmatrix} \quad (2.4)$$

con

$$\begin{aligned} I_1 &= \frac{I_y - I_z}{I_x} \\ I_2 &= \frac{I_z - I_x}{I_y} \\ I_3 &= \frac{I_x - I_y}{I_z} \\ \omega_d &= \omega_2 + \omega_4 - \omega_1 - \omega_3 \end{aligned} \quad (2.5)$$

y

$$w^T = [x \ \dot{x} \ y \ \dot{y} \ z \ \dot{z} \ \phi \ \dot{\phi} \ \theta \ \dot{\theta} \ \psi \ \dot{\psi}] \quad (2.6)$$

El ángulo θ es positivo cuando el cabeceo es hacia atrás, es decir, cuando la nariz del cuadricóptero sube, y negativo cuando ésta baja. El ángulo ϕ es positivo cuando el cuadricóptero se ladea a la izquierda y negativo cuando es a la derecha. El ángulo ψ aumenta cuando el cuadricóptero gira en el sentido de las manecillas del reloj y disminuye cuando gira en sentido contrario. Por esto, la entrada que afecta positivamente al ángulo θ aumenta la velocidad de los motores uno y dos, y disminuye la de los motores tres y cuatro, o viceversa, para afectar al ángulo negativamente; la entrada que afecta positivamente al ángulo ϕ

aumenta la velocidad de los motores dos y tres, mientras disminuye la de los motores uno y cuatro, y al contrario para afectar al ángulo negativamente; la entrada que afecta al ángulo ψ positivamente aumenta la velocidad de los motores dos y cuatro, y disminuye la de los motores uno y tres, y de nuevo, al contrario para afectar negativamente a dicho ángulo. De acuerdo con esta descripción, el vector u de entradas queda de la siguiente forma:

$$u = \begin{bmatrix} b(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \\ b(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2) \\ b(-\omega_1^2 - \omega_4^2 + \omega_2^2 + \omega_3^2) \\ d(\omega_1^2 + \omega_3^2 - \omega_2^2 - \omega_4^2) \end{bmatrix} \quad (2.7)$$

donde ω_i representa la velocidad angular de cada uno de los rotores, b es el coeficiente de empuje de las hélices y d es el coeficiente de fricción con el viento.

2.2. Parámetros

Para poder tener completo el modelo matemático del cuadricóptero es necesario darle un valor a los parámetros especificados en la sección anterior. Uno de ellos es el coeficiente de empuje de las hélices, que es el valor con el que se convierte la velocidad de las hélices en fuerza y se puede ver en la ecuación 2.3. Se diseñó un experimento para convertirlo, utilizando directamente el valor digital de la velocidad de las hélices en vez del valor real, pues es más fácil obtener el valor de la señal PWM que se envía al cuadricóptero.

PWM significa, por sus siglas en inglés, modulación de ancho de pulso (Pulse Width Modulation), y es la manera digital de emular el comportamiento analógico de una señal, pues en el ámbito digital, los valores de una señal sólo pueden tomar dos valores: prendido o apagado. La idea detrás del PWM es que el voltaje promedio de la señal digital en un pequeño intervalo de tiempo sea igual al voltaje que se podría obtener con una señal analógica. Para que esto funcione, la frecuencia de la señal digital debe ser lo suficientemente alta, de manera que los motores no se prendan y apaguen cada que la señal digital también lo hace. Matemáticamente, dada la señal $f(t)$, el valor promedio en un período de tiempo T está dado por:

$$\bar{y} = \frac{1}{T} \int_0^T f(t) dt$$

2 Modelado del sistema



Figura 2.3 Medición del coeficiente de empuje.

Colocando al cuadricóptero en una balanza, como se muestra en la Figura 2.3, se puede medir su peso, y al disminuir este debido al empuje de los motores es posible conocer la fuerza impresa por el motor correspondiente a la velocidad de la hélice.

Se relacionaron las señales de PWM de los cuatro motores, al cuadrado y sumadas, con la suposición de que corresponden linealmente a una velocidad angular. Debido a que la fuerza que proporcionan las hélices es proporcional al cuadrado de su velocidad, se hizo un cambio de variable, sustituyendo la velocidad angular al cuadrado como x en la forma general de la recta $y = mx + c$. Una vez con todo en orden, se realizó la regresión lineal y se obtuvo una pendiente -representada con m en la forma general de la recta- de $6.549007669092e - 6$ [N/PWM²], que en el modelo está representada con la letra b y un valor de correlación de 0.993179217656, lo que significa que la recta obtenida teóricamente explica de manera adecuada los datos obtenidos experimentalmente.

En la Figura 2.4 se pueden ver las curvas obtenidas experimental y teóricamente, donde el eje de las x es la velocidad angular como valor de PWM, y el eje de las y es el empuje que imprime la hélice en Newtons.

La distancia del brazo de palanca, es decir, la distancia entre el centro de la hélice, que es donde se ejerce la fuerza de empuje, a cada eje del cuadricóptero se midió simplemente con una regla.

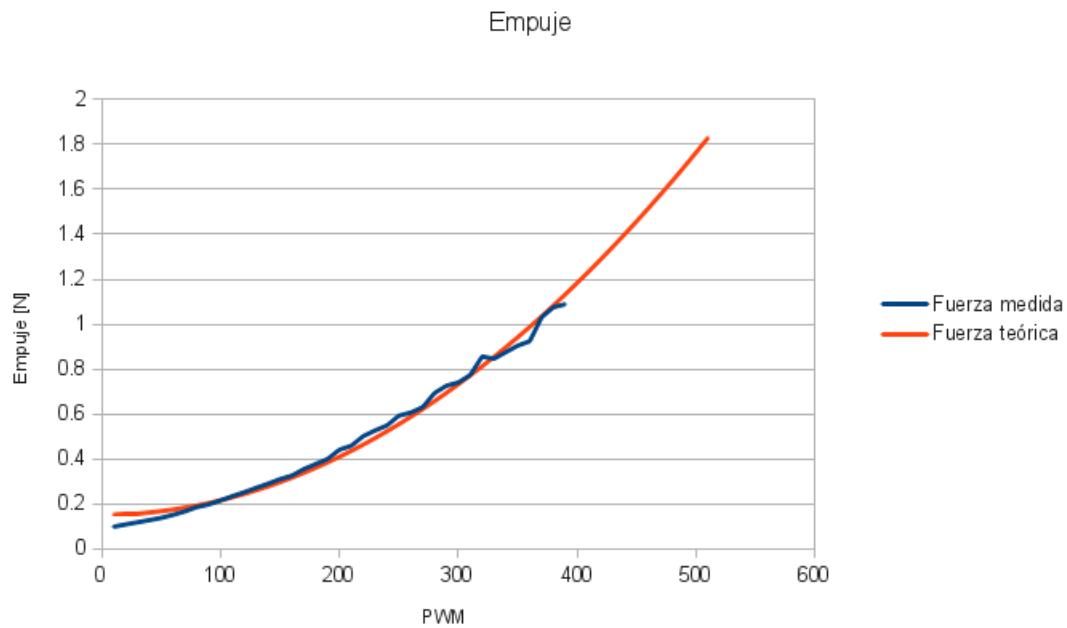


Figura 2.4 Empuje medido y regresión lineal.

Los momentos de inercia, que son los valores más complicados de obtener, se encontraron en [15], y fueron obtenidos realizando un modelo 3D del cuadricóptero.

El peso del cuadricóptero fue medido en el mismo experimento en el que se midió el coeficiente de empuje.

3 Descripción del controlador y simulaciones

Se decidió que un controlador adecuado para el cuadricóptero sería un controlador por modos deslizantes, debido a su robustez en cuanto a incertidumbres paramétricas y en cuanto a perturbaciones externas desconocidas. Sin embargo, inicialmente se probó un controlador PID, que es más simple de diseñar y de simular.

3.1. Control PID y simulación

El control PID es uno de los tipos de control más comunes debido en parte a su sencillez de diseño y de implementación pues sólo se necesita sintonizar tres ganancias para que funcione correctamente. El control proporcional (P) corrige el error actual y es la forma más simple de control después del control on/off. El problema del control proporcional es que, aunque corrige perturbaciones transitorias, no corrige perturbaciones permanentes, lo que crea un error de estado estable. Para corregir eso se agrega la parte integral (I), y al acumularse en el tiempo el error de estado estable, la parte integral lo corrige. Para mejorar el funcionamiento del controlador se agrega la parte derivativa (D), que funciona como una especie de predicción de los errores futuros, por lo que ayuda a disminuir el sobrepasso máximo.

El control PID es un controlador para sistemas lineales, por lo que si se desea usar con un sistema no lineal, es necesario linealizarlo al rededor de un punto de equilibrio, y si el punto

3 Descripción del controlador y simulaciones

de operación se aleja del punto de equilibrio, el controlador deja de ser capaz de estabilizar el sistema, pues el modelo linealizado deja de comportarse como el sistema real.

Un cuadricóptero, siendo un sistema no lineal, debe linealizarse para la implementación del controlador PID. Para esto se decide un punto de operación, es decir, se le da un valor deseado a cada uno de los estados que componen el sistema, que se denomina también punto de equilibrio y se representa con X^e .

Linealizar el sistema es obtener una función lineal que se comporte de manera similar a la función original, que es no lineal. Una forma de hacer esto es utilizando la serie de Taylor, que aproxima una función con una serie de términos de la siguiente forma:

$$f(x) \simeq \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x - a) \quad (3.1)$$

Si sólo se utiliza el segundo término y se recorren los ejes para que la función evaluada en 0 sea igual a 0, se tiene una función lineal con respecto a x que aproxima a la función original mientras esté cerca del punto $x = a$. Extendiendo esta idea a una función vectorial como la que representa el modelo matemático del cuadricóptero, se puede linealizar el sistema no lineal $\dot{w} = f(w, t)$, $v = h(w, t)$ para que tenga la forma de un sistema lineal en variables de estado, es decir:

$$\begin{cases} \dot{w} = Aw + Bu \\ v = Cw \end{cases}$$

Donde:

$$A = \left. \frac{\partial f}{\partial w} \right|_{w=W^e}$$

$$B = \left. \frac{\partial f}{\partial u} \right|_{w=W^e}$$

$$C = \left. \frac{\partial h}{\partial w} \right|_{w=W^e}$$

Dado que en este momento controlar los estados correspondientes a las coordenadas espaciales x e y no es parte del objetivo, se eliminarán del modelo linealizado, tomando en cuenta el resto de los estados sólamente. Así, se obtiene el sistema siguiente:

$$\left\{ \begin{array}{l} \dot{w} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} w + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1/m & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & l/Ix & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l/Iy & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/Iz & 0 \end{bmatrix} u \\ u = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 & g \end{bmatrix} \end{array} \right. \quad (3.2)$$

Donde:

$$w = \begin{bmatrix} z \\ \dot{z} \\ \phi \\ \dot{\phi} \\ \vartheta \\ \dot{\vartheta} \\ \psi \\ \dot{\psi} \end{bmatrix}$$

A partir de la ecuación 3.2 se puede observar que se obtienen cuatro sistemas desacoplados: uno para la altitud, otro para el ladeo, otro para el cabeceo y otro para el guiñado. Cada uno de los sistemas tiene la forma:

$$\dot{w} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} w + \begin{bmatrix} 0 \\ r \end{bmatrix} u$$

lo que los hace muy similares entre sí, pues resulta ser una masa sin amortiguación con una entrada para controlarla. Su función de transferencia es

$$G = \frac{r}{s^2} \quad (3.3)$$

y al realimentarse y agregar un bloque PID como se muestra en la Figura 3.1, la función

3 Descripción del controlador y simulaciones

de transferencia resulta

$$H = \frac{k_d s^2 + k_p s + k_i}{\frac{1}{r} s^3 + k_d s^2 + k_p s + k_i} \quad (3.4)$$

donde k_p , k_i y k_d son las ganancias proporcional, integral y derivativa, respectivamente.

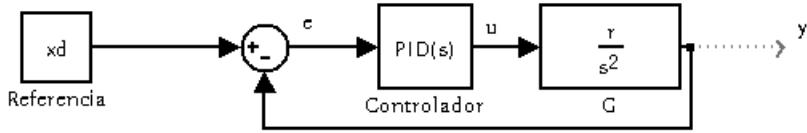


Figura 3.1 Diagrama de bloques del controlador PID.

Debido a la forma de la función de transferencia, se pueden asignar valores a cada una de las ganancias de modo que los polos del sistema se encuentren en cierto lugar deseado. Se decide inicialmente colocar los polos en -1 , -2 y -3 , para tener una respuesta estable, sin oscilaciones y que además no es tan exigente con la señal de entrada, es decir, requiere relativamente poca fuerza debido a que el polo dominante, el que se encuentra en -1 , provoca que el sistema tarde cerca de 5 segundos en estabilizarse, lo que para un sistema mecánico puede ser algo lento. Para lograr que los polos se encuentren en los puntos deseados, es necesario tener un polinomio en el denominador de la función de transferencia como el siguiente:

$$s^3 + 6s^2 + 11s + 6 = 0 \quad (3.5)$$

Si se factoriza $\frac{1}{r}$ de cada una de las ganancias en el denominador de la ecuación 3.4, al igualar a cero el polinomio se puede eliminar ese $\frac{1}{r}$, y asignando el valor a las ganancias tal como el polinomio de la ecuación 3.5 y luego tomando en cuenta de nuevo el $\frac{1}{r}$ factorizado, se obtienen los siguientes valores:

$$\begin{cases} k_p = 11/r \\ k_i = 6/r \\ k_d = 6/r \end{cases}$$

que al aplicarse en cada uno de los subsistemas, sustituyendo r por su valor correspon-

diente da lugar a los cuatro controladores.

Para el controlador de altitud, dado que r es igual a $1/m$, es decir, el inverso de la masa del cuadricóptero, las constantes del controlador son $k_p = 11m$, $k_i = 6m$ y $k_d = 6m$. Mientras que el modelo en variables de estado es:

$$\begin{cases} \dot{w} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} w + \begin{bmatrix} 0 & 0 \\ 1/m & -1 \end{bmatrix} u \\ v = \begin{bmatrix} 1 & 0 \end{bmatrix} w \end{cases} \quad u = \begin{bmatrix} u_1 & g \end{bmatrix} \quad (3.6)$$

Para el controlador de cabeceo, con r igual a l/Ix , resultan las siguientes constantes: $k_p = 11\frac{Ix}{l}$, $k_i = 6\frac{Ix}{l}$ y $k_d = 6\frac{Ix}{l}$, para el sistema en variables de estado

$$\begin{cases} \dot{w} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} w + \begin{bmatrix} 0 \\ l/Ix \end{bmatrix} u \\ v = \begin{bmatrix} 1 & 0 \end{bmatrix} w \end{cases} \quad u = \begin{bmatrix} u_2 \end{bmatrix} \quad (3.7)$$

En cuanto al controlador de ladeo, cuyo valor de r es igual a l/Iy , resultan las siguientes constantes: $k_p = 11\frac{Iy}{l}$, $k_i = 6\frac{Iy}{l}$ y $k_d = 6\frac{Iy}{l}$, para el sistema en variables de estado

$$\begin{cases} \dot{w} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} w + \begin{bmatrix} 0 \\ l/Iy \end{bmatrix} u \\ v = \begin{bmatrix} 1 & 0 \end{bmatrix} w \end{cases} \quad u = \begin{bmatrix} u_3 \end{bmatrix} \quad (3.8)$$

Mientras que con el controlador de guiñado, r es igual a $1/Iz$, por lo que las constantes usadas son: $k_p = 11Iz$, $k_i = 6Iz$ y $k_d = 6Iz$, para el sistema en variables de estado

$$\begin{cases} \dot{w} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} w + \begin{bmatrix} 0 \\ 1/Ix \end{bmatrix} u \\ v = \begin{bmatrix} 1 & 0 \end{bmatrix} w \end{cases} \quad u = \begin{bmatrix} u_4 \end{bmatrix} \quad (3.9)$$

Cada una de las entradas u_i se define de la siguiente forma:

$$u_i = k_p e + k_i \int_0^t e d\tau + k_d \frac{de}{dt} \quad (3.10)$$

donde e representa el error entre el estado que se quiere controlar y su valor de referencia, es decir, el valor en torno al cual fue linealizado, el punto de equilibrio.

3 Descripción del controlador y simulaciones

Se simuló el controlador PID diseñado de forma que estabilizara al cuadricóptero en cuanto a la postura, sin tomar en cuenta la velocidad o posición espacial, excepto por la altitud. La prueba se realizó comenzando en el origen del marco de referencia como condición inicial, con velocidades iguales a cero, pero con una inclinación de treinta grados para el ángulo ϕ , de veinte grados para el ángulo ϑ y de dieciséis grados para el ángulo ψ . Además el controlador de altitud tenía que elevar el cuadricóptero a una altura de medio metro. El programa utilizado para realizar la simulación fue MATLAB®, aprovechando el poder y sencillez de Simulink® y de MATLAB® en general [16].

En la Figura 3.2 se muestra el resultado de la estabilización de los tres ángulos y en la Figura 3.3 se muestra el resultado en cuanto al control de altitud. Como se puede observar en la primer gráfica, aunque los ángulos tardan cerca de cinco segundos en estabilizarse, tardan aproximadamente un segundo en cruzar por primera vez el eje, es decir, llegar a cero grados, y por esto, el hecho de que la estabilización sea lenta no es algo grave, pues los ángulos se encuentran la mayor parte del tiempo relativamente cerca de cero grados.

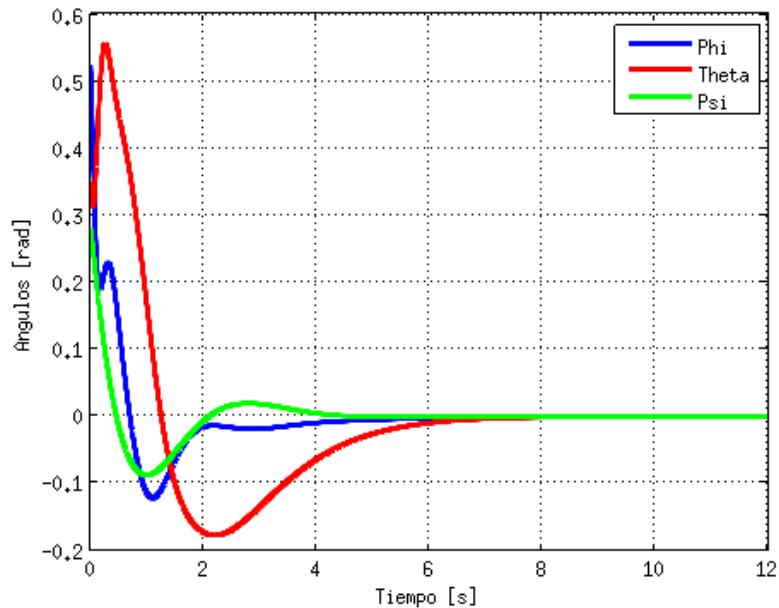


Figura 3.2 Simulación de estabilización de los ángulos.

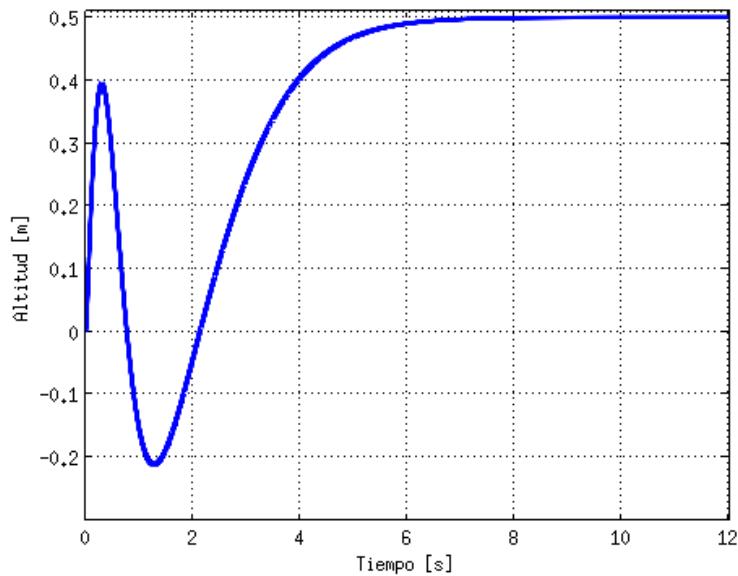


Figura 3.3 Primera simulación del control de altitud.

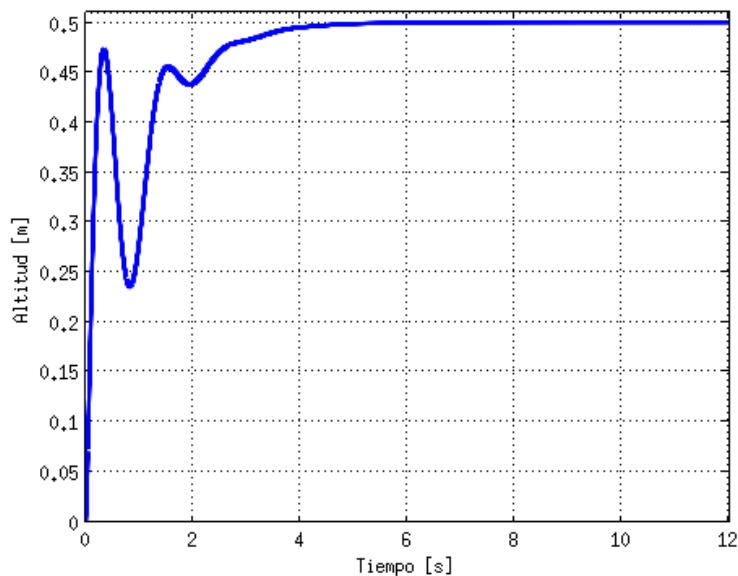


Figura 3.4 Simulación del controlador de altitud corregido.

3 Descripción del controlador y simulaciones

En cuanto al control de altitud, el resultado es más bien malo, pues aproximadamente un segundo después de comenzar el vuelo se registran valores negativos, lo que significa que el cuadricóptero llegó al suelo después de haberse elevado, cosa que además de ser un mal despegue, puede representar un golpe capaz de dañar la estructura del aparato. Por esto fue necesario reajustar las ganancias mediante prueba y error en simulaciones siguientes, de modo que se evitara esa caída después de despegar y los valores elegidos fueron 15 para k_p , 13 para k_i y 2 para k_d .

En la Figura 3.4 se puede ver el resultado de la prueba de vuelo con las ganancias del control de altitud reajustadas, que mejoran el tiempo de asentamiento, además de evitar la caída del cuadricóptero.

3.2. Modos deslizantes

Una manera de abordar los sistemas de control robusto son los llamados modos deslizantes, basados en la idea de que es más simple controlar sistemas de primer orden que sistemas de orden superior, ya sea con incertidumbres o no linealidades [17]. La idea es realimentar el sistema con ganancias variables que generan una ley de control que no es continua en el tiempo. Estas ganancias están diseñadas para que el sistema commute repetidamente entre una ganancia y otra, es decir, entre una ley de control y otra, de modo que ésta ley de control no se comporta completamente como ninguna de las dos, más bien se deslizará sobre la frontera entre las dos regiones.

Como ejemplo tomemos el sistema siguiente:

$$\begin{cases} \dot{w} = & \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} w + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \\ v = & \begin{bmatrix} 0 & 1 \end{bmatrix} w \end{cases}$$

cuya función de transferencia, una vez añadida la realimentación con ganancia k es la siguiente:

$$G(s) = \frac{1}{s^2 - s + 1 + k}$$

Para el ejemplo se eligen dos valores para la ganancia: -6 y 6 y en las figuras 3.5 y 3.6 se muestran los planos de fase de la función de transferencia correspondiente a cada ganancia, donde se puede notar rápidamente que el sistema realimentado con cualquiera

de las dos ganancias resulta en un sistema inestable, sin embargo, es posible combinar el comportamiento visto en ambos planos de fase al conmutar de una ganancia a otra y esto se hace al definir una línea de conmutación a partir de una variable s que, dependiendo de qué lado de la línea de conmutación se encuentre, será mayor o menor a cero. Para el ejemplo definimos la variable s de la siguiente forma:

$$s = w_2 + mw_1$$

donde m es la pendiente de la línea de conmutación y w_1 y w_2 son los estados del sistema.

Observando el plano de fase correspondiente a la ganancia negativa, se puede notar que al ser dividido a la mitad por cualquier línea que se defina, existen partes del plano que siguen siendo inestables y lo llevarían lejos de la línea, en vez de hacia ella. Para el plano de fase correspondiente a la ganancia positiva, en cualquier punto en el que se encuentre terminará llegando a la línea de conmutación, y una vez ahí, se deslizará sobre ella hasta llegar a estabilizarse sin abandonar la línea. Por esto, la línea de conmutación se conoce como superficie deslizante. Sin embargo aún hay que atender el hecho de que en ciertas partes del plano no tiende a acercarse a la superficie deslizante. Para esto se modifica la definición de s , de modo que el plano esté dividido en cuatro partes como se muestra en las figuras 3.5 y 3.6, y cada ganancia funcione dentro de las secciones que no están marcadas con una línea discontinua roja. Para esto la variable s se define como sigue:

$$s = w_1(w_2 + mw_1)$$

y la ley de control u es entonces:

$$u = \begin{cases} -k_1 e & \text{para } s < 0 \\ -k_2 e & \text{para } s > 0 \end{cases}$$

donde $k_1 = -6$, $k_2 = 6$ y $e = w_{1d} - w_1$.

3 Descripción del controlador y simulaciones

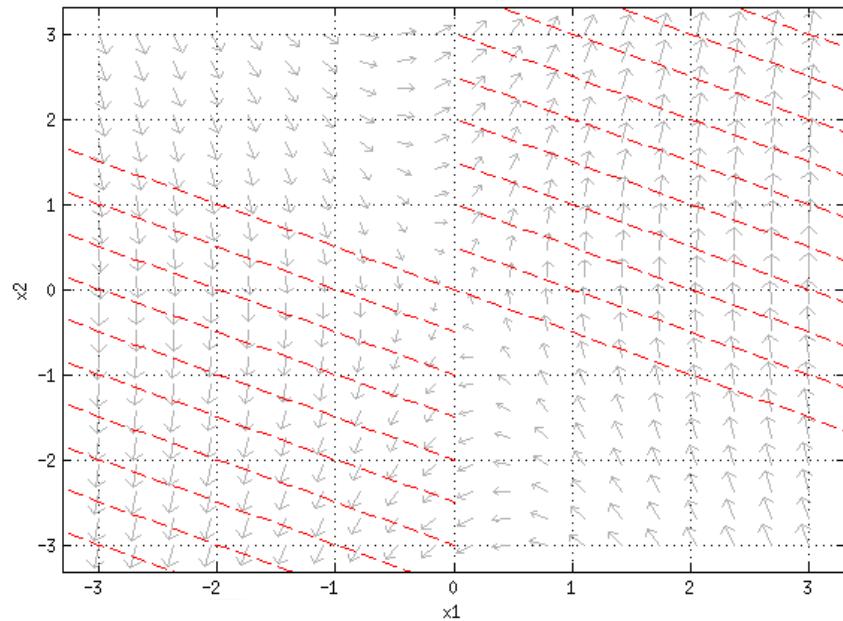


Figura 3.5 Plano de fase para la ganancia $k = -6$.

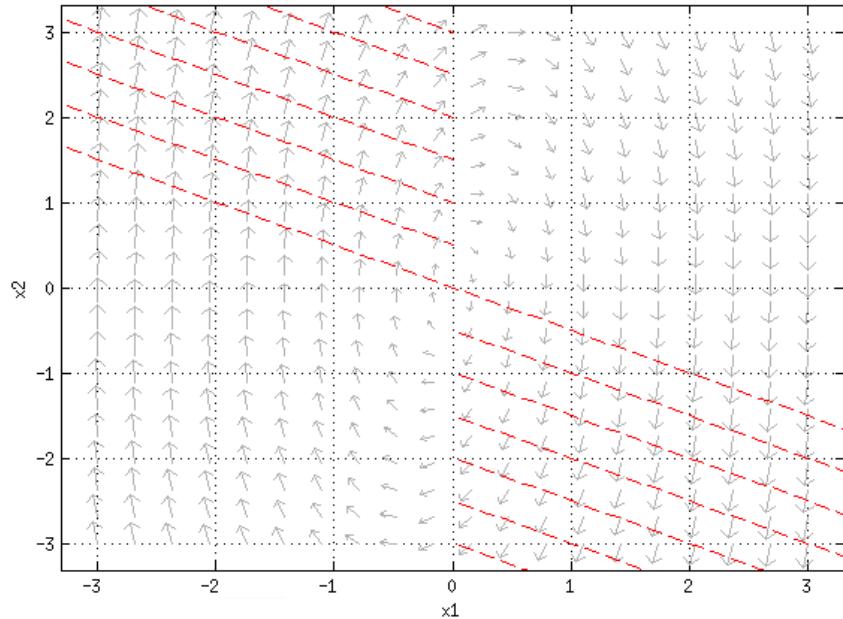


Figura 3.6 Plano de fase para la ganancia $k = 6$.

Ya con la ley de control definida, se simularon dos experimentos con condiciones iniciales diferentes. El resultado de las simulaciones se puede observar en la Figura 3.7, y por la trayectoria de cada experimento en el plano de fase se puede reconocer la combinación del comportamiento de los planos mostrados en las figuras 3.5 y 3.6.

Con esto podemos observar que para diseñar un controlador por modos deslizantes de primer orden, es necesario un correcto diseño de la superficie deslizante, de modo que ésta atraiga al sistema a ella para que una vez la toque, se deslice hacia el punto deseado [18].

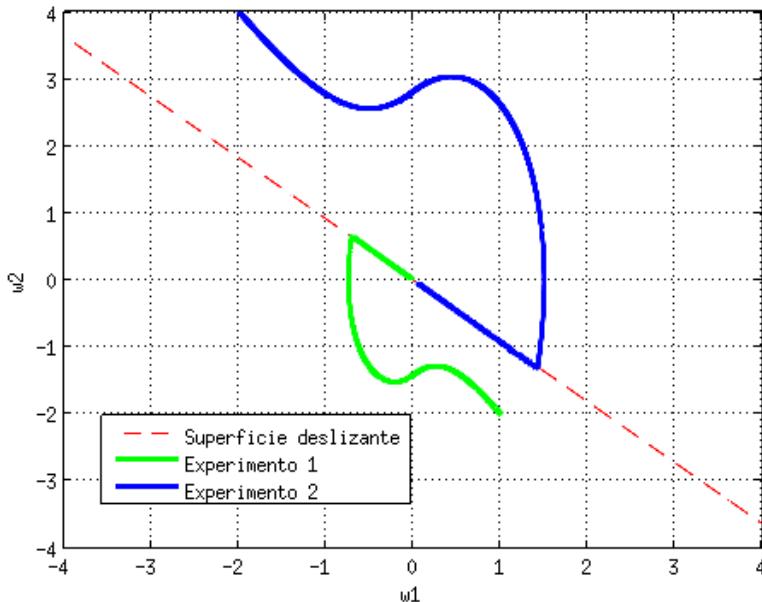


Figura 3.7 Simulación del modo deslizante.

Idealmente, la conmutación entre un comportamiento y otro ocurre a una frecuencia infinita y así, se tendría un deslizamiento sobre la superficie que genera una curva continua y mayormente suave sobre los estados, sin embargo, al ser imposible en la realidad lograr una frecuencia de conmutación infinita, realmente el sistema no se desliza sobre la superficie deslizante, sino que *salta* de un lado a otro de esta, lo que genera vibraciones en el sistema que pueden llegar a ser dañinas para él. A este fenómeno se le conoce como *chattering* y lo más deseable es evitarlo.

3.3. Modos deslizantes de orden superior

Mientras los controladores por modos deslizantes son capaces de estabilizar sistemas sujetos a condiciones de alta incertidumbre de una manera robusta, muestran también una clara desventaja que se conoce como *chattering*, que es el generar vibraciones de alta frecuencia en el sistema que se desea controlar y que pueden ocasionar daños en él. Es por esto que se desarrollaron los modos deslizantes de orden superior, que pretenden eliminar el efecto de *chattering* sin perder la efectividad y robustez de los modos deslizantes de primer orden. El orden del modo deslizante corresponde al número de derivadas continuas de s .

3.3.1. Algoritmo supertwisting

El algoritmo supertwisting propone una ley de control que atenúa los efectos de *chattering* generados por los cambios de alta frecuencia característicos de los modos deslizantes de primer orden [19], primero utilizando una función continua que sustituya a la función signo, que en este caso es la raíz cuadrada, como puede observarse en la Figura 3.8. En una segunda parte, utilizando una ley de control virtual, basada en la integral de un modo deslizante de primer orden.

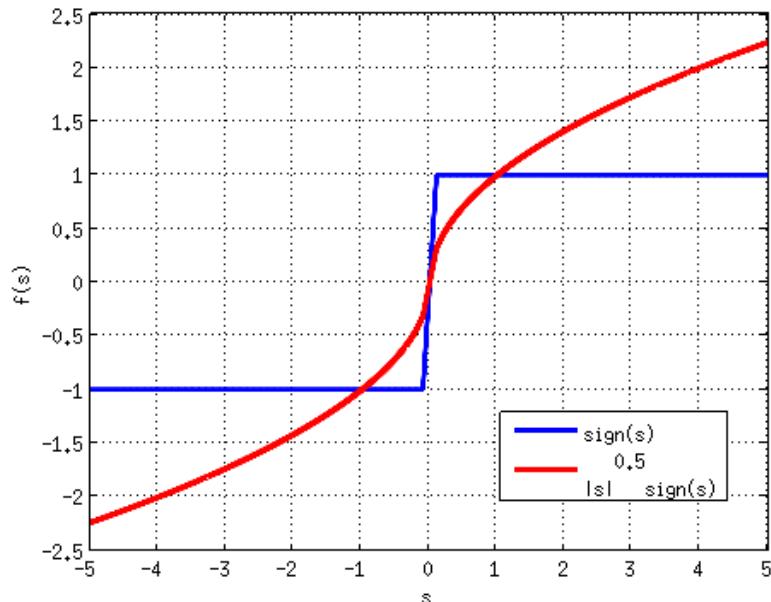


Figura 3.8 Comparación de función signo y raíz cuadrada.

3.3 Modos deslizantes de orden superior

La forma del algoritmo super twisting es la siguiente:

$$u = -k_1 |s|^{\frac{1}{2}} \operatorname{sign}(s) + v,$$

$$\dot{v} = -k_2 \operatorname{sign}(s)$$

donde k_1 y k_2 son las ganancias que se deben ajustar para que el controlador funcione bajo los siguientes criterios:

1)

$$k_1 > \sqrt{\frac{2}{K_m k_2 - C}} \frac{(K_m k_2 + C) K_M (1 + q)}{K_m^2 (1 - q)} \quad (3.11)$$

2)

$$K_m k_2 > C \quad (3.12)$$

donde K_m , K_M , C y q son constantes positivas tal que: $|\dot{a}| \leq C$, $0 \leq K_m \leq b \leq K_M$, $0 < q < 1$; donde a y b forman parte del sistema:

$$\dot{w} = a(t, w) + b(t)u \quad (3.13)$$

Tomando el modelo del cuadricóptero en forma de ecuaciones de estado de la forma siguiente:

$$\dot{w} = f(t, w) + g(t, w)u + w(t) \quad (3.14)$$

donde $w(t)$ representa un vector de perturbaciones externas, $g(t, w)$ representa una matriz de inercia y $f(t, w)$ es el modelo matemático del cuadricóptero. Dado que en realidad $g(t, w)$ es una matriz constante, pues los momentos de inercia no varían con el tiempo, K_m y K_M pueden tomar ese valor exacto, que para el caso del ángulo ϑ es 5.3942. Para la constante C se realizó una simulación y se graficó el cómo variaba $f(t, w)$ con respecto al tiempo y se observó que salvo algunos picos, en general se encontraba acotada a ± 10 , por lo que ese fue el valor que se le otorgó a C . Como q simplemente debía ser un valor entre 0 y 1 se le dio el valor de 0.5. Con estos valores se obtuvo que k_1 debía ser mayor a 18.4146 y a k_2 se le dio un valor de 2.

A partir de la forma básica del algoritmo super-twisting se puede obtener la siguiente ley de control que estabilice al modelo de la ecuación 3.14:

3 Descripción del controlador y simulaciones

$$u = g^{-1}(-\lambda \dot{e} - k_1 |s|^{1/2} sign(s) - k_2 \int_0^t sign(s)d\tau) \quad (3.15)$$

donde s es la variable deslizante que se define como sigue:

$$s = \dot{e} + \lambda e \quad (3.16)$$

y finalmente e y \dot{e} son el error y la derivada del error respectivamente, es decir:

$$e = w_d - w \quad (3.17)$$

$$\dot{e} = \dot{w}_d - \dot{w} \quad (3.18)$$

En la ecuación 3.15 se agrega el término $-\lambda \dot{e}$, para que ayude en tareas de seguimiento, y λ , que comparte en la ecuación 3.16 es una constante positiva que en este caso se eligió con un valor de 3, que significa que para la superficie deslizante es tres veces más importante corregir el error que su derivada.

3.3.2. Simulación

Una vez que se tuvieron todos los valores necesarios para la ley de control en la ecuación 3.15 se simuló con MATLAB, añadiendo un controlador PID que le indica los ángulos deseados al control de postura de modo que alcanzara un punto en el espacio, aprovechando la capacidad del controlador de postura de realizar seguimiento de una trayectoria en el tiempo. Se añadieron además dos perturbaciones; la primera un pulso de 0.2 segundos de duración y diez Newtons de fuerza a los dos segundos después de iniciada la simulación (de doce segundos que dura); la segunda un escalón de amplitud igual a 2.5 Newtons a partir de los cuatro segundos de simulación y que se mantiene hasta el final de ésta.

Se realizó la misma simulación con el controlador PID antes desarrollado para hacer una comparación y como puede verse en las gráficas siguientes, mientras el cuadricóptero con un controlador PID se aleja bastante del punto deseado debido a las perturbaciones, el cuadricóptero con controlador por modos deslizantes apenas *siente* las perturbaciones y se mantiene todo el tiempo muy cerca del punto deseado. Ambas pruebas fueron simuladas con una cantidad de ruido considerable y los dos controladores respondieron bien a pesar del ruido.

3.3 Modos deslizantes de orden superior

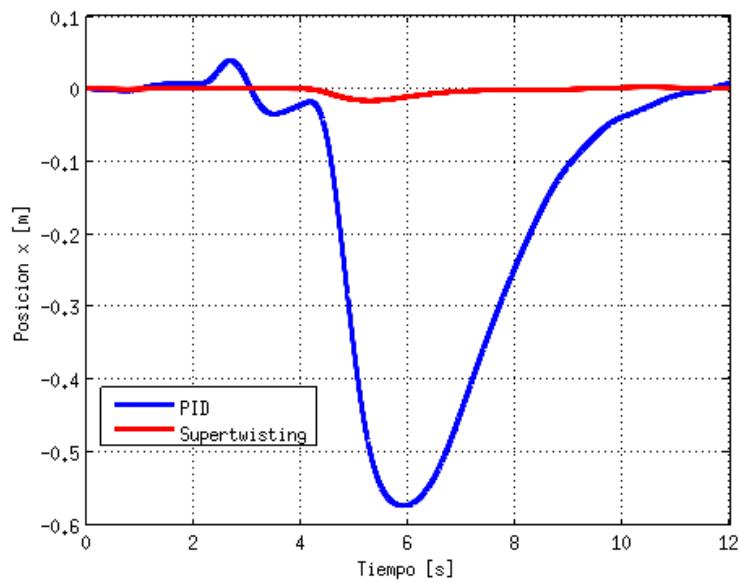


Figura 3.9 Comparación de control PID contra Super-twisting en eje x .

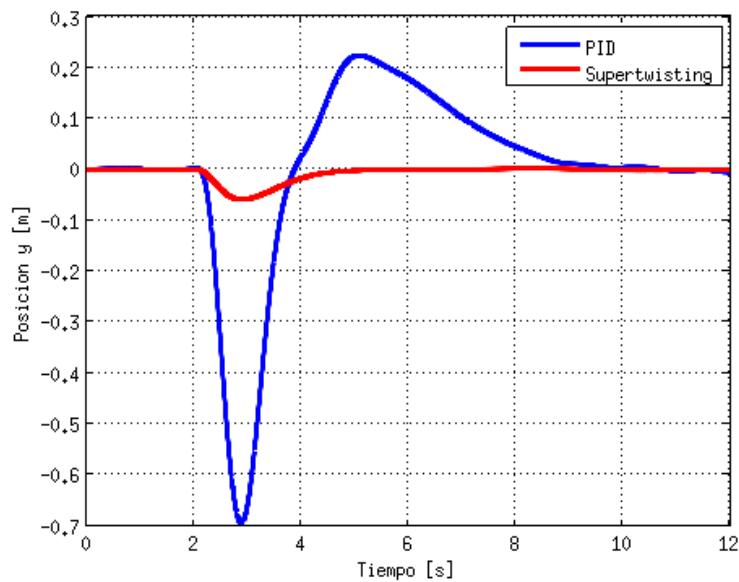


Figura 3.10 Comparación de control PID contra Super-twisting en eje y .

3 Descripción del controlador y simulaciones

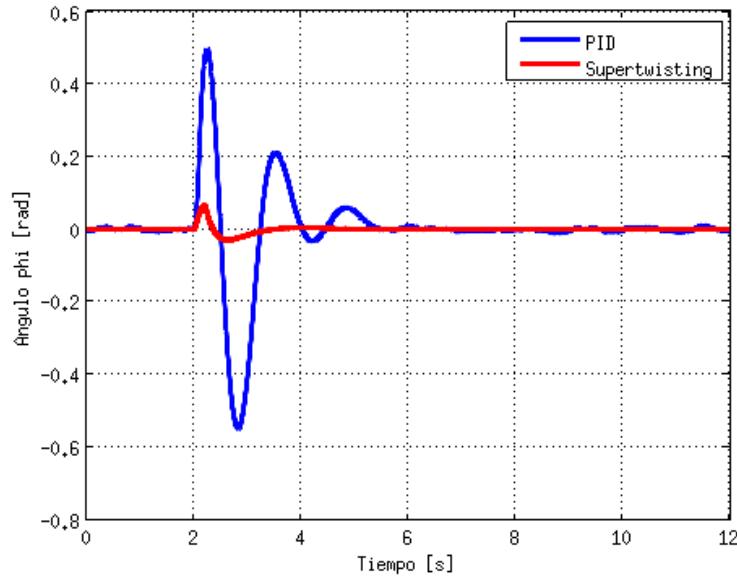


Figura 3.11 Comparación de control PID contra Super-twisting en ángulo ϕ .

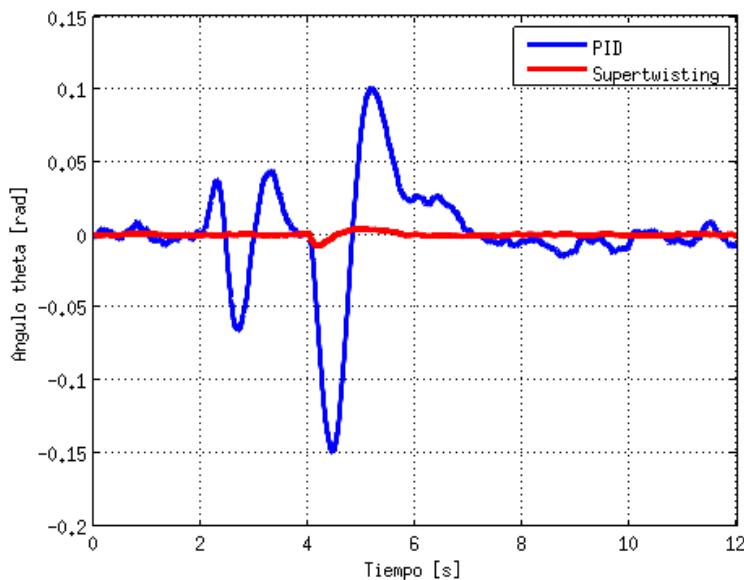
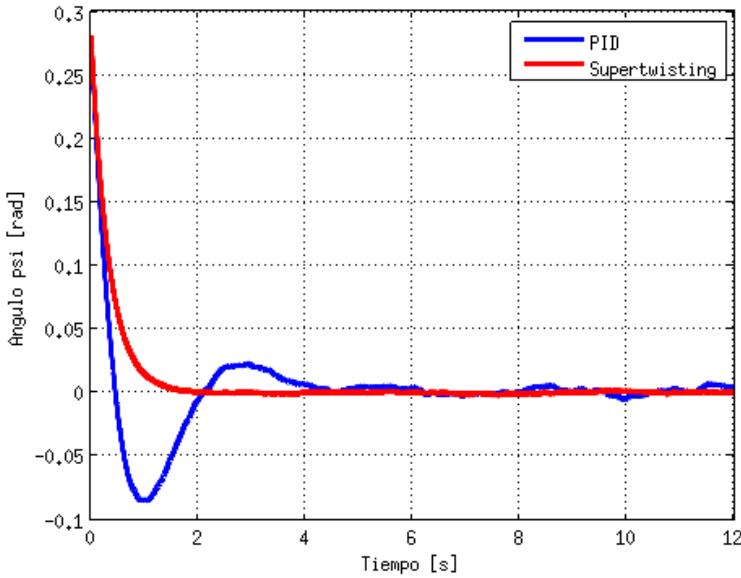


Figura 3.12 Comparación de control PID contra Super-twisting en ángulo θ .

Figura 3.13 Comparación de control PID contra Super-twisting en ángulo ψ .

3.4. Derivador numérico

Dado que el controlador propuesto, en la variable s requiere el valor de la velocidad angular, el cual no es proporcionado por el propio AR.Drone, fue necesario derivar la posición angular. Durante la implementación del controlador con Python, la señal obtenida del cuadricóptero relativa a la posición angular es una señal muy limpia, por lo que inicialmente se pensó en derivarla de la manera clásica $\dot{x} = \frac{x_n - x_{n-1}}{dt}$, y al momento de hacer pruebas y graficar los resultados de los experimentos, se pudo ver que las señales de control tenían algunos picos fuera de la curva separados por una distancia bastante regular a la vista. Esto era causado porque los datos recibidos no cambiaban cada ciclo del bucle del controlador, entonces la derivada calculada resultaba ser cero, y cuando se obtenía el nuevo dato, el valor de la derivada se disparaba, pues el diferencial de tiempo era muy pequeño para el cambio que había ocurrido en el ángulo. Por eso se decidió implementar un derivador numérico que además funciona como un filtro para la señal.

Supóngase el derivador no lineal siguiente:

$$\dot{z} = -k\text{sign}(z - f(t)) \quad (3.19)$$

3 Descripción del controlador y simulaciones

Donde z es la señal $f(t)$ filtrada, \dot{z} su derivada, y por lo tanto, la derivada de la función $f(t)$, que es lo que se desea encontrar, k es una constante positiva y *sign* es la función signo. Este simple derivador funciona bajo la premisa de que, aunque no se conoce la función $f(t)$ ni su derivada, se conoce que ésta última es acotada y que su valor siempre está entre los límites de $-k$ y k , y al momento de que z alcanza a $f(t)$, se produce una commutación de frecuencia infinita en la función signo, lo que le permite permanecer igual a la función original. El problema es que la derivada es una función discontinua que en realidad sólo toma tres valores, $-k$, k y cero. Así, se propone aumentar el orden del diferenciador, añadiendo una ecuación diferencial, para que quede de la siguiente manera:

$$\begin{cases} \dot{z}_0 = v, & v = -k_0|z_1 - f(t)|^{\frac{1}{2}}\text{sign}(z_1 - f(t)) + z_1 \\ \dot{z}_1 = -k_1\text{sign}(z_1 - v) \end{cases} \quad (3.20)$$

Donde k_0 y k_1 son constantes positivas. De esta manera, la última derivada, en este caso la segunda, es una función discontinua que al integrarse para formar parte de la primera derivada se convierte en una función continua. A ésta se le suma un segundo término que ayuda a corregir los errores de la estimación con \dot{z}_1 utilizando la misma idea y la función signo aunque ahora combinada con la potencia $1/2$ que permite tener una curva suave y continua para la primera derivada, que al integrarse da lugar a una función continua que converge en $f(t)$, la función original pero filtrada.

Este modelo puede extenderse a un derivador de orden arbitrario n de la siguiente forma:

$$\begin{cases} \dot{z}_0 = v_0, & v_0 = -k_0|z_0 - f(t)|^{n/(n+1)}\text{sign}(z_0 - f(t)) + z_1 \\ \dot{z}_1 = v_1, & v_1 = -k_1|z_1 - v_0|^{(n-1)/n}\text{sign}(z_1 - v_0) + z_2 \\ \vdots \\ \dot{z}_{n-1} = v_{n-1}, & v_{n-1} = -k_{n-1}|z_{n-1} - v_{n-2}|^{1/2}\text{sign}(z_{n-1} - v_{n-2}) + z_n \\ \dot{z}_n = -k_n\text{sign}(z_n - v_{n-1}) \end{cases} \quad (3.21)$$

Donde todas las constantes k_i son positivas, y de acuerdo con [20] se eligen de la siguiente manera: entre mayor sea su valor más rápido converge el derivador y más sensible es al ruido de la señal $f(t)$.

Fácilmente se puede observar que un derivador de orden, por ejemplo cuatro, contiene uno de orden tres en él, que además contiene uno de orden dos, y así sucesivamente. Esto

3.4 Derivador numérico

permite obtener no sólo la primer derivada numérica de una señal dada, sino además las n derivadas que se deseen, aunque debe notarse que, por ejemplo, un derivador de orden diez podría representar un fuerte trabajo computacional que probablemente no valga la pena si sólo se desea obtener la primera derivada de la señal.

4 Arquitectura del cuadricóptero

El dispositivo usado fue un cuadricóptero comercial llamado AR.Drone, desarrollado por la empresa Parrot.

El cuadricóptero está instrumentado para medir la orientación con un acelerómetro, un giroscopio y una brújula de tres ejes cada uno; y para medir la altitud con un sonar. Además cuenta con dos cámaras, una frontal y una vertical, de la que se sirve para controlar la posición en el modo *hover*. Cuenta con un sistema embebido con sistema operativo Linux sobre un procesador ARM y 128MB de memoria RAM. Tiene activo un servidor telnet, que al recibir una conexión da acceso a una consola para controlar el sistema operativo, por lo que es posible transmitirle archivos o crearlos en el mismo dispositivo. Además es posible compilar código para ARM en una computadora convencional (arquitectura x86) y ejecutarlo en el cuadricóptero. A este sistema embebido está conectada, a través del puerto serial, la tarjeta de navegación que recibe la información sin procesar de los sensores, y también el circuito encargado de controlar la velocidad de los motores a través de PWM. Por lo tanto es posible abrir una conexión serial a los puertos `/dev/ttyPA1`, donde se encuentra conectado el circuito encargado de los motores, y `/dev/ttyPA2`, donde está conectada la tarjeta de navegación.

Usualmente la manera de usar el AR.Drone es con un dispositivo móvil que tenga la aplicación desarrollada por el fabricante para poder manejarlo. De esta manera se usa el controlador PID interno del cuadricóptero y la interfaz gráfica sirve para indicarle al controlador los valores deseados en cuanto a altitud y postura. En la Figura 4.1 se puede observar este modelo de funcionamiento.

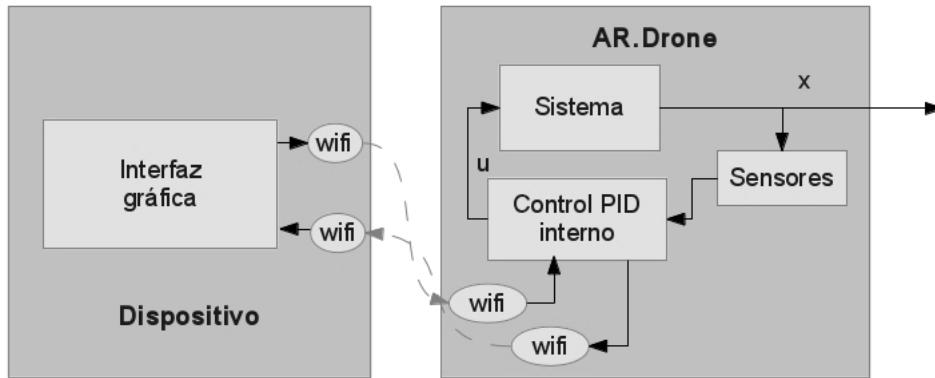


Figura 4.1 Modelo de funcionamiento de fábrica.

El cuadricóptero cuenta con un programa ubicado en `/bin/program.elf` que hace de servidor al que se conecta el dispositivo con la aplicación de manejo del AR.Drone. Este programa es el que contiene los controladores PID y el que se encarga de procesar la información de la tarjeta de navegación y utilizarla para los controladores y enviarla al dispositivo a través de la conexión WiFi. La aplicación en el dispositivo, es el lado cliente y no realiza ningún procesamiento en cuanto al vuelo del cuadricóptero, simplemente muestra la información obtenida desde el cuadricóptero y envía instrucciones que el controlador interpreta como ángulos o altitudes deseadas.

4.1. Comunicación WiFi con el AR.Drone

Para poder lograr comunicación entre el programa servidor en el AR.Drone y la computadora, que no cuenta con la aplicación cliente, se utilizó la librería libre *libardrone* que puede ser encontrada en [13]. Esta librería está hecha con Python y permite que con muy pocas líneas de código se pueda manejar el cuadricóptero. El sistema operativo mantiene cinco puertos de red abiertos. Uno es el puerto 21, usado para conexiones FTP, por el que uno se puede conectar para transferirle actualizaciones de firmware. Otro es el puerto 23, gestionado por un servidor Telnet y sirve para trabajar directamente con el sistema operativo. Los otros tres son manejados por el servidor (`/bin/program.elf`). El primero es el puerto 5554 UDP, por el que el cuadricóptero envía aproximadamente cada 30 milisegundos su información de navegación, es decir su postura, su estado, velocidades, altitud, estado de la batería, etc. Por el puerto 5555 UDP el cuadricóptero envía las imágenes capturadas

por sus cámaras, que vienen codificadas en un formato especificado en [14] y que la librería decodifica automáticamente. Finalmente, por el puerto 5556 UDP, recibe la información de los comandos enviados al cuadricóptero. Entre los comandos que recibe el AR.Drone se encuentra el despegar, aterrizar, parada de emergencia, o de movimiento hacia atrás, adelante, derecha o izquierda, controlando su inclinación hacia los lados y hacia adelante o hacia atrás, su velocidad vertical y su velocidad angular al rededor del eje vertical. Además se le pueden enviar comandos para manejar independientemente las velocidades de sus motores con una señal PWM, evitando así usar el control embebido que incluye. También se pueden enviar comandos para modificar las once ganancias del control PID interno.

El algoritmo 4.1 es un ejemplo simple que hace volar al AR.Drone y esperar unos segundos en el aire antes de aterrizar.

Algoritmo 4.1

```
import libardrone
from time import sleep
drone=libardrone.ARDrone()
drone.takeoff()
drone.hover()
sleep(5)
drone.land()
drone.halt()
```

La librería tiene una clase principal llamada *ARDrone*, y al momento de instanciar un objeto de ella se crea la conexión con el dispositivo, aunque para ello, la computadora debe estar conectada a la red inalámbrica que el dispositivo crea. Una vez creado el objeto *drone* se puede acceder al streaming de video, a la información de navegación, o se le pueden enviar las instrucciones, como *takeoff()*, método que lo hace encender los motores y despegar. El método *hover()* lo mantiene en el mismo punto espacialmente. Después *sleep(5)* espera cinco segundos para pasar a la siguiente línea de código, y finalmente *land()* le indica que debe descender y *halt()* cierra la conexión.

Para poder leer la información de los sensores, la librería crea una estructura de datos como parte del objeto instanciado de la clase *ARDrone*, en el ejemplo el objeto tiene el nombre de *drone*. La estructura de datos se llama *navdata* y contiene la última lectura de valores recibida desde el AR.Drone, y se accede a ella como se muestra en el algoritmo 4.2.

Algoritmo 4.2

```
import libardrone
drone = libardrone.ARDrone()
altitud = drone.navdata[0]['altitude']
theta = drone.navdata[0]['theta']
phi = drone.navdata[0]['phi']
psi = drone.navdata[0]['psi']
drone.halt()
print "La altitud del cuadricóptero es de %d unidades" % altitud
print "La inclinación de cabeceo es de %dº" % theta
print "La inclinación de ladeo es de %dº" % phi
print "La orientación de guiñado es de %dº" % psi
```

El ejemplo muestra cómo se guardan los valores de altitud y orientación del cuadricóptero en ese instante en sus respectivas variables y lo imprime en la pantalla. También, como se especifica en [14], se puede obtener información del estado de la batería, estado del cuadricóptero, entre otros.

El método *pwm()* envía la señal de PWM deseada a los motores y es la manera que se tiene de variar la velocidad de las hélices para evitar usar el controlador que tiene integrado el aparato.

El AR.Drone acepta una resolución de nueve bits en la señal de PWM, lo que significa que toma valores desde 0 hasta 511, por lo que si se desea que los motores trabajen, por ejemplo, a media carga, se envía un valor de 256 a cada uno de ellos con la siguiente línea de código: *drone.pwm(256,256,256,256)*.

4.2. Modelo 3D

Se decidió incorporar un modelo 3D de un cuadricóptero que permitiera observar en la pantalla el comportamiento del cuadricóptero real y que ilustrara de una manera más gráfica los resultados de las simulaciones. El modelo fue diseñado con Blender, programa libre de modelado utilizado para realizar películas y videojuegos principalmente, pues tiene un motor de simulación física [21]. El diseño no está basado en un modelo en particular de algún cuadricóptero, y simplemente ilustra la apariencia a rasgos generales de uno de estos aparatos.

En la Figura 4.2 se puede observar el modelo creado y la interfaz de Blender. Generalmente el formato de archivo que usa Blender es *.blend*, sin embargo puede exportar los modelos

a otros formatos que puedan ser usados por un programa diferente a Blender y uno de estos es el formato *.obj*. Este formato incluye una lista de todos los vértices que forman la figura con coordenadas en el espacio, es decir, un dato para la posición en el eje *x*, otro dato para la posición en el eje *y* y otro para la posición en el eje *z*. Después de la lista de vértices viene una lista de “caras” formadas por estos vértices, entre tres y cuatro de ellos. Éstas “caras” son polígonos en el espacio, que juntos forman la superficie del objeto en 3D. Para indicar un vértice en un documento de formato *.obj*, la línea comienza con la letra *v*, seguida por tres números decimales, todo separado por espacios. Para indicar una cara, la línea comienza con la letra *f*, seguida por tres o cuatro números enteros, y cada número corresponde a una coordenada en la lista de vértices, también la letra *f* y los números enteros se encuentran separados por un espacio. Así, es muy fácil diseñar un programa que pueda leer e interpretar este tipo de archivos, para después desplegarlos en una imagen o una animación.

Para este propósito se utilizó Python, un lenguaje libre orientado a objetos y OpenGL, una librería de computación gráfica ampliamente usada en CAD, realidad virtual o videojuegos, pues permite la creación de aplicaciones que produzcan gráficos en dos y tres dimensiones [22].

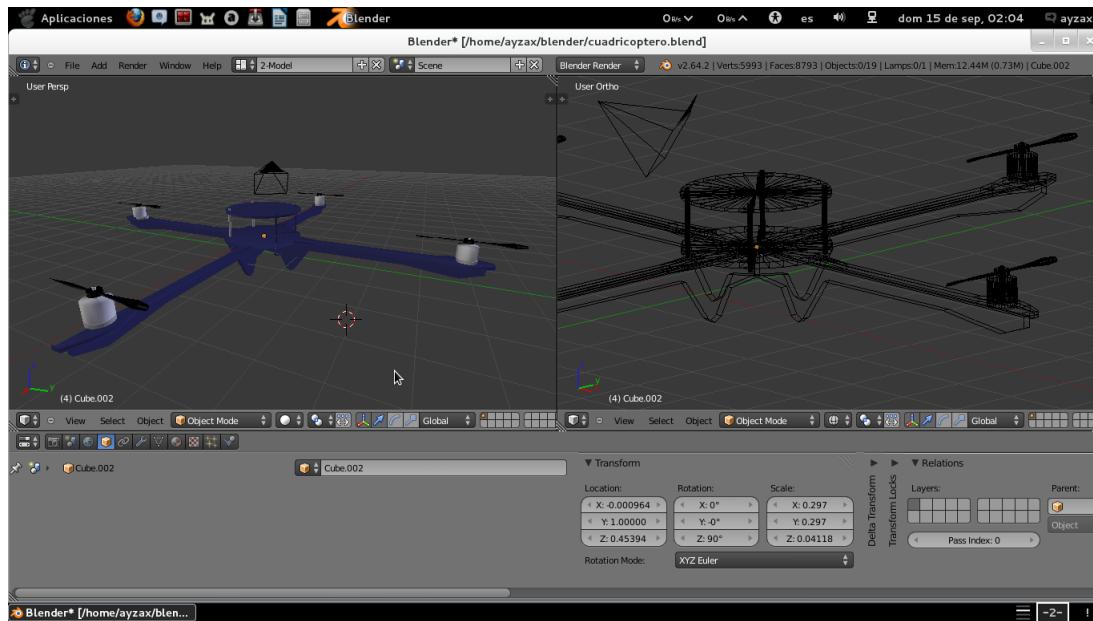


Figura 4.2 Interfaz gráfica de Blender.

4 Arquitectura del cuadricóptero

Tomando los datos generados por las simulaciones, es decir, orientación y posición espacial del cuadricóptero, es posible, mediante una serie de transformaciones, generar una animación donde se vea el movimiento del aparato en el tiempo. La primera de estas transformaciones es la de rotación. Utilizando la matriz de rotación en la ecuación 2.1, y multiplicándola por cada uno de los vértices, se obtiene la nueva posición de cada uno de ellos, a la que después se le agrega la traslación. Esto se puede ver de la siguiente manera:

$$v_n = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi \\ s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} x_T \\ y_T \\ z_T \end{bmatrix} \quad (4.1)$$

donde x, y, z son las coordenadas del vértice; x_T, y_T, z_T forman el vector de traslación y v_n indica el vértice nuevo. Esta ecuación se puede simplificar utilizando una matriz de transformación extendida de la siguiente forma:

$$v_n = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi & x_T \\ s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi & y_T \\ -s\theta & c\theta s\phi & c\theta c\phi & z_T \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (4.2)$$

Este proceso debe aplicarse con todos los vértices para cada cuadro de la animación, teniendo cuidado de guardar intacta una lista con los vértices originales, es decir, guardar dos listas, una con los vértices originales y otra con los transformados. Aunque el tener dos listas ocupa el doble de memoria, se ahorra tiempo de procesamiento, pues hay vértices que son usados por más de una cara, y el aplicarles la transformación al momento que son usados significaría hacerlo más de una vez para algunos de ellos. Este proceso de rotar y trasladar todos los vértices conlleva cierta cantidad de procesamiento, y dado que Python es un lenguaje interpretado y no compilado, esto puede ser notorio en el rendimiento del programa. Por esta razón es importante tener un modelo 3D de cierta manera simple, sin demasiados puntos para transformar, pues entre más puntos tenga la figura, más tiempo tarda en desplegarla y por lo tanto se puede perder la sensación de movimiento.

En cuanto al uso del modelo 3D junto con el modelo real, no era posible ubicar al modelo 3D en el espacio, pues las coordenadas geográficas no podían ser proporcionadas por el AR.Drone, ni medidas experimentalmente, a excepción de la altitud. Por lo demás, era posible ver el comportamiento del modelo físico repetido en el modelo virtual en tiempo real.

4.3. Interfaz gráfica



Figura 4.3 Interfaz gráfica.

Para el desarrollo de la interfaz gráfica se utilizó la librería PyGame, una librería creada principalmente para el desarrollo de videojuegos, por lo que incluye una gran variedad de funcionalidades para manejo de gráficos en pantalla, así como la interacción con los dispositivos de entrada como el teclado, mouse o un joystick [23].

La interfaz se desarrolló simulando la configuración del panel de un helicóptero, incluyendo marcadores para la altitud, la batería y la velocidad de los motores, además de una brújula y un indicador de horizonte. También, aprovechando las cámaras con que cuenta el AR.Drone se incluyó un recuadro donde se muestra la imagen de la cámara frontal, y en un pequeño

4 Arquitectura del cuadricóptero

recuadro la de la cámara inferior. En la Figura 4.3 se puede ver una captura de la ventana de la interfaz gráfica en uno de los experimentos de vuelo.

4.3.1. Indicador de altitud

Siendo que el cuadricóptero cuenta con un sonar en la parte inferior, es muy simple conocer su altitud, pues además el sistema embebido del cuadricóptero filtra la señal y entrega un valor consistente. Sin embargo fue necesario hacer una calibración del sensor para conocer el dato en una unidad inteligible. El sonar puede medir hasta seis metros de distancia, sin embargo la medida máxima en el indicador se decidió que fuera de cinco metros porque se ajustaba mejor a la carátula del indicador. Para la calibración se realizaron una serie de mediciones con una escala y su correspondiente valor entregado por el sensor.

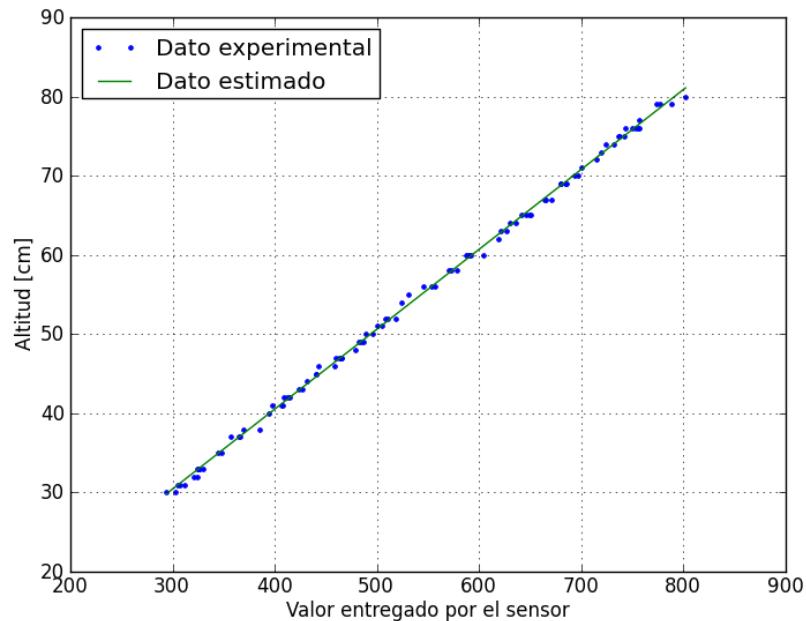


Figura 4.4 Calibración del sensor de altitud.

Después de tener datos suficientes se hizo una regresión lineal para obtener una ecuación de la forma $y = mx + c$, donde x es el valor entregado por el sensor y y es la distancia medida en centímetros. Los datos obtenidos son $m = 0.100894375451$ y $c = 0.172518077706$, con una correlación de $r = 0.999481$, que indica que la recta y explica correctamente la serie de puntos obtenidos experimentalmente. Los datos medidos y la recta obtenida pueden verse en la Figura 4.4.

La altitud se señala en el indicador con una línea que apunta el valor de la altitud en metros como se muestra en la Figura 4.5, sin embargo, el valor exacto puede ser leído en el display de mensajes.



Figura 4.5 Indicador de altitud.



Figura 4.6 Brújula.

4.3.2. Brújula

Para la brújula sólo se tuvo que comparar la posición del AR.Drone con una brújula real para saber cuál era la correspondencia entre el valor entregado por el sensor magnético, que es un valor entre cero y trescientos sesenta. Se obtuvo entonces que cero grados corresponden al Este, y noventa grados al Norte. Después se colocó una línea sobre el indicador que apunta al punto cardinal indicado por el sensor. En la Figura 4.6 se muestra el indicador.

4.3.3. Indicador de horizonte

El indicador de horizonte muestra los ángulos de ladeo y cabeceo en una misma carátula. Es un círculo dividido en dos, azul y café, de forma que parezca el cielo y la tierra. La línea de horizonte gira tanto como el sensor indique el ladeo del aparato, y al coincidir con las marcas se puede saber el ángulo. Además, dependiendo del cabeceo, la línea de horizonte sube o baja, y al coincidir con las marcas horizontales se conoce el ángulo de inclinación. Como ejemplo, en la Figura 4.7 se puede observar una inclinación de ladeo de 15° y 10° de cabeceo.



Figura 4.7 Indicador de horizonte.

4.3.4. Indicador de la velocidad de los motores

Para mostrar la velocidad de los motores se incluyó un display en la parte inferior izquierda que cuenta con una cuadrícula de cuatro columnas, cada una correspondiente a un motor, y la velocidad se muestra por medio de una barra que se completa conforme la velocidad es mayor y está dividido en porcentaje, siendo el máximo 100 % de la velocidad, correspondiente a un valor de 511 para la señal digital. El indicador de la velocidad de los motores sólo funciona cuando el controlador externo es utilizado ya que la librería libardrone no permite conocer la velocidad de los motores al usar el controlador que tiene de

fábrica.

4.3.5. Display de mensajes

En el mismo display donde se muestra la velocidad de los motores, que puede verse en la Figura 4.8, del lado izquierdo hay un espacio relativamente amplio para desplegar mensajes, como las instrucciones para manejar el AR.Drone, mensajes de error y el estado de la batería.

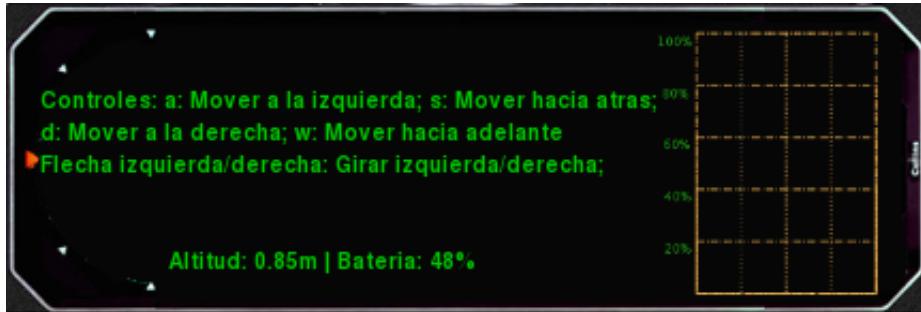


Figura 4.8 Display de mensajes.

5 Implementación, pruebas y resultados

Dado el modelo del cuadricóptero de la ecuación 2.4, y la ley de control propuesta en la ecuación 3.15, una vez simulado con MATLAB se implementó utilizando Python como lenguaje de programación, la librería descrita en el capítulo anterior como medio de comunicación con el AR.Drone y la librería numpy, desarrollada como una librería especializada en realizar cálculos numéricos de manera rápida en aplicaciones científicas, que como ejemplo, implementa una clase para utilizar matrices y realizar operaciones con ellas, además de que tiene un mejor rendimiento debido a que es un módulo compilado y no interpretado [25].

5.1. Implementación con Python

Como se describe en la Figura 5.1, el modelo de funcionamiento varía con respecto al propuesto por el fabricante en que la información de los sensores se le entrega a la interfaz gráfica via WiFi para ser procesada en el controlador propuesto en el capítulo 3, que después envía las señales de control al sistema, es decir, a los motores, via WiFi sin tomar en cuenta el controlador interno del AR.Drone.

5.1.1. Control de postura

El principal problema inicialmente fue que el tiempo que tardaba en llegar un dato después del anterior era aproximadamente de medio segundo, que es demasiado, por lo que se tuvo que editar la librería para eliminar la transmisión de video. De esta manera se mejoró el tiempo que transcurría entre cada dato a cerca de cincuenta milisegundos, que era

5 Implementación, pruebas y resultados

sustancialmente mejor que antes, sin embargo en ninguna de las pruebas de vuelo se logró estabilizar al cuadricóptero.

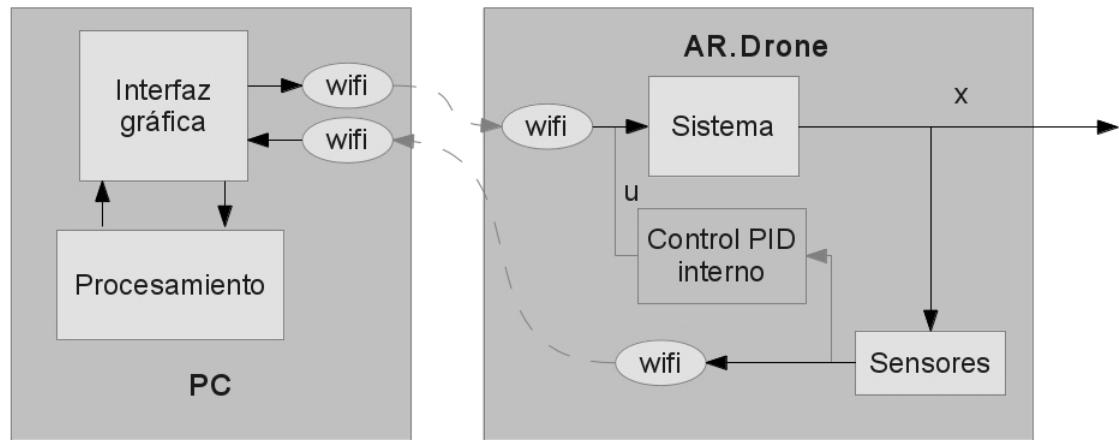


Figura 5.1 Modelo de funcionamiento propuesto.

Se añadió un derivador numérico que además filtrara la señal, pues la derivada calculada de la manera tradicional no proporcionaba un valor adecuado debido a la frecuencia con la que se recibía la información de los sensores y la forma de la señal de control mejoró, sin embargo, el desempeño del controlador no fue el deseado, pues el sistema entraba en oscilaciones cada vez más grandes, probablemente debido al retraso en el flujo de datos de los sensores al controlador y del controlador a los actuadores, además de la frecuencia de los datos. En la Figura 5.2 se puede ver el resultado de uno de los experimentos de vuelo y se puede notar el tiempo de retraso en las señales de los sensores en la forma un tanto cuadrada de la señal, además de las oscilaciones cada vez mayores.

Como prueba se decidió aumentar el valor de las ganancias pues en la simulación, al hacer esto se alcanzaba el valor deseado más rápidamente, sin embargo, como muestra la Figura 5.3, en el experimento esto sólo hizo las oscilaciones más frecuentes.

Debido a que no fue posible estabilizar al cuadricóptero, tampoco fue posible comprobar el funcionamiento del controlador de altitud.

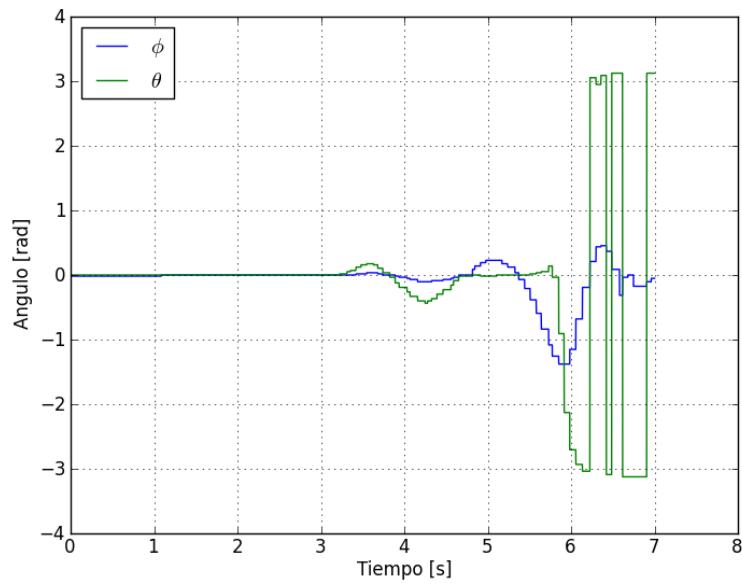


Figura 5.2 Ángulos en experimento de vuelo.

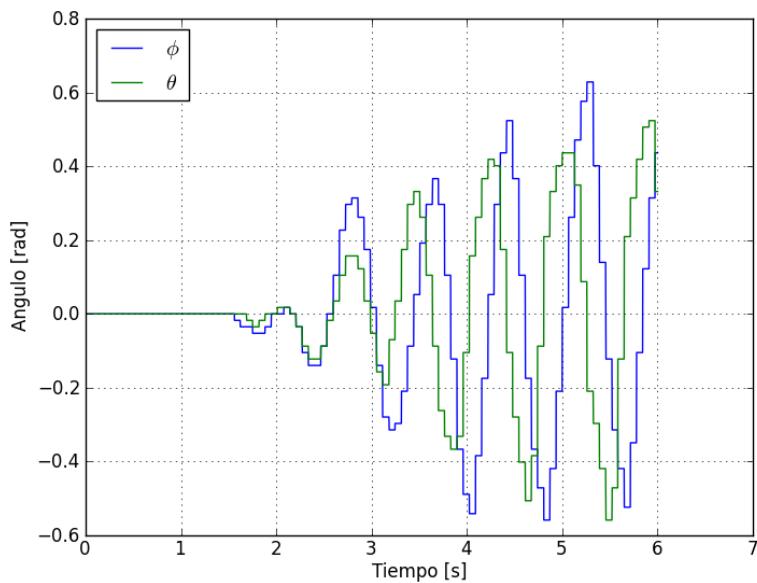


Figura 5.3 Ángulos en experimento de vuelo.

5.2. Implementación con C

Dado que con el primer modelo de funcionamiento no fue posible que el cuadricóptero volara de manera estable se decidió buscar una manera alternativa de implementar el controlador de modo que no hubiera este retraso entre la comunicación de la planta y el controlador. La manera más lógica para realizar esto era programar directamente el AR.Drone, pues tiene un sistema operativo capaz de ejecutar software hecho a la medida que fácilmente se le puede transferir, y tiene una forma de acceder a controlar el hardware, es decir, leer los sensores y manejar los motores, a través del puerto serial. Utilizando la librería encontrada en [24], se desarrolló un firmware alternativo para realizar los experimentos. La librería se encarga de procesar la información de la tarjeta de navegación con un filtro de Kalman y la deja disponible en una estructura de datos. Cuenta con una función para controlar la velocidad de los motores con PWM dándole como argumento un valor entre cero y uno que representa el ciclo de carga. Gracias a que la librería se encarga de decodificar y codificar los datos para comunicarse con las tarjetas de navegación y de control de los motores no es necesario trabajar directamente con los dispositivos a través del puerto serial.

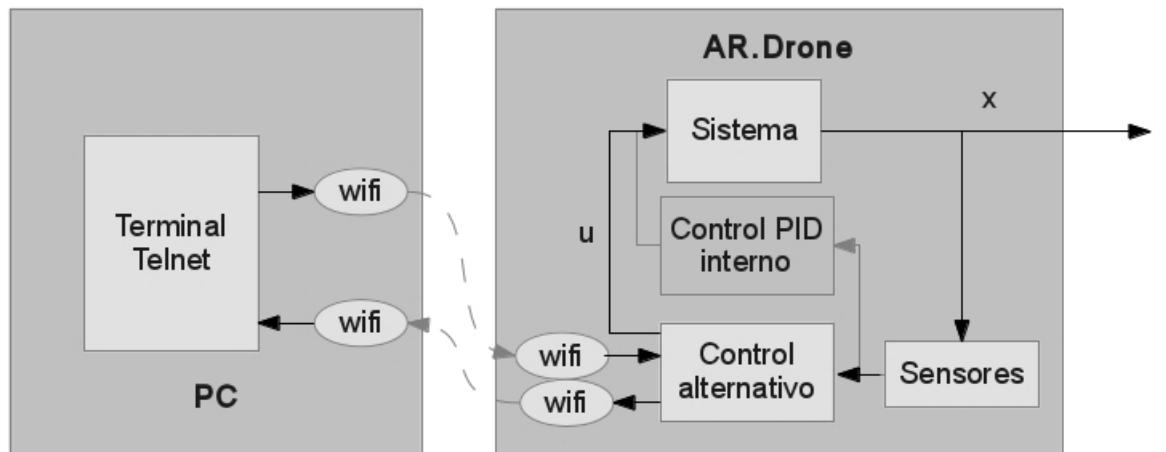


Figura 5.4 Modelo de funcionamiento propuesto.

Antes de ejecutar el firmware que se desarrolló es necesario detener el proceso que corresponde a `/bin/program.elf`, pues como este se encuentra usando los archivos `/dev/ttyPA1` y `/dev/ttyPA2` no es posible para otro proceso acceder a ellos. Esto se hace al *matar* el id de proceso que corresponde al programa, que suele ser el id 962, así que se ejecuta desde la

consola en telnet el comando `kill 962`, y entonces ya es posible ejecutar el firmware propio. Este modelo se representa en la Figura 5.4.

Para cuestiones de recolección de datos se decidió que el programa imprimiera en pantalla los datos que se quisieran analizar, como los valores de los ángulos, las velocidades angulares, la altitud, etc. y se redirigió la salida del programa a un archivo que después es recogido a través del servidor FTP para ser procesado, de esta manera no es necesario hacer más pesado el firmware añadiendo manejo de archivos.

5.2.1. Control de postura

Para desarrollar el firmware propio se *tradujeron* los programas hechos en Python a C, con sus debidas modificaciones, pues Python es un lenguaje orientado a objetos y C no. Inicialmente se hicieron pruebas sin el derivador numérico, utilizando la forma tradicional de derivar, es decir, la diferencia entre el valor anterior y el actual dividida entre la diferencia de tiempo, pues, a diferencia del programa en Python, se tenían datos continuos en cada ciclo, por lo que no debía de suceder el mismo problema que sucedió entonces. Sin embargo, como puede observarse en la Figura 5.5, a pesar de que la señal de los ángulos está filtrada y aparenta no tener ruido, su derivada, en la Figura 5.6, amplifica el poco ruido y resulta ser inutilizable.

Por esta razón se decidió implementar el derivador numérico presentado en la sección 3.4, que es capaz de obtener la derivada de la señal sin ser afectado por el ruido. Se realizaron simulaciones con una señal de prueba, comparando el desempeño de derivadores de diferente orden. Gracias a las simulaciones se decidió utilizar uno de quinto orden, debido a que computacionalmente no representaba una gran carga y tenía un desempeño notablemente mejor que los de orden inferior.

Retomando la forma del derivador que se describe en la ecuación 3.21, se propone un derivador de quinto orden que tenga la siguiente forma:

$$\left\{ \begin{array}{l} \dot{z}_0 = v_0, \\ \dot{z}_1 = v_1, \\ \dot{z}_2 = v_2, \\ \dot{z}_3 = v_3, \\ \dot{z}_4 = v_4, \\ \dot{z}_5 = -k_5 \text{sign}(z_5 - v_4) \end{array} \right. \quad \begin{array}{l} v_0 = -k_0 |z_0 - f(t)|^{5/6} \text{sign}(z_0 - f(t)) + z_1 \\ v_1 = -k_1 |z_1 - v_0|^{4/5} \text{sign}(z_1 - v_0) + z_2 \\ v_2 = -k_2 |z_2 - v_1|^{3/4} \text{sign}(z_2 - v_1) + z_3 \\ v_3 = -k_3 |z_3 - v_2|^{2/3} \text{sign}(z_3 - v_2) + z_4 \\ v_4 = -k_4 |z_4 - v_3|^{1/2} \text{sign}(z_4 - v_3) + z_5 \end{array} \quad (5.1)$$

5 Implementación, pruebas y resultados

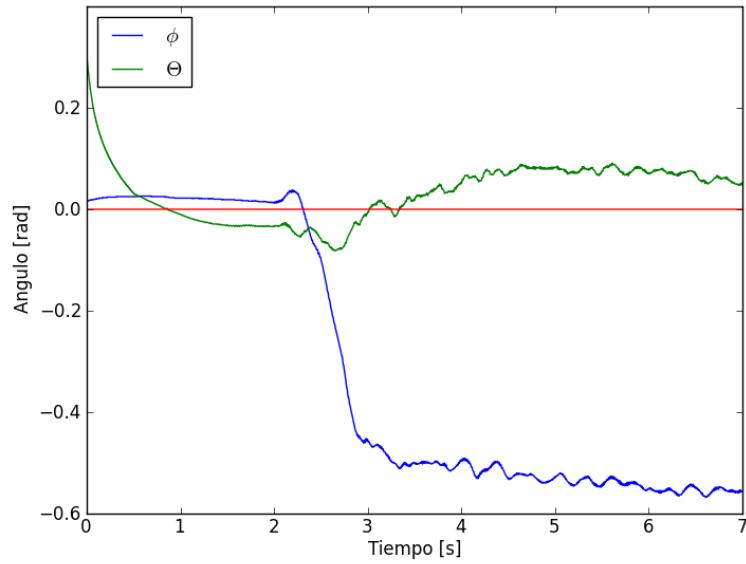


Figura 5.5 Experimento de medición de ángulos.

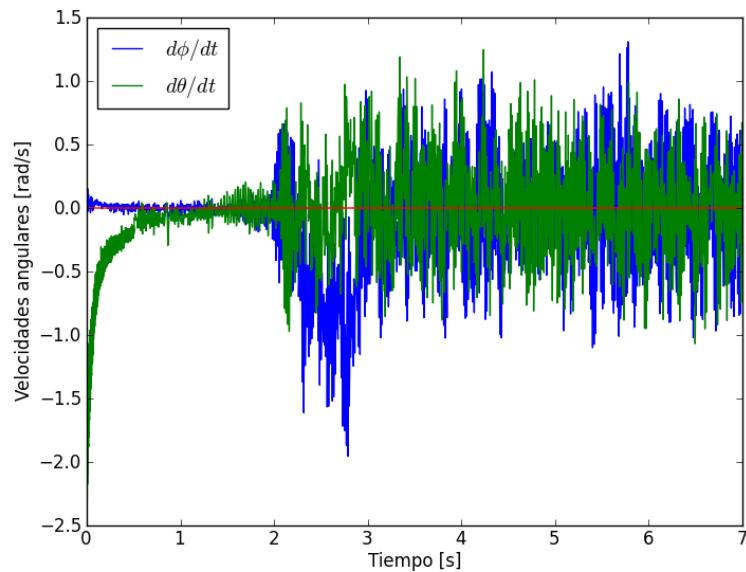


Figura 5.6 Derivada numérica de los ángulos medidaos.

Para darle un valor a las ganancias se utilizó la misma señal de prueba que para comparar los derivadores, la cual era una onda senoidal afectada por una cantidad de ruido de aproximadamente diez porciento de la amplitud de la señal. Recordando que de acuerdo con [20], entre mayor fuera el valor de las ganancias mayor sería la sensibilidad al ruido, se eligieron los valores $k_0 = 15$, $k_1 = 12$, $k_2 = 8$, $k_3 = 4$, $k_4 = 2.2$ y $k_5 = 1.1$, que hacían un buen trabajo con la señal de prueba escogida. Al implementar el derivador numérico en el cuadricóptero con estas ganancias se observaron curvas suaves para los ángulos y velocidades angulares, sin embargo, en las pruebas de vuelo no lograba estabilizarse, pues aunque se observaba un comportamiento aparentemente mejor, terminaba entrando en oscilaciones cada vez más grandes.

Al comparar en las gráficas los valores leídos y calculados para los ángulos y velocidades angulares respectivamente, con los valores filtrados, como puede verse en las figuras 5.7 y 5.8, se observó que aunque las velocidades angulares obtenidas mediante el derivador tenían la forma de las velocidades angulares calculadas al dividir el valor actual menos el anterior entre la diferencia de tiempo, las filtradas estaban atrasadas aproximadamente 140 milisegundos, que al parecer, para la dinámica del cuadricóptero es demasiado.

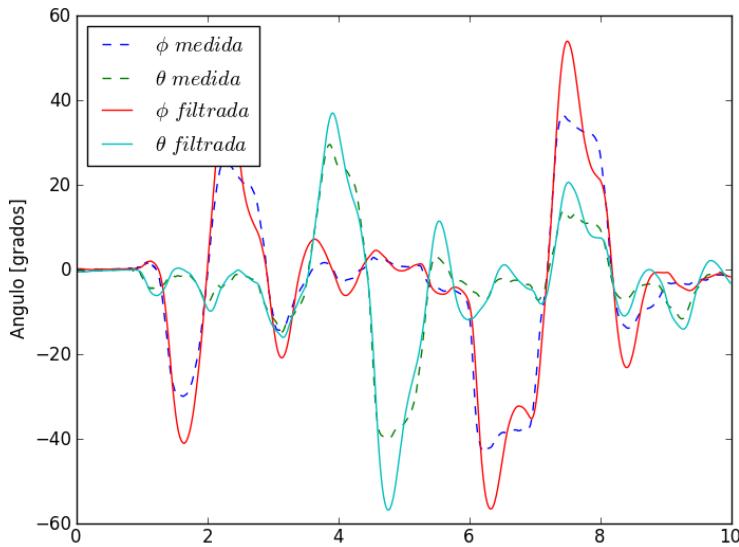


Figura 5.7 Señal de ángulos filtrada.

5 Implementación, pruebas y resultados

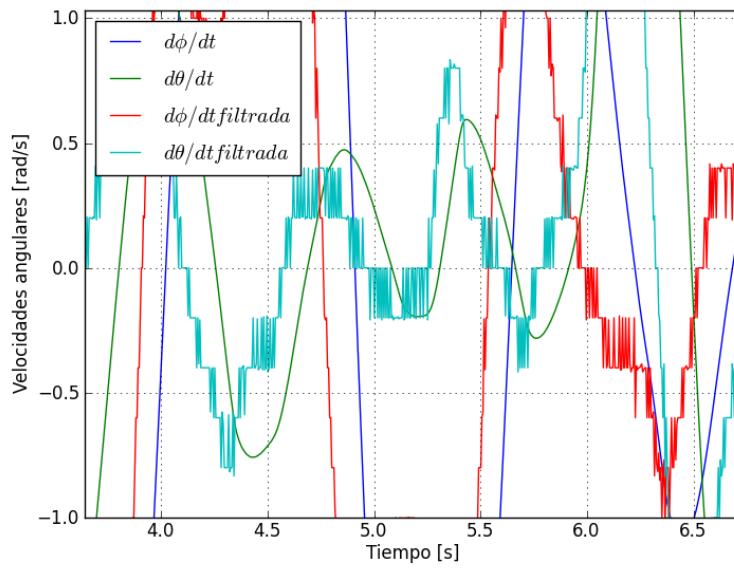


Figura 5.8 Velocidad angular filtrada

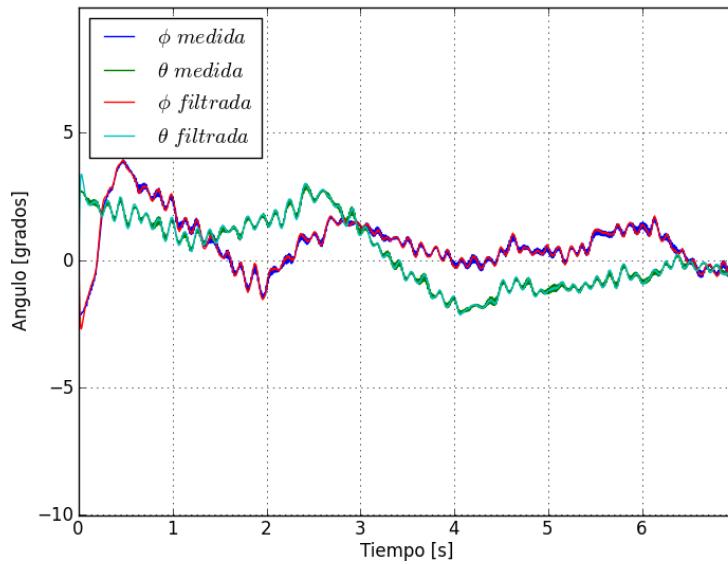


Figura 5.9 Prueba de vuelo. Señal de ángulos filtrada.

Después de ciertas pruebas con una señal de prueba modificada y con el cuadricóptero mismo, se reajustaron los valores de las ganancias en $k_0 = 50$, $k_1 = 30$, $k_2 = 15$, $k_3 = 7$, $k_4 = 5$ y $k_5 = 2.5$, y en la primera prueba de vuelo, que duró cuatro segundos, se pudo estabilizar el cuadricóptero y se mantuvieron los ángulos a un máximo de 3.5° , lo que le permitió volar totalmente estable durante la prueba, donde la señal filtrada por el derivador numérico se encontró durante todo momento sobre la señal medida, además de una señal limpia y sin retrasos para la velocidad angular. En la Figura 5.9 se muestran los ángulos ϑ y ϕ durante una prueba de vuelo posterior.

5.2.2. Control de altitud

Para pruebas de vuelo de mayor duración inicialmente se tuvo un problema con el control de altitud, que es el mismo controlador PID que se desarrolló para las simulaciones iniciales. Se notó en uno de los experimentos que el dato de altitud entregado por el sensor no era consistente con el comportamiento que se veía y se decidió calibrar el sensor, a pesar de que la librería indica que el dato entregado está dado en centímetros. En los resultados obtenidos del experimento para la calibración se notó que el sensor no es capaz de medir distancias menores a aproximadamente 20cm, y para todas estas distancias otorgaba el mismo valor. Con el resto de valores se realizó una regresión lineal para convertir el dato entregado por el sensor en centímetros. Resultó un valor de correlación $r = 0.999180$, que indica que la recta obtenida explica correctamente los puntos generados en el experimento. La pendiente y la ordenada al origen de la recta resultaron tener un valor de $m = 0.825208333607$ y $c = -0.459954219352$, respectivamente. La recta correspondiente a esta pendiente y ordenada al origen, además de los datos del experimento, pueden verse en la Figura 5.10.

Después de tener el sensor correctamente calibrado se decidió reajustar la ley de control del subsistema correspondiente a la altitud, que como puede verse en la ecuación 3.6, contiene el término g , que es la aceleración de la gravedad. Al añadir el término mg a la ley de control u_1 lo que se desea es cancelar el efecto de la gravedad que debe ser contrarrestado por el controlador PID, y de esta manera, el controlador sólo debe de encargarse de controlar la dinámica de una masa sin fricción y el cambio probó mejorar el desempeño del controlador en la simulación y en las pruebas de vuelo.

En la Figura 5.11 se puede ver una gráfica de la altitud del cuadricóptero durante una prueba de vuelo. La altura a la que debe llegar son treinta centímetros. Debido a la saturación del sensor en distancias pequeñas, la línea de altitud comienza cerca de los 23 cm, cosa que además de nublar un poco la visualización del comportamiento durante el vuelo,

5 Implementación, pruebas y resultados

provoca que el controlador trabaje más lentamente, pues hasta donde *él* sabe, el error es 23 cm menor a lo que en realidad es.

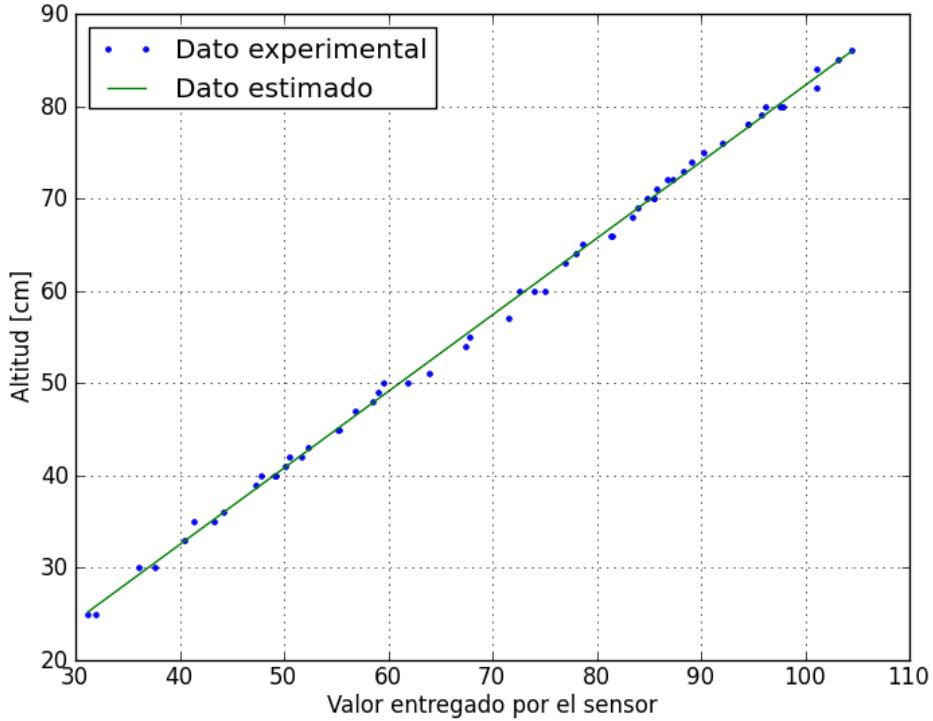


Figura 5.10 Calibración del sensor de altitud.

Para ayudar un poco a la visualización se agregó una línea discontinua roja que va del punto donde inicia el cuadricóptero en la prueba de vuelo, es decir, 0 cm en 0 segundos, al punto donde se obtienen por primera vez datos diferentes con el sensor, que es aproximadamente a los dos segundos de iniciada la prueba; esto con la suposición de que el recorrido durante ese tiempo fue una línea recta, lo cual probablemente no es cierto, sin embargo, es la suposición más simple posible. En la gráfica se puede notar que el cuadricóptero llega a la altura deseada poco después de los cinco segundos y se mantiene cerca durante el resto de la prueba de vuelo. Se observan también una pequeña caída cerca de los tres segundos, que posiblemente es provocada por cómo funciona el controlador, igual que se vio en las simulaciones con MATLAB. La prueba de vuelo de la que se grafica la altitud en la Figura 5.11 es de la que se mostraron los ángulos en la Figura 5.9.

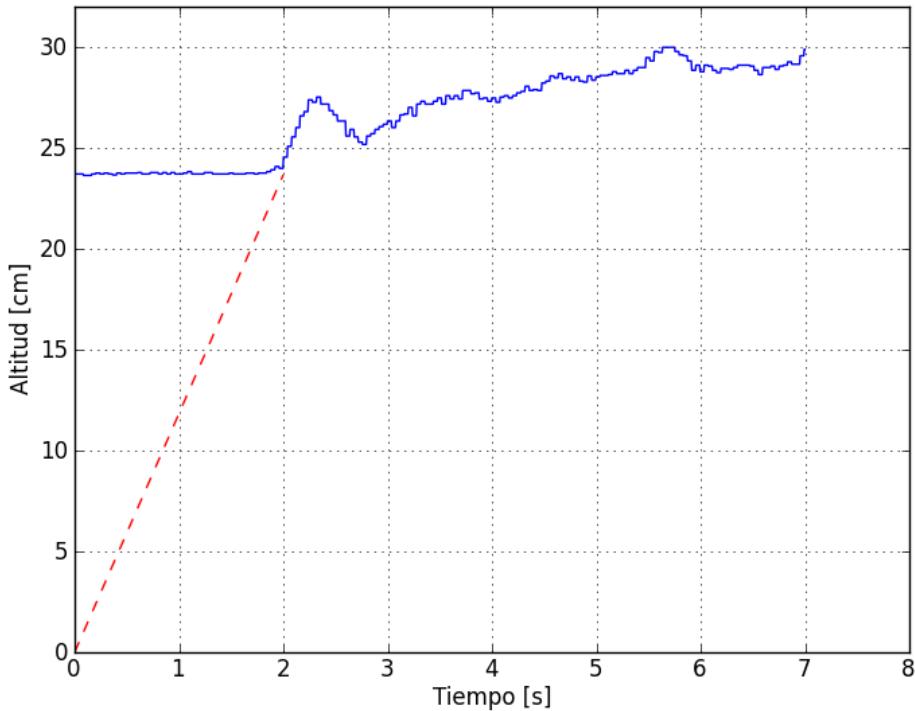


Figura 5.11 Altitud en prueba de vuelo .

5.2.3. Comparación con el controlador de fábrica

Para comparar el desempeño de ambos controladores, el implementado y el de fábrica, se hicieron dos pruebas agregando perturbaciones al sistema. La primera de las pruebas era que el cuadricóptero despegara y poco después, ya en vuelo, se agregaba un pequeño peso colgado de una de las orillas del cuadricóptero para desestabilizarlo y ver cómo reaccionaba el controlador ante dicha perturbación. En la Figura 5.12 se muestran las gráficas para los ángulos y la altitud del experimento realizado utilizando el controlador por modos deslizantes que se implementó.

5 Implementación, pruebas y resultados

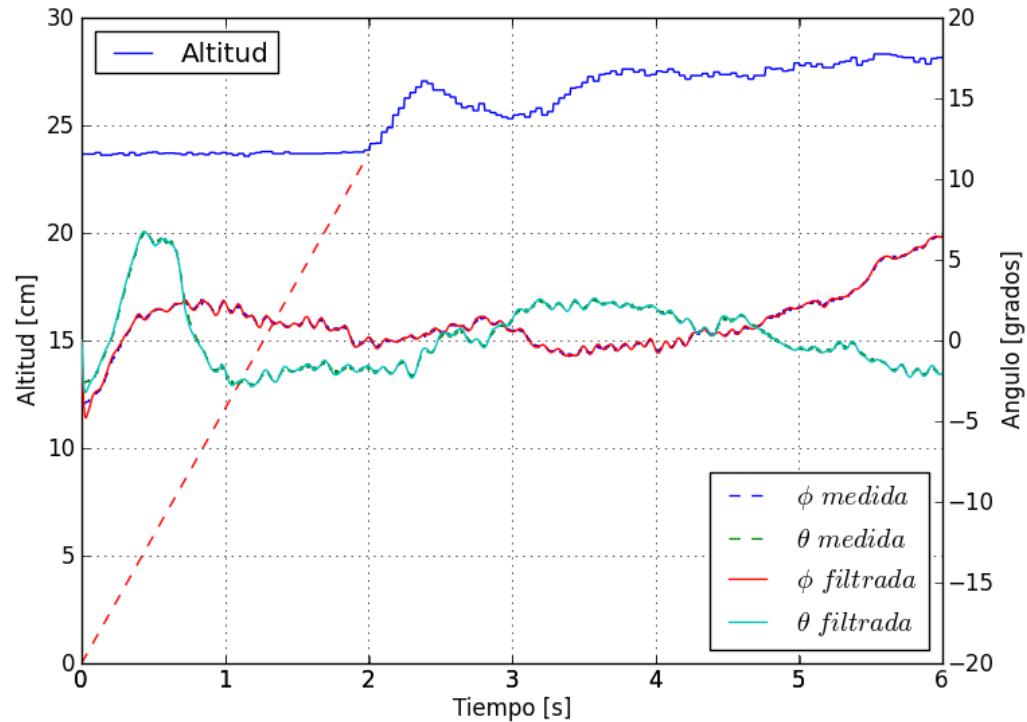


Figura 5.12 Ángulos y altitud en prueba de vuelo con perturbación. Control Supertwisting.

Al observar la figura no se aprecia un gran cambio, comparadas con la prueba de vuelo en las figuras 5.9 y 5.11, donde no hubo perturbaciones, a excepción del final de la prueba, donde uno de los ángulos se eleva a más de cinco grados, sin embargo, no es un valor demasiado elevado y puede ser contrarrestado. En la prueba de vuelo la diferencia fue que durante lo que duró la prueba no se alcanzó la altura deseada, sin embargo el cuadricóptero se mantuvo elevado durante toda la prueba.

Con la intención de tener una figura para comparar el vuelo con y sin perturbaciones durante el uso del controlador de fábrica, se incluye la Figura 5.13, y en la Figura 5.14 se muestra el desempeño del controlador que el cuadricóptero tiene de fábrica ante la primera perturbación propuesta.

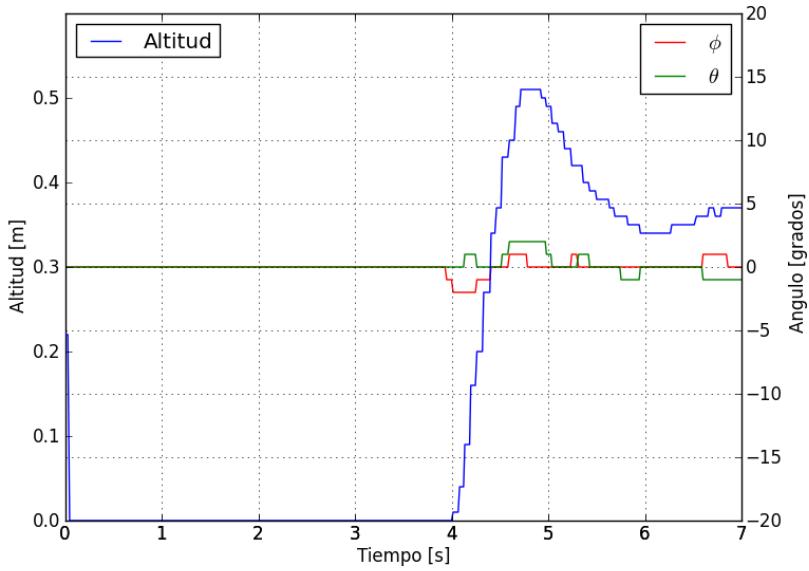


Figura 5.13 Gráfica del comportamiento del controlador de fábrica sin perturbaciones.

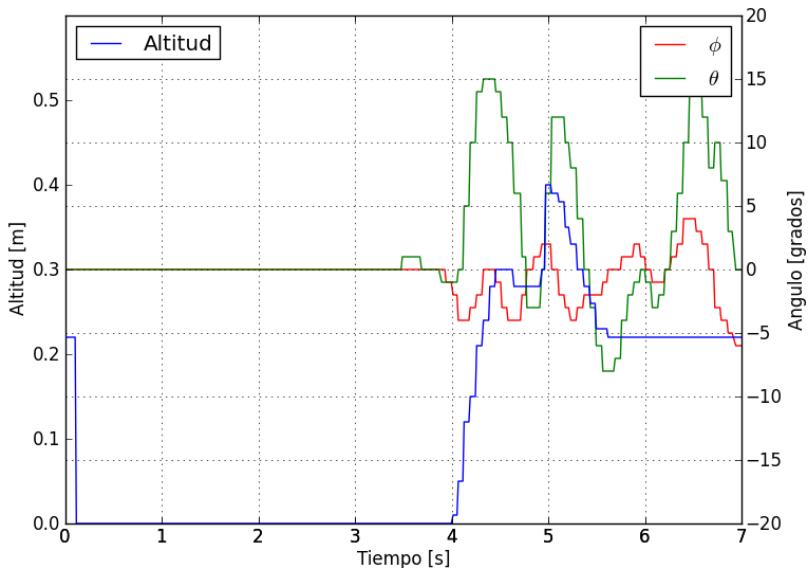


Figura 5.14 Ángulos y altitud en prueba de vuelo con perturbación. Control de fábrica.

5 Implementación, pruebas y resultados

Para la segunda prueba, se colocó el mismo peso extra en una de las orillas del cuadricóptero, sin embargo, para esta prueba el cuadricóptero debía despegar cargando el peso desde el principio de la prueba. En la Figura 5.15 se muestra la prueba de vuelo con el controlador por modos deslizantes. Se puede ver que al igual que con la prueba anterior, no hay una gran diferencia entre el vuelo con el peso o sin él, a excepción de los ángulos al final de la prueba, sin embargo, eso fue causado por un choque que lo desestabilizó. También se nota que no llega a la altitud deseada, aunque se mantuvo elevado durante toda la prueba.

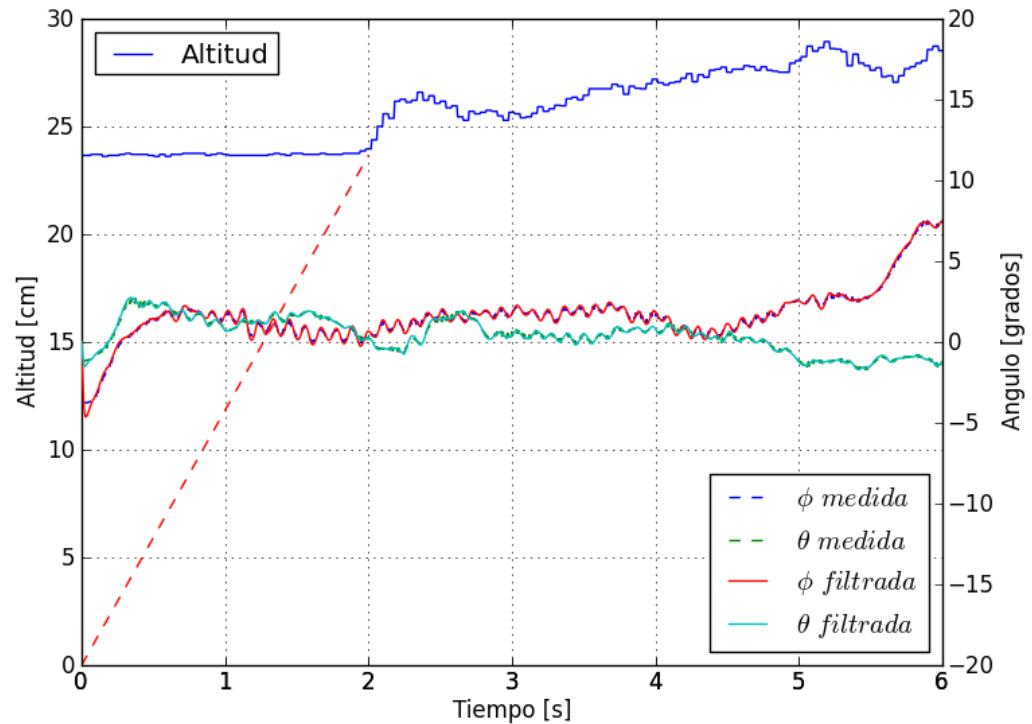


Figura 5.15 Ángulos y altitud en prueba de vuelo con perturbación. Control Supertwisting.

El comportamiento del cuadricóptero con el controlador de fábrica fue similar en ambas pruebas, como muestra la Figura 5.16. La desestabilización en los ángulos fue menor, sin embargo, en ambos casos el cuadricóptero no se mantuvo elevado. Aunque en la gráfica del primer experimento en la Figura 5.14 no se aprecia una curva completa para la altitud, posiblemente debido a errores con el sensor, tal vez debido a la inclinación del cuadricóptero debida al peso, es posible notar que llegó a una altura poco mayor a 40cm, lo cual es

considerablemente mayor a lo que se logró en la segunda prueba debido a que en la primera, el peso no colgaba del cuadricóptero desde el principio, pero a partir de que el cuadricóptero tuvo que soportar el peso comenzó a descender hasta llegar al suelo, donde se comportaba como si siguiera volando, pero sin despegarse del suelo. En la segunda prueba, al tener que despegar con el peso, de inmediato descendió, elevándose unos pocos centímetros durante aproximadamente un segundo y medio, como se ve en la figura.

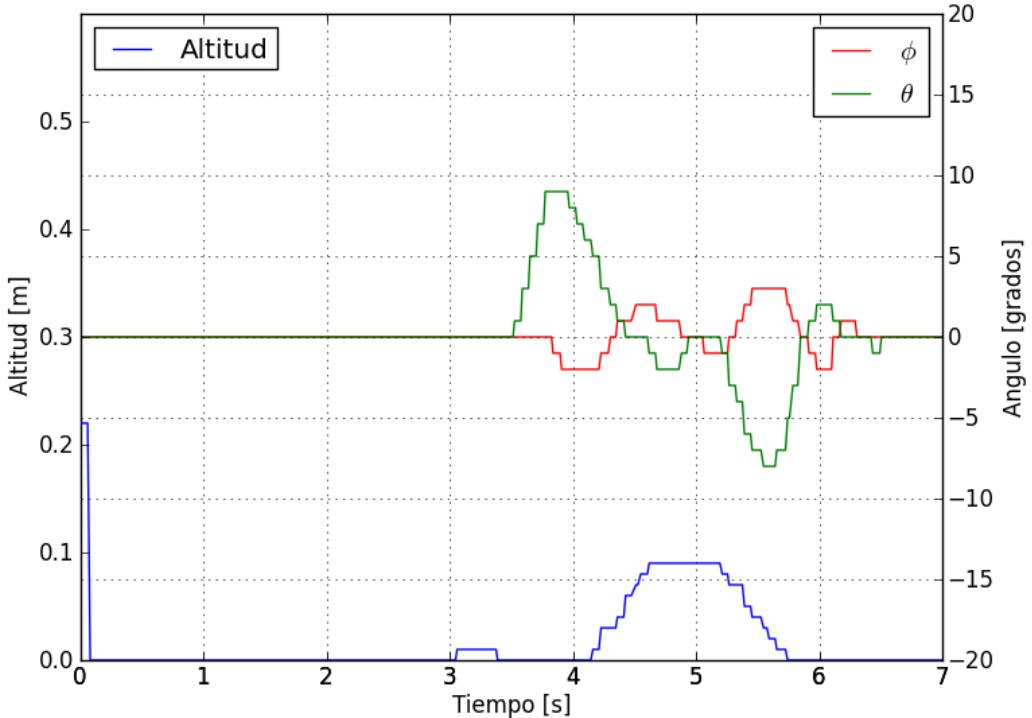


Figura 5.16 Ángulos y altitud en prueba de vuelo con perturbación. Control de fábrica.

El problema en estas pruebas es que, al no tener un control de posición, el cuadricóptero alcanza una velocidad horizontal debido a la inclinación que ocasiona en el aparato el peso añadido, y aunque los ángulos son después estabilizados, no se contrarresta esa velocidad. El problema con esto es que durante el vuelo se llega a los límites del laboratorio donde se realizan las pruebas, y al chocar contra algún objeto se desestabiliza el vuelo. Por esta razón se hicieron pruebas de vuelo un poco más cortas.

6 Conclusiones

Durante los experimentos se pudo comprobar la altamente inestable dinámica de un cuadricóptero, por lo que el diseño de un controlador capaz de estabilizarlo a pesar de las condiciones externas y las incertidumbres paramétricas es indispensable, especialmente en tareas más exigentes que el simple vuelo por entretenimiento, pues un cuadricóptero es un dispositivo que ofrece una gran variedad de aplicaciones, desde el monitoreo de ambientes inaccesibles para el hombre, a la grabación aérea para una película, pasando por una cantidad diversa de la que la imaginación es el límite.

También es necesario elegir un esquema de funcionamiento que permita al controlador un desempeño adecuado, pues a pesar de las ventajas ofrecidas por el controlador elegido en este trabajo, no fue posible lograr el desempeño deseado durante la primer etapa de la implementación. Esto fue debido probablemente a la baja frecuencia de recepción de datos, que no estaba muy alejada de la frecuencia con la que, según el fabricante, el sistema los entregaba, y con la que no fue posible lograr la estabilización. Probablemente también debido al retraso generado por el esquema de funcionamiento, pues como pudo verse en experimentos posteriores, donde la señal de la velocidad angular tenía un retraso considerable, fue imposible la estabilización mientras no se corrigió el problema.

El segundo esquema de funcionamiento propuesto fue capaz de estabilizar el sistema ya que se contaba con una acción de control prácticamente en tiempo real, que cada aproximadamente un milisegundo se actualizaba con datos también de tiempo real para corregir los errores y permitir un vuelo estable.

Probablemente el mejor modelo de funcionamiento sea el propuesto por el fabricante, en

6 Conclusiones

el que el controlador es capaz de trabajar en tiempo real, con datos también en tiempo real, y que se comunica con un segundo dispositivo con mayores capacidades de procesamiento, encargado de dar instrucciones al controlador para que, a pesar del retraso que pueda haber debido a la comunicación, el cuadricóptero se encuentre siempre en vuelo estable. Así el segundo dispositivo puede ser capaz de usar los datos, por ejemplo de las cámaras del cuadricóptero, para realizar un vuelo autónomo, siguiendo una ruta, o seguir un objeto en movimiento.

De las pruebas de vuelo realizadas para comparar el desempeño de los dos controladores, el diseñado por el fabricante y el propuesto, se puede decir que, al comparar las gráficas de vuelo sin perturbaciones con las gráficas en donde hay perturbaciones, el controlador propuesto en esta tesis aparenta ser menos afectado por el peso añadido al cuadricóptero, siendo que ambos, el control de postura y altitud lograron estabilizarlo y elevarlo a pesar del peso añadido, mientras que el control de postura de fábrica se vió más afectado llegando a ángulos mayores incluso a 15° , el control de altitud no logró mantenerlo en vuelo. Esto puede ser debido a medidas de seguridad en el firmware con las que se saturen las señales de control una vez en vuelo estable, sin embargo, al ser éste un programa del que no se conoce el código sólo es posible especular al respecto.

6.1. Trabajo futuro

Aunque se obtuvieron resultados satisfactorios en la implementación del controlador externo por modos deslizantes en el cuadricóptero AR.Drone, no es posible aún pensar en el controlador propuesto como un reemplazo para el controlador de fábrica, pues aún hay aspectos que se deben mejorar o completar. Como líneas de trabajo próximas a seguir a partir de los resultados obtenidos en esta tesis se sugieren las siguientes:

- aprovechar el hecho de que la librería utilizada hace un cálculo del movimiento horizontal del cuadricóptero utilizando la cámara vertical;
- implementar un sistema de cámaras externo para determinar la posición espacial del cuadricóptero en interiores;
- implementación de controlador de posición espacial que alimente al controlador de postura;
- realizar seguimiento de trayectoria para los ángulos de inclinación del cuadricóptero, con el objetivo de que el aparato después pueda seguir trayectorias en el espacio;

- análisis de las imágenes de las cámaras a bordo para determinar la posición relativa a un objeto, con la intención de seguir a dicho objeto.

6.2. Código

La versión digital de esta tesis en formato PDF, así como mis datos de contacto y los programas utilizados en los experimentos, tanto los desarrollados en Python como los desarrollados en C, pueden ser descargados desde el repositorio creado en mi cuenta de GitHub® en la dirección <http://www.github.com/albertoibm/Thesis>. El pseudocódigo del algoritmo supertwisting, del derivador numérico y del controlador PID se incluyen en el apéndice.

A Pseudocódigo

A.1. Algoritmo Supertwisting

```
t0 = time()           // Define tiempo inicial
while time() - t0 < totaltime:
    // Durante el tiempo que dure la prueba
    dt = time() - ta      // Diferencial de tiempo
    ta = time()            // Tiempo anterior
    alt = get_altitude()   // Obtiene lecturas
    th = get_theta()
    ph = get_phi()
    ps = get_psi()
    x = [th, ph, ps]       // Vector de estados
    xp = dif(x,dt)         // Derivar
    // Calcula s
    error = xd - x
    errorp = xpd - xp
    s = errorp + lambda * error
    // Calcula u del control de postura
    integral.update(s, dt)
    u = ginv * (-lambda * errorp
                 - K1 * sqrt(abs(s)) * sign(s))
```

A Pseudocódigo

```

        - K2 * integral.value() )
// Calcula u del control de altitud
u4 = max(0, control_alt.output(altd - alt, dt)) // Controlador PID
// Calcula velocidades de los motores
o1 = sqrt(-(b * u[2] + d * u[0] - d * u4 + d * u[1])
/ (4 * b * d) )
o2 = sqrt((-d * u[0] + d * u[1] + b * u[2] + d * u4)
/ (4 * b * d) )
o3 = sqrt(-(-d * u[1] + b * u[2] - d * u4 - d * u[0])
/ (4 * b * d) )
o4 = sqrt(d * u4 + b * u[2] + d * u[0] - d * u[1])
/ (4 * b * d) )
pwm(o1, o2, o3, o4)

```

A.2. Derivador numérico

```

class DiffOrdN:
    function DiffOrdN(n, gains):
        ord = n
        diffS = []
        for i in range(n + 1):
            diffS.append(Diff(i, gains[n-i]))
    function update(val, dt):
        zm1 = 0
        for i in range(ord, -1, -1):
            if i == 0:
                zm1 = 0
            else:
                zm1 = diffS[i-1].output()
                diffS[i].transfer(val, zm1)
                val = diffS[i].value()
                diffS[i].integrate(val, dt)
    function output():
        outs = []
        for i in range(ord + 1):

```

```

        outs.append( self.diffs[i].output
                      () )
    return outs

class Diff:
    function Diff(n,gain):
        ord = n
        integral = 0
        tmp = 0
        preintegral = 0
    function transfer(input,zm1):
        preintegral = -sign(integral - input) * gain *
                      power(abs(integral - input), pow_n(ord)) + zm1
    function value():
        return preintegral
    function integrate(val, dt):
        integral += (val + tmp) * dt / 2
        tmp = val
    function output():
        return integral
function pow_n(n):
    return n/(n+1)

```

A.3. Controlador PID

```

class PID:
    function PID(kp, ki, kd):
        I = 0
        ea = 0
    function output(e, dt):
        I += (e - ea) / 2 * dt
        I += dt * ea
        D = (e - ea) / dt
        ea = e
        return kp * e + ki * I + kd * D

```


Bibliografía

- [1] J.M. McMichael y M.S. Francis. Micro air vehicles-toward a new dimension in flight. DARPA Document, 1997.
- [2] Samir Bouabdallah, Design and control of quadrotors with application to autonomous flying.
- [3] Samir Bouabdallah et al., Full Control of a Quadrotor, Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, 153-158.
- [4] Stanislaw H. Zak, Systems and Control.
- [5] Holger Voos, Nonlinear State-Dependent Riccati Equation Control of a Quadrotor UAV, Proceedings of the 2006 IEEE International Conference on Control Applications , 2547-2552.
- [6] Holger Voos, Nonlinear and Neural Network-based Control of a Small Four-Rotor Aerial Robot.
- [7] H. Rafaralahy et al., Simultaneous observer based sensor diagnosis and speed estimation of Unmanned Aerial Vehicle, Proceedings of the 47th IEEE Conference on Decision and Control, 2938-2943.
- [8] Mehmet Önder Efe, Neural Network Assisted Computationally Simple $PI^\lambda D^\mu$ Control of a Quadrotor UAV, IEEE Transactions on Industrial Informatics, Vol. 7, no. 2, May 2011.

Bibliografía

- [9] Frank Hoffmann et al., Attitude estimation and control of a quadrocopter, The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1072-1077.
- [10] L. Derafa, A. Benallegue y L. Fridman, Super twisting control algorithm for the attitude tracking of a four rotors UAV, Journal of the Franklin Institute, doi:10.1016/j.jfranklin.2011.10.011
- [11] A. Benallegue, A. Mokhtari y L. Fridman, High-order sliding-mode observer for a quadrotor UAV, International Journal of Robust and Nonlinear Control.
- [12] Markus Achtelik et al., Visual Tracking and Control of a Quadcopter Using a Stereo Camera System and Inertial Sensors.
- [13] <https://github.com/venthur/python-ardrone> revisado el 15 de septiembre de 2013
- [14] Stephane Piskorski Nicolas Brulez Pierre Eline , AR.Drone Developer Guide
- [15] Arnoud Visser et al., Closing the gap between simulation and reality in the sensor and motion models of an autonomous AR.Drone
- [16] <http://www.mathworks.com/products/matlab/> revisado el 22 de octubre de 2013
- [17] Jean-Jacques E. Slotine y Weiping Li, Applied Nonlinear Control. .
- [18] Otto Föllinger, Nichtlineare Regelungen I.
- [19] Leonid Fridman, Jaime Moreno, y Rafael Iriarte (Eds.), Sliding Modes after the First Decade of the 21st Century.
- [20] Arie Levant (2003): Higher order sliding modes, differentiation and output-feedback control, International Journal of Control, 76:9-10, 924-941
- [21] <http://www.blender.org/about/> revisado el 22 de octubre de 2013
- [22] <http://www.opengl.org/about/> revisado el 22 de octubre de 2013
- [23] <http://www.pygame.org/wiki/about> revisado el 22 de octubre de 2013
- [24] <https://github.com/ardrone/ardrone> revisado el 22 de octubre de 2013
- [25] <http://www.numpy.org/> revisado el 22 de octubre de 2013