

Facultad de Ingeniería de la
Universidad de la República

Proyecto de fin de carrera de Ingeniería Eléctrica

Vuelo autónomo de un cuadricóptero

Autores:
Joaquín Berrutti
Lucas Falkenstein
Federico Favaro

Tutor:
Ing. Rafael Canetti

julio, 2015

Agradecimientos

Agradecemos principalmente a nuestros familiares y amigos por el constante apoyo brindado a lo largo de este proyecto y los integrantes del proyecto antecesor uQuad!. También agradecemos a Mauricio Gonzalez y Leonardo Steinfeld quienes aportaron su conocimiento y experiencia en diversas áreas involucradas.

Finalmente agradecemos a Gabriel Falkenstein por su ayuda en el diseño de algunas partes físicas y al personal de CLUBUR Fútbol 5 por permitirnos utilizar su cancha durante algunas pruebas con el GPS.

Resumen

El objetivo principal del presente proyecto es la automatización del vuelo de un UAV (Unmanned Aerial Vehicle) con arquitectura de cuadricóptero.

Por un lado se describe el material seleccionado para el armado físico del cuadricóptero y de la electrónica utilizada. Esto incluye el frame, motores, hélices, varios sensores y placa de desarrollo entre otras cosas. Se destaca además la utilización de una plataforma open hardware/open source para controlar su actitud. A su vez se implementan algoritmos de generación y seguimiento de trayectorias a partir de la especificación de determinados waypoints dados por el usuario. Finalmente se diseñan una serie de controladores que permiten, en conjunto con los algoritmos antes mencionados, controlar el vuelo de forma de cumplir los objetivos impuestos.

Abstract

The main goal of this project is the automation of the flight of a UAV (Unmanned Aerial Vehicle) with quadcopter architecture.

On one hand, the materials selected for the physical assembly of the quadcopter and the electronic used is described. This includes the frame, engines, propellers, sensors and development board among other things. The use of a platform open hardware/open source to control the attitude is underlined. In addition, path planning and tracking algorithms are implemented through the specification of certain waypoints given by the user. Finally, the design of several controllers is needed in order to complete the automation of the flight.

Tabla de contenidos

Capítulo 1

Introducción

Los vehículos aéreos no tripulados, UAV por sus siglas en inglés, tienen un gran número de aplicaciones. Los primeros desarrollos se dieron en el área militar, donde se utilizaban plataformas del tipo *Fixed wings*, avionetas controladas remotamente por un usuario. Los avances tecnológicos de los últimos años permitieron llevar a bordo grandes capacidades de procesamiento y mecanismos de sensado, ya que se redujo considerablemente las dimensiones de los circuitos integrados. Esto derivó en una revolución en el campo de los UAVs, que desembocó en el cuadricóptero comercial que se conoce hoy en día. Estos vehículos se conocen popularmente como *drones* y son relativamente accesibles al público en general.

Los cuadricópteros son UAVs extremadamente maniobrables gracias a su gran inestabilidad. Pero al tratarse de sistemas inestables, dependen de una computadora y sensores a bordo, los cuales logran la estabilidad del vehículo accionando sobre la velocidad de cada uno de los rotores de forma independiente. Los avances tecnológicos también dieron lugar al desarrollo de sistemas de autonomía de vuelo, aumentando aún mas las prestaciones. Se pueden automatizar labores tales como fotografía, vigilancia, mediciones y distribución de materiales entre otras aplicaciones.

Al día de hoy existe una gran variedad de cuadricópteros comerciales, como lo son el *DJI Phantom*[1] o el *A.R.Drone*[2]. Sin embargo, existen también varios proyectos *open source*, por ejemplo [6], [7], que permiten el armado y desarrollo de cuadricópteros personalizados.

El objetivo de este proyecto de fin de carrera es el diseño y construcción de un cuadricóptero autónomo capaz de realizar un recorrido por una trayectoria definida a través de una serie de puntos del espacio. Implica el diseño, la adquisición de partes y armado físico del vehículo, el diseño de controladores y el desarrollo de algoritmos de generación y seguimiento de trayectorias.

[1] http://www.dji.com/product/phantom
[2] http://ardrone2.parrot.com/

Como opción de diseño, se optó por adquirir una plataforma *open source* del proyecto *OpenPilot* [8] que solucione el control de actitud, de forma de centrar el enfoque del proyecto en el control de vuelo y la generación y seguimiento de trayectorias.

Antecedentes

El cuadricóptero se ha constituido en una plataforma contemporánea para la formación e investigación en las universidades. Esto se debe al amplio rango de aplicaciones que el mismo puede ser capaz de abarcar, además de presentar una serie de problemas muy importantes a resolver para los cuales es necesario proponer soluciones tecnológicas que pueden llegar a ser innovadoras.

Existe en el mundo entero una gran cantidad de trabajos al respecto que contribuyen al crecimiento del estado del arte. Tal es el caso de Davide Scaramuzza et al. [5] y de Jason N. Gross et al. [12], quienes utilizan técnicas innovadoras para posicionar al vehículo en el espacio con mayor precisión, uno de los mayores desafíos existentes.

En particular, dentro del Instituto de Ingeniería Eléctrica de la *Udelar* existe una línea de trabajo sobre robótica móvil, en el marco de la cual nacieron diversos proyectos sobre UAVs. El primero de estos fue AMFO1-Bobby [2], donde se diseñó y construyó un avión a escala para ser volado mediante un control remoto y con ciertas capacidades de vuelo autónomo. Como continuación del primero surge el proyecto AUION [1], que se centró en resolver el diseño del controlador del avión en situación de vuelo. Siguiendo la línea de los primeros dos surgió el proyecto *uQuad!* [16], donde se diseñó e integró un sistema de control a una plataforma comercial con arquitectura de cuadricóptero.

El presente proyecto continúa la línea de trabajo mencionada. Tomando como punto de partida la plataforma desarrollada por *uQuad!*, se busca dotar a un cuadricóptero con capacidad de vuelo autónomo, de forma que el mismo sea capaz de cumplir con determinado objetivo de vuelo sin la necesidad de control por parte del usuario.

Objetivos específicos

Dentro del objetivo general de dotar de capacidad de vuelo autónomo al cuadricóptero, se definen los siguientes objetivos específicos:

- Incrementar la capacidad sensorial agregando sensores de proximidad, tanto para mejorar la estimación de altura como para dotar al cuadricóptero de la habilidad de detectar obstáculos.

- Implementar la generación de trayectorias en base a una serie de puntos del espacio, de ahora en más *waypoints*, ingresados por el usuario. Dichos *waypoints* deberán contener especificado además el ángulo de llegada. Entonces, la trayectoria generada deberá pasar de forma eficiente por todos los *waypoints* con el ángulo especificado.

- Generar un sistema de control que permita el seguimiento de las trayectorias generadas.

Con el fin de garantizar el cumplimento de los objetivos se plantean ciertos criterios a cumplir:

- Dentro de la medición de altura y la detección de obstáculos se establecen las siguientes metas:

  - Correcta detección de obstáculos a distancias menores a 5m.
  - Error máximo de 15cm en la medida de altura para distancias menores a 5m respecto del suelo (durante el despegue y aterrizaje).

- Respecto del seguimiento de trayectorias autogeneradas se establecen los siguientes criterios de éxito:

  - Error máximo en posicionamiento horizontal de 6m (en exteriores).
  - Error máximo en posicionamiento vertical de 70cm.

Organización del documento

El documento está organizado en partes independientes, de las cuales se presenta una breve descripción para facilitar su lectura.

- Descripción de la solución adoptada: El capítulo 2 presenta un panorama general del proyecto, describiendo la solución a los desafíos planteados. En el capítulo 3 se describe el hardware utilizado y sus criterios de elección.

- Caracterización del sistema: Los capítulos de esta parte abarcan el estudio experimental de los componentes utilizados permitiendo su caracterización.

- Algoritmos de autonomía: Desarrollo, simulaciones y pruebas de los algoritmos de generación y seguimiento de trayectorias que permiten la autonomía de vuelo.

- Controladores: En esta parte se describe el diseño de los distintos controladores necesarios.
- Implementación: Aquí se explican todos los detalles relevantes a la implementación, tanto del hardware como del software.

Parte I

Descripción de la solución adoptada

Capítulo 2

Descripción del sistema

Este capítulo presenta una descripción general de todos los aspectos involucrados en la solución adoptada para el desarrollo de un cuadricóptero autónomo. El lector podrá obtener un panorama general de lo desarrollado en este Proyecto de Fin de Carrera antes de entrar en detalles teóricos, técnicos y experimentales que se encuentran desarrollados en los siguientes capítulos.

2.1. Sistema físico

Un cuadricóptero consiste básicamente en una estructura rígida en forma de cruz con cuatro rotores fijados a los extremos de cada brazo. Como se puede observar en la figura 2.1, cada rotor posee un una hélice acoplada, que al girar (con velocidad ω_i) impone una fuerza (f_i) y un momentos (T_{M_i}), influyendo directamente en la dinámica del sistema. Los sistemas de referencia de interés son el inercial $S_I = \{x, y, z\}$ y el sistema solidario al cuadricóptero $S_{sol} = \{x_B, y_B, z_B\}$ donde x_B e y_B son colineales con las bisectrices de los brazos y z_B es ortogonal a los anteriores. La magnitud de las fuerzas y los momentos están relacionadas directamente con las velocidades de giro, por lo tanto, variando éstas en los cuatro motores de forma independiente se logra realizar las acciones de control. Cabe destacar que dos rotores giran en sentido horario mientras que los otros dos lo hacen en sentido antihorario, de forma de anular el momento neto ejercido en el eje z_B.

La suma de las cuatro fuerzas f_i se denomina empuje o *thrust* (Th) y es la encargada de vencer la fuerza peso, permitiendo el vuelo del vehículo. Dependiendo si la fuerza (Th) es mayor, menor o igual al peso, el cuadricóptero se moverá hacia arriba, hacia abajo o permanecerá suspendido en el aire (*hovering*) respectivamente.

Para describir los movimientos de rotación del cuadricóptero se introduce el concepto de ángulos de *Euler*, nombrados en la jerga aeroespacial como

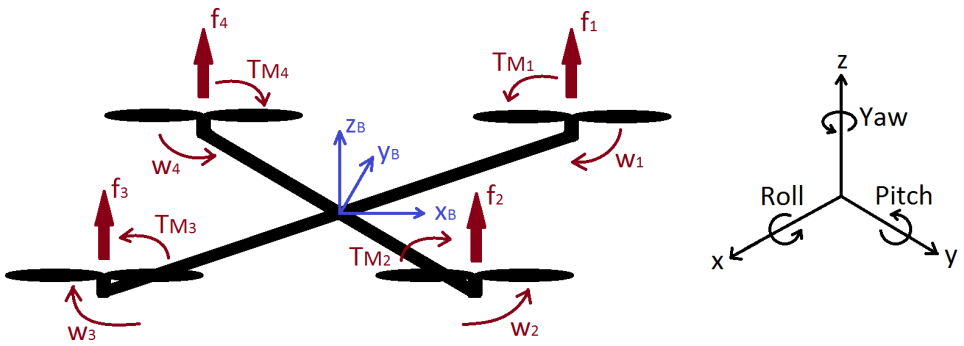


Figura 2.1: Sistema de coordenadas y modelo físico simplificado.

Yaw, *Pitch* y *Roll* (ver figura 2.3). El *Yaw* describe una rotación respecto al eje z, el *Pitch* una rotación respecto al eje y_B y el *Roll* una rotación respecto al eje x_B. En la figura 2.1 se muestran dichos ángulos representados en el cuadricóptero.

Lograr rotaciones implica crear desbalances entre las velocidades de los motores. A continuación se detallan las distintas relaciones que se deben imponer entre las velocidades de los motores para lograr movimientos en los ángulos de *Euler* (referirse a la figura 2.1):

- Para lograr movimiento en *Pitch* se deben aumentar (o disminuir) en conjunto las velocidades angulares de los motores 3 y 4 respecto de los motores 1 y 2. Esto es variar: $(\omega_1 + \omega_2) - (\omega_3 + \omega_4)$.

- Para lograr movimiento en *Roll* se deben aumentar (o disminuir) en conjunto las velocidades angulares de los motores 2 y 3 respecto de los motores 1 y 4. Esto es variar: $(\omega_2 + \omega_3) - (\omega_1 + \omega_4)$.

- Para el caso del ángulo *Yaw*, el giro se logra creando un desbalance entre los momentos (T_{M_i}) producidos por los motores. Dichos momentos se deben principalmente a los efectos aerodinámicos implicados en el giro de las hélices, actúan en sentido contrario a las mismas y su magnitud se puede considerar proporcional al cuadrado de la velocidad de giro [4] (ω_i). Esto se traduce en variar: $(\omega_1^2 + \omega_3^2) - (\omega_2^2 + \omega_4^2)$

2.2. Diagrama del sistema implementado

El sistema desarrollado para permitir el vuelo autónomo del cuadricóptero consta de dos grandes bloques. Por un lado se soluciona el control de la actitud y por otro se abarca la autonomía de vuelo, es decir, la generación y seguimiento de trayectorias. En la figura 2.2 se aprecia el diagrama general

del sistema eléctrico. El control de actitud, es decir de *Pitch*, *Roll* y *Yaw*, se realiza en el bloque *CC3D* y la autonomía de vuelo radica en *Beagleboard*.

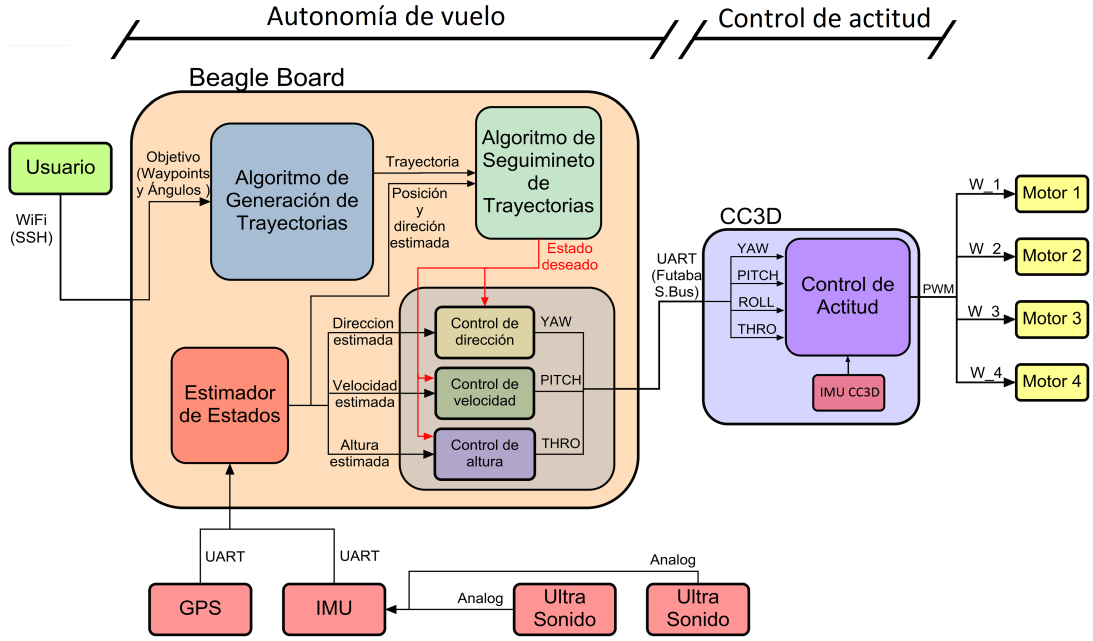


Figura 2.2: Diagrama del sistema eléctrico.

2.3. Control de actitud

El control de actitud para cuadricópteros es un área extensamente desarrollada por la comunidad. A los efectos de centrar el proyecto en el desarrollo de la autonomía de vuelo, se hace uso de una placa controladora desarrollada por el proyecto *Open Pilot* [8].

Como se aprecia en el diagrama, el control de actitud de cuadricóptero está a cargo de la *CC3D*. Esta recibe como entradas los ángulos de *Euler* a seguir y a partir de la estimación de los mismos ejecuta lazos de control del tipo PID. Dicho sistema actúa variando independientemente la velocidad de los cuatro motores. El sistema de control de actitud recibe un cuarto parámetro, denominado *Throtle*, señal a partir de la cual se maneja la suma de las velocidades de los cuatro rotores, modificando el empuje (Th).

2.4. Autonomía de vuelo

Dotar de autonomía al cuadricóptero combina una serie de algoritmos y procesos en cuyo desarrollo e implementación está centrado el foco de este

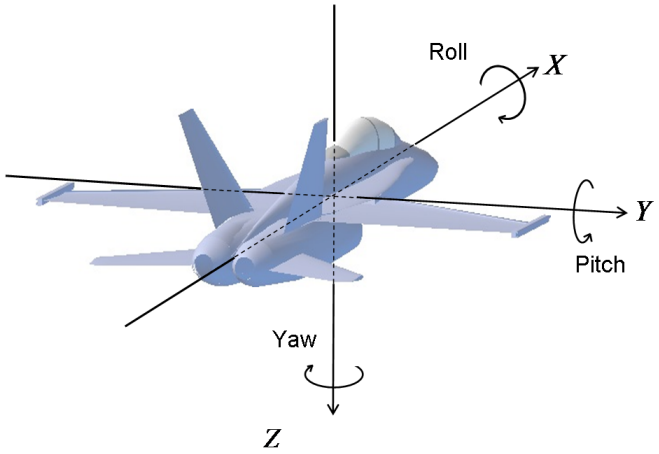


Figura 2.3: Ángulos de *Euler* denominados *Yaw*, *Pitch* y *Roll*.

Proyecto de Fin de Carrera. La totalidad de los procesos implicados corren en una única placa y forman parte de un programa principal común, manejando como periféricos a la placa de control de actitud y el hardware sensorial, tal como se muestra en el diagrama.

2.4.1. Estrategia de vuelo

La estrategia de vuelo utilizada en condiciones normales (sin obstáculos presentes), es la siguiente:

- Mantener una velocidad horizontal de módulo constante a lo largo del vuelo (lo que se traduce en un *Pitch* aproximadamente constante en régimen).
- El control de seguimiento de trayectorias se realiza actuando únicamente sobre el ángulo *Yaw*.
- El ángulo de *Roll* se mantiene nulo durante el vuelo, salvo en la maniobra evasiva.
- Cuando la trayectoria implica variaciones en altura, estas se recorren a velocidad de módulo constante, tal que la distancia vertical deseada sea recorrida en el mismo tiempo que la horizontal.

La opción de implementar el seguimiento de trayectorias utilizando solamente el ángulo *Yaw* tiene como objetivo buscar una solución sencilla al problema de seguimiento.

2.4.2. Generación de trayectorias

La generación de trayectorias se realiza antes de iniciar la misión y se repite únicamente en situaciones puntuales en las cuales es necesario un replanteo del camino a seguir (desvíos importantes o presencia de obstáculos). Se genera la trayectoria a seguir por el vehículo cumpliendo con las siguientes consignas:

- En el recorrido se deben atravesar todos los *waypoints* ingresados por el usuario.
- Cada *waypoint* debe ser alcanzado con la dirección indicada.
- La trayectoria debe ser realizable por parte del vehículo. No debe exigirle al sistema físico movimientos que éste no pueda realizar.
- Cumpliendo con los puntos anteriores, se debe optimizar la trayectoria minimizando la distancia a ser recorrida.

2.4.3. Seguimiento de trayectorias

El algoritmo de seguimiento se divide en dos partes, consiguiendo por un lado el seguimiento de la proyección horizontal de la trayectoria y por otro el seguimiento de altura.

El algoritmo de seguimiento horizontal decide, a partir de la estimación de posición, la dirección que debe tomar el vehículo para seguir la trayectoria. Como resultado se genera la señal Yaw_d (dirección deseada) a seguir por el controlador de actitud.

Por otro lado, el seguimiento vertical de trayectorias se realiza de forma tal que la posición vertical de cada *waypoint* se alcance aproximadamente al mismo tiempo que la posición vertical.

2.4.4. Control de *Yaw*

La placa controladora de actitud *CC3D* controla la derivada del *Yaw* y no directamente el ángulo. Por lo tanto, es necesario implementar un controlador que logre el ángulo de *Yaw* deseado calculado por el algoritmo de seguimiento.

2.4.5. Control de velocidad

El recorrido se realiza a una velocidad constante determinada previamente por el usuario. Se desarrolla un controlador de tipo PID para lograr la velocidad deseada a lo largo del trayecto. Éste se realimenta de la estimación de la velocidad del cuadricóptero y acciona mediante la variación del ángulo

Pitch ya que la dinámica según el eje x_B depende directamente del valor dicho ángulo.

2.4.6. Control de altitud

Al igual que en el control de velocidad, se implementa un controlador PID para el seguimiento de la altura de vuelo. La acción de control consiste en variar la suma de las velocidades de los cuatro motores, modificando el empuje neto y por ende la aceleración vertical del cuadricóptero.

2.4.7. Evasión de obstáculos

La evasión de obstáculos es un algoritmo de vuelo que se ejecuta en casos excepcionales, cuando se detecta la presencia de un obstáculo en la trayectoria a seguir.

Capítulo 3

Hardware eléctrico y mecánico

Una de las tareas fundamentales en el proyecto es el armado físico del vehículo a partir de sus componentes básicos. La elección del hardware debe realizarse con cautela, errores en éste punto pueden impactar fuertemente en las diferentes etapas futuras del proyecto. Se trata de un proceso que combina investigación teórica, del estado del arte y de factores comerciales.

Los componentes que forman parte de un cuadricóptero pueden clasificados en dos grandes grupos. Por un lado están los elementos mecánicos y eléctricos tales como la estructura, propulsores y sistema de energía. Por otro lado está la inteligencia, es decir, los sensores y las plataformas con microcontroladores donde corren los procesos y algoritmos encargados de dotar de estabilidad y autonomía al vehículo.

3.1. Componentes básicos del cuadricóptero

El diagrama de la figura 3.1 muestra los componentes eléctricos y electrónicos que junto con la estructura mecánica conforman la solución adoptada para el cuadricóptero. Las siguientes secciones describen cada componente por separado.

3.1.1. Estructura

Comúnmente llamado *frame*, la estructura da forma, rigidez y hace de soporte a todo el resto de los componentes. El material más adecuado para fabricar este tipo de estructuras es la fibra de carbono, ya que posee propiedades mecánicas similares al acero y es tan ligera como la madera o el plástico.

Sus dimensiones deben corresponderse con la carga a transportar y la potencia de los motores. Cuanto más grande, más carga extra (*payload*) será posible transportar.

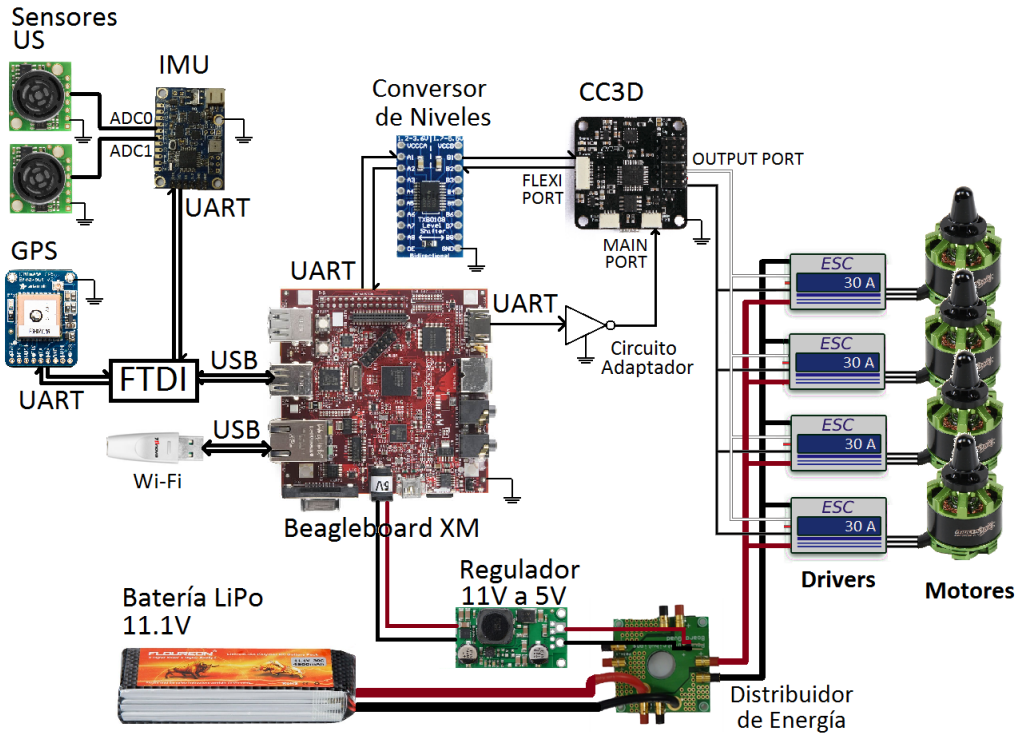


Figura 3.1: Arquitectura eléctrica de la solución implementada.

Otro factor a tener en cuenta son los espacios disponibles para la instalación del resto de los componentes. Resulta ventajosa la disponibilidad de varios niveles de bandejas y la posibilidad de agregado o modificación de la disposición de las mismas.

Elección

Se escoge el *frame* marca *Tarot* modelo *Iron man 650* (ver figura 3.2). Es robusto, liviano y presenta tres niveles de bandejas para la colocación de hardware. Sus principales características son:

Figura 3.2: *Tarot Iron man 650.*

- Material: Fibra de Carbono
- Peso: 450g
- Dimensiones: 65cm de diámetro, 20cm de altura.

Su diámetro permite instalar motores de potencia suficiente para transportar la carga extra del orden del kilogramo, cumpliendo los objetivos planteados.

3.1.2. Sistema de propulsión: motores, controladores y hélices

El sistema de propulsión está compuesto por motores, ESC (Electronic Speed Controller) y hélices. La elección de estos componentes debe ser en conjunto asegurando su compatibilidad.

Los motores eléctricos sin escobillas con rotor externo (Brushless Outrunner DC Motors) son comúnmente utilizados en vehículos radio-controlados debido a su eficiencia, potencia, durabilidad y bajo peso. Constan de un rotor de imán permanente y tres pares de bobinas en el estator alimentadas con corriente continua controlada por los ESC, tal como se describe en [3]. Dicho controlador se encarga de energizar con sincronía los bobinados generando un campo giratorio. La forma de onda generada por los ESC se aprecia en la figura 3.3[1] y es la encargada de energizar cada uno de los bobinados en el instante oportuno.

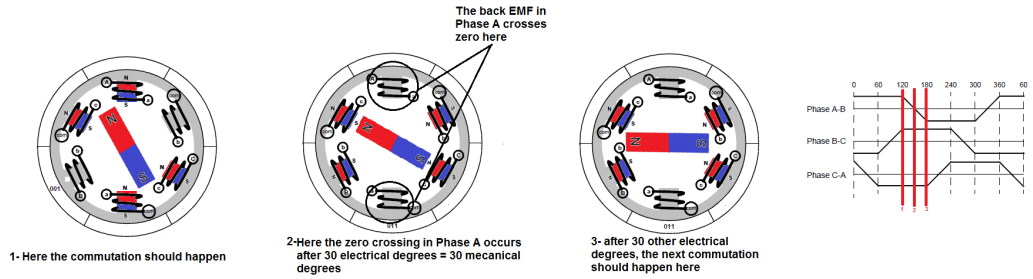


Figura 3.3: Funcionamiento del motor de continua sin escobillas de rotor interno, similar al caso de rotor externo.

Las hélices giran de forma solidaria al eje del rotor y son las encargadas de ejercer la fuerza de empuje. Dicha fuerza depende de las dimensiones de las hélices y de la velocidad de giro de las mismas.

Para la elección del sistema de propulsión se utilizaron dos herramientas online[2][3], útiles para conocer el empuje realizable en función de la configuración.

Elección del motor

La oferta es variada diferenciándose en potencia, velocidad máxima, voltaje de funcionamiento, consumo de corriente y eficiencia. Las características en las cuales se basa la elección son las siguientes.

[1] http://robotics.stackexchange.com/questions/2228/zero-crossing-events-with-brushless-dc-motors

[2] http://personal.osi.hu/fuzesisz/strc_eng/

[3] http://www.ecalc.ch/xcoptercalc.php

- Kv o $rpm/volt$: Es la relación entre la velocidad angular que puede brindar el motor y el voltaje de la batería utilizada. Por ejemplo, un motor de $1000Kv$ con una batería de $12V$ podrá girar a un máximo de $12000rpm$.
- Voltaje de funcionamiento del motor, expresado en el número de celdas de la batería.
- Corriente máxima. Además de brindar una idea de la potencia y consumo, se trata de información importante al momento de elegir un controlador compatible.
- Potencia máxima en *Watts*, brinda una idea del porte del motor.

Se escoge el motor marca *Turnigy* modelo *Multistar* de $700Kv$, sus características principales son:

- RPM/V: $700Kv$. Se trata de un motor con menor velocidad final que la media pero de mayor capacidad de tracción, necesaria en este caso para mover al vehículo más la carga adicional.
- Potencia máxima: $340W$. Se traduce en un empuje máximo del orden del kilogramo típicamente dependiendo de las hélices.
- Voltaje: $2S \sim 4S$ ($7{,}4V \sim 14{,}8V$).
- Corriente máxima: $23A$.
- Número de polos: 14.
- Peso: $99g$.

Elección del ESC

Se prefiere que este controlador disponga de un *firmware* libre en caso que sea necesario realizar configuraciones adicionales o adaptaciones. También debe disponer de un *BEC*, regulador de voltaje que brinde $5V$ a partir de la batería de $11{,}1V$ para la alimentación de la electrónica.

Se opta por los *Afro ESC* de $30A$ de *firmware* abierto *SimonK*[4]. Estos tienen las siguientes características:

- Corriente nominal: $30A$.
- Voltaje: $7{,}4V$ a $14{,}8V$.
- BEC: $5V$, $0{,}5A$.
- Peso: $26{,}5g$.

[4] `https://github.com/sim-/tgy`

Hélices

Existen diversos tipos de hélices, la variación en su forma y dimensiones influyen en la fuerza y la velocidad lograda. Se debe tener en cuenta dos parámetros, el diámetro y el *pitch*[5].

Cuanto mayor sea el Kv de los motores, mayor será la velocidad de giro, por lo que menor debe ser el diámetro de las hélices. Una hélice más grande, utilizando un motor de menor Kv, permite mayor eficiencia y fuerza.

Por otro lado, cuanto mayor sea el *pitch*, mayor será la fuerza ejercida por el motor en cada giro aumentando la demanda de corriente. Valores chicos logran más eficiencia pero menor potencia.

Una última característica de importancia es el material de construcción. Debe ser liviano y rígido para evitar vibraciones que disminuyan la perfomance y estabilidad. Al igual que en el caso del *frame*, la fibra de carbono es el material más apropiado para su fabricación.

Se escogen hélices de diámetro relativamente grande acorde con la potencia y Kv del motor. El *pitch* es más grande que el promedio, permitiendo lograr un mayor empuje.

- Diámetro: 12 pulgadas
- *Pitch*: 6 pulgadas
- Material: Fibra de carbono

3.1.3. Fuente de energía

Se utiliza una batería de $11{,}1V$ formada por tres celdas de polímero de litio, cada una de $3{,}7V$. Poseen alta capacidad de descarga, del orden de los $80A$, característica necesaria debido a que el consumo de cada motor puede superar los $15A$.

Considerando un consumo nominal de $10A$ por motor se espera una autonomía de vuelo de entre siete y ocho minutos para una batería de $5000mA/hr$ de capacidad.

Luego del frame, la batería es el componente más pesado a bordo alcanzando los $420g$.

3.1.4. Regulador de voltaje

Es necesaria una fuente estable de $5V$ para un consumo de al menos $1{,}5A$ por parte de la electrónica. Dado que la única fuente disponible es la batería

[5]Se define como la distancia que avanzaría la hélice al girar una vuelta completa en un medio sólido. No debe confundirse con el ángulo de *Euler*

de 11,1V, se probaron dos alternativas para la conversión de voltaje, ambas utilizadas en distintas etapas del proyecto.

Circuito con regulador LM350T

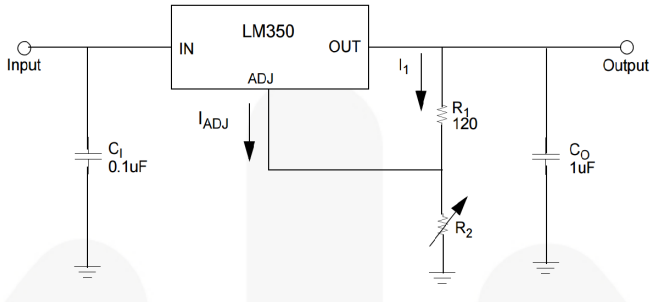


Figura 3.4: Circuito regulador LM350T.

Durante las primeras pruebas se implementa el circuito regulador de voltaje según el esquema de la figura 3.4. En las condiciones previstas cumple las características detalladas en la tabla 3.1.

El voltaje en bornes de R_1 es fijo y vale 1,25V, la salida V_o se expresa en la ecuación 3.1.

$$V_o = 1{,}25(1 + R_2/R_1) + I_{ADJ}R2 \tag{3.1}$$

Despreciando el término $I_{ADJ}R2$, se determina el valor R_2 con el fin de obtener $V_o = 5V$3.2.

$$R_2 = (V_o/1{,}25 - 1)R_1 = 360\Omega \tag{3.2}$$

V_{in}	11,1V
V_{out}	5V
I_{salida}^{max}	4,5A
I_{ADJ}^{max}	100μA

Tabla 3.1: Características del rectificador *LM350T*.

Regulador de voltaje S18V20ALV switcheado

En la implementación final se hace uso de un regulador switcheado ajustable modelo *S18V20ALV*[6] (ver figura3.5). Se opta por esta opción debido a su eficiencia en comparación al *LM350T*, ver tabla 3.2.

[6] `https://www.pololu.com/product/2572`

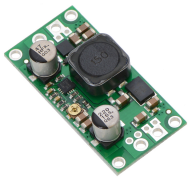


Figura 3.5: Regulador de voltaje ajustable 4-12V S18V20ALV.

V_{in}	$11{,}1V$
V_{out}	$5V$
I_{salida}^{max}	$3A$
Eficiencia	$> 80\,\%$

Tabla 3.2: Características del rectificador *S18V20ALV*.

3.2. Instrumentación

La capacidad sensorial a bordo está compuesta por sensores de proximidad ultra sonido, GPS y una IMU, o Unidad de medición inercial. Adicionalmente se cuenta con sensores incluidos en la placa *CC3D* que se describe más adelante.

3.2.1. IMU

Consiste en una sola placa que agrupa giróscopos, accelerómetros, magnetómetro, barómetro y termómetro. Los datos obtenidos permiten estimar la velocidad, orientación y posición del vehículo siempre que los datos sean tratados de forma adecuada. En el caso específico de este proyecto sólo se hace uso de los datos barométricos para la estimación de la altura de vuelo. Esto se debe a que la placa controladora de actitud *CC3D* posse su propia IMU.

La *Mongoose 9DoF* IMU combina la totalidad de los sensores antes mencionados y consta de un microcontrolador de fácil programación mediante el compilador de *Arduino*. Sus características principales son:

- Microcontrolador: *Atmel MEGA 328P*, $16MGz$.
- Acelerómetro: ADXL345
- Giróscopo: ITG3200
- Magnetómetro: HMC5843
- Barómetro: BMP085
- ADC: $3{,}3V$, 10 bits.

La utilidad específica que se le da a la IMU es la obtención de la medida del barómetro para la estimación de la altura de vuelo junto con la inclusión de los sensores de distancia mediante el uso de dos ADC disponibles. La tabla 3.3 lista las características del barómetro, una descripción completa del resto de los componentes puede ser consultada en [16].

Rango	300 a 1100 hPa (9000 a -500m)
Resolución	1Pa
Precisión. Abs.	typ/max ±1.0/±3.0 hPa
Precisión Rel.	±0.5 hPa
Datos nuevos	typ/max: 3/4.5ms - 17/25ms
Ancho de banda	333/40Hz
Ruído (hPa)	0.06 - 0.03
Ruído (m)	0.5 - 0.25

Tabla 3.3: Características del sensor de presión *BMP085* tomadas de [16].

3.2.2. GPS

El sistema de posicionamiento global (GPS) permite, mediante a la recepción de señales provenientes de una red de 24 satélites, determinar la posición tridimensional de un objeto en cualquier punto de la superficie terrestre. La precisión del instrumento depende de la calidad del dispositivo receptor, de la visibilidad de los satélites y de la distribución espacial de los mismos. El módulo se encarga de recibir las señales de los satélites, procesar la información y dejarla a disposición del resto del sistema.

Existe una gran variedad de receptores en el mercado, la elección se puede hacer en base a las propiedades físicas y técnicas que dispongan.

Características principales de un módulo GPS

Las características fundamentales de este tipo de instrumentos son:

- Protocolo de comunicación: Es la forma en la cual el instrumento presenta los datos para ser utilizados por el resto del sistema. El más ampliamente utilizado es el protocolo NMEA (National Marine Electronics Association) que mediante distintas sentencias en ASCII brinda información sobre la posición, velocidad, hora, cantidad de satélites y posición de los mismos.

- Sensibilidad de la antena medida usualmente en dBm.

- Frecuencia de actualización. Refiere al tiempo que transcurre entre la disponibilidad de dos datos consecutivos. Resulta muy importante disponer de la máxima tasa de actualización posible, ya que de ésta dependerá la velocidad de muestreo de los algoritmos diseñados.

- Peso y consumo: Al igual que para toda la electrónica del proyecto, se busca el menor consumo y peso posible.

Elección

Se utiliza el módulo *Adafruit Ultimate GPS Breakout* de la figura 3.6 con las siguientes características según el fabricante:

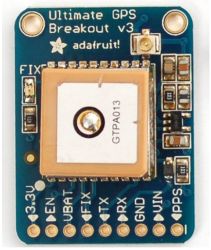


Figura 3.6: Adafruit Ultimate GPS Breakout.

- Protocolo: *NMEA 0183*, mediante UART hasta 115200 baudios.

- Frecuencia de actualización: 1 a $10Hz$.

- Antena: $-165dBm$, es capaz se detectar hasta 22 satélites en 66 canales .

- Rango de alimentación: $3V$ a $5VDC$.

- Consumo: 20 a $25mA$.

- Precisión en posición: $< 3m$.

- Precisión en velocidad: $< 0{,}1m/s$.

- Arranque en frío[7] : $34seg$.

3.2.3. Sensor de proximidad

El sensor de proximidad es útil para la detección de objetos que se coloquen frente al mismo y permite obtener una medida de la distancia entre el sensor y el objeto detectado. En el presente proyecto se utiliza con el fin de detectar obstáculos durante el vuelo y para obtener una estimación de la altura de vuelo mientras su alcance lo permita.

Dentro de la variedad existente en el mercado se selecciona un sensor de ultrasonido. Su funcionamiento se basa en la emisión de una onda ultrasónica

[7]El arranque en frío se realiza cada vez que el módulo GPS se apaga sin fuente de alimentación de respaldo conectado. Almanaque y datos de efemérides deben ser descargados de los satélites para poder comenzar a brindar datos validos, este proceso lleva al rededor de 35 segundos típicamente

con una frecuencia del orden de los $40KHz$, que es detectada por un receptor luego de reflejarse en el objeto que se encuentre de frente. Conociendo los tiempos de emisión y recepción de la onda generada puede calcularse la distancia hasta el objeto en el cual se refleja, esto es:

$$d = \frac{v(t_{emision} - t_{recepcion})}{2} \tag{3.3}$$

Donde d es la distancia entre el sensor y el objeto sensado, v es la velocidad de propagación de la onda en el medio específico y $t_{emision}$ y $t_{recepcion}$ son los tiempos en que se emite y se detecta la onda respectivamente. El factor de proporcionalidad $1/2$ surge por el hecho de que la onda recorre dos veces la distancia a medir.

Elección

El sensor seleccionado para cumplir esta tarea es el *MaxBotix 1340*[8]. Sus principales características son las siguientes:

- Rango de alimentación: $3{,}3V$ a $5VDC$
- Detección de objetos a partir de $1mm$
- Rango de medición: $20cm$ a $765cm$
- Salidas disponibles:
  - Analógica
  - Serial
- Frecuencia de actualización: Hasta $10Hz$
- Consumo: $2{,}1mA$ a $3{,}4mA$ dependiendo de la alimentación utilizada

3.3. Inteligencia

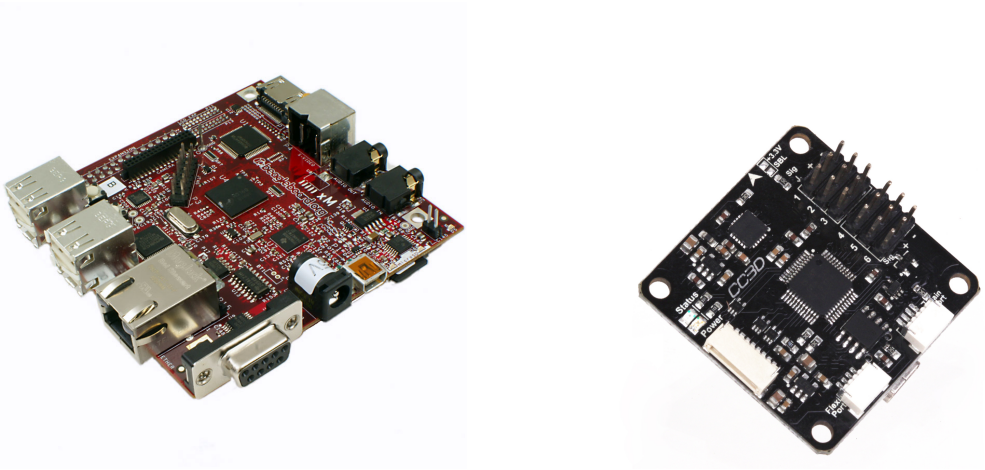
A diferencia de otro tipo de vehículos, la inteligencia a bordo es fundamental para el vuelo de los multirotores debido a su gran inestabilidad, razón por la cual requieren la acción de diversos controladores. Si se pretende dotar de autonomía de vuelo al vehículo, se necesita hardware capaz de correr algoritmos de generación y seguimiento de trayectorias en tiempo real.

La inteligencia está repartida en dos sistemas distintos, uno de bajo nivel encargado del control de actitud y comando de motores y otro de alto nivel encargado de la generación de trayectorias, seguimiento de las mismas, interfaz con el usuario y comunicación con periféricos.

[8] `http://www.maxbotix.com/documents/XL-MaxSonar-EZ_Datasheet.pdf`

3.3.1. Autonomía de vuelo

Se utiliza una computadora del tipo *Single board computer* para ejecutar los distintos algoritmos de autonomía. Un entorno *Linux* facilita el desarrollo debido a la disponibilidad de una gran oferta de librerías, funcionalidades de interfaz, multitarea, entre otras ventajas.



(a) Beagle Board - XM (b) *CC3D*

Figura 3.7: Inteligencia del cuadrióptero.

La *BeagleBoard-XM* (figura 3.7a) es una placa de desarrollo *open source* que corre una distribución de *Linux* llamada *Ångström*, especialmente diseñada para sistemas de bajos recursos. Posee un microprocesador *TI ARM Cortex A8* de $1GHz$ y un DSP *TMS320C64x+* de $800MHz$. La memoria RAM es de $512Mb$. Posee puertos UART y USB para la comunicación con periféricos y con el sistema de bajo nivel. A su vez dispone de una placa de red con puerto Ethernet y un puerto RS232. Mediante un *dongle WiFi* se levanta una red *AD-Hoc* con la cual se accede remotamente al sistema.

3.3.2. Control de actitud

La placa *CC3D* es un controlador de actitud *open hardware* y *open software* desarrollado por *Open Pilot* [8]. Se encarga de controlar los ángulos de *Euler* del sistema haciendo uso de una IMU integrada que posee acelerómetros y giróscopos. Recibe como entradas el estado deseado de los ángulos *Pitch* y *Roll*, la velocidad de giro del ángulo *Yaw* y el *Throttle*. Utilizando dichas entradas y la información de los sensores establece apropiadamente la velocidad de giro de los motores. Deja a disposición del resto del sistema los datos obtenidos a partir de la IMU incorporada (telemetría). Sus características son:

- Procesador: *STM32* 32-bit corriendo a $90MIPs$ con $128KB$ *Flash* y $20KB$ RAM.
- EEPROM: $4Mbits$.
- IMU: *MPU6000* .
- Comunicacion: USB, Serial y PWM.
- Peso: $30g$
- Consumo: $5V$, $70mA$.

3.4. Tabla de componentes utilizados

En la tabla 3.4 se resumen todos los componentes utilizados para el armado del cuadricóptero junto con el peso y precio de cada elemento. Se agregan además algunos elementos no mencionados.

Componente	**Modelo**	**Peso** g	**Precio** $US\$$	**Cantidad**
Frame	Tarot - Ironman 650	476	102,58	1
Motor	Turnigy - Multistar 700Kv	105	26,95	4
Hélice	Hobbiking 12X6	18	9,49	4
Controlador	Afro ESC - 30A	27	14	4
Batería	Floureon LiPo 3S 4500mAh	286	32	1
Distribuidor de energía	XT60	30	9	1
Regulador switcheado	S18V20ALV	6	20	1
Computadora a bordo	Beagle Board - XM	200	150	1
Controlador de Actitud	*CC3D*	13	30	1
Sensor ultra sonido	Maxbotix - MB1340	6	30	2
IMU	Mongoose 9DOF	5	0	1
Conversor de niveles lógicos	Adafruit TXB0108	4	15	1
FTDI		6	0	1
Enlace RF	Hobbypower Apc220	14	40	1
Dongle WiFi	7i nova	6	0	1
Bandeja de acrílico	Hecha a medida	42	25	1

Tabla 3.4: Componentes que componen al cuadricóptero.

La suma de todos los componentes tiene un costo de U$S 685,34 y el peso total es de $1700g$ sin considerar conectores, cables etc.

Parte II

Caracterización de los componentes del sistema

Capítulo 4

Caracterización del sistema de propulsión

El sistema de propulsión del cuadricóptero está constituido de motores, controladores y hélices. Resulta fundamental caracterizar dicho sistema, relacionando la señal de comando enviada a los ESC con la fuerza de empuje que el sistema de propulsión realiza sobre el cuadricóptero.

4.1. Objetivo de la medida

Hallar la relación existente entre el ancho de los pulsos PWM (medido en μs) enviado al ESC con los siguientes parámetros de respuesta:

- Velocidad angular desplegada (rpm)
- Fuerza ejercida o empuje (gramos)
- Consumo (Ampres)

Las medidas obtenidas son necesarias para el diseño del control y la determinación del punto de operación. Además, conocer el consumo permite calcular tiempos de vuelo esperados según la carga que transporta el vehículo y la capacidad de la batería.

4.2. Materiales utilizados

Conjunto de componentes a caracterizar:

- Motor marca *Turnigy*, modelo *Multistar 2814-700*.
- Hélice 12x6 (12 pulgadas de diámetro y 6 pulgadas de *pitch*).

- ESC marca *Afroesc*, modelo de $30A$.
- Batería *LiPo* $3300mAh$ $30C$.

Equipos de medida y dispositivos auxiliares para la realización del experimento (ver figura 4.3):

- Instrumento de medida de velocidad de la hélice[1] desarollado por el grupo de proyecto *Uquad!* [16]. Su funcionamiento requiere los siguientes equipos:
  - Generador de señales marca *Tektronix*, modelo *CFG250*.
  - Fuente de $5V$ DC.
  - Oscliloscopio marca *Tektronix*, modelo *TBS 1062*.
- Banco de medida de fuerza armado especialmente[2], requiere los siguientes materiales.
  - Balanza marca *Camry*, modelo *EK9320* , resolución $1g$.
  - Soporte especialmente diseñado, ver figura 4.3.
- Multímetro marca *Extech*, modelo *MN35*, para medir el consumo.

4.3. Procedimiento

La velocidad angular del los motores se controla variando el ancho de los pulsos de la señal PWM enviada a los controladores. Se define ξ como el ancho de los pulsos medido en μs y ω como la velocidad angular del motor en rpm.

El rango de funcionamiento es $1062\mu s \leq \xi \leq 2000\mu s$ según se comprobó experimentalmente. Para pulsos de $\xi > 2000\mu s$ el motor satura. Con

[1] Consiste en un emisor y un sensor infrarojo enfrentados, la frecuencia de la señal cuadrada obtenida al hacer pasar las hélices y obstruir la linea de vista resulta el doble que la frecuencia de giro.

[2] Una forma práctica de medir la fuerza ejercida por el motor es utilizando una balanza y un contrapeso. En éste caso el contrapeso consiste en una barra de hierro horizontal. Uno de sus extremos va apoyado sobre la balanza y el otro sobre una superficie firme. Con el motor situado en el centro de la barra se logra que el flujo de aire desplazado por la hélices no ejerza presión sobre la bandeja de la balanza distorsionando la medida. El peso de la configuración se repartirá equitativamente en los dos puntos de apoyo por lo que la medida de la balanza representa la mitad de la fuerza ejercida por el motor. Ver figura 4.3.

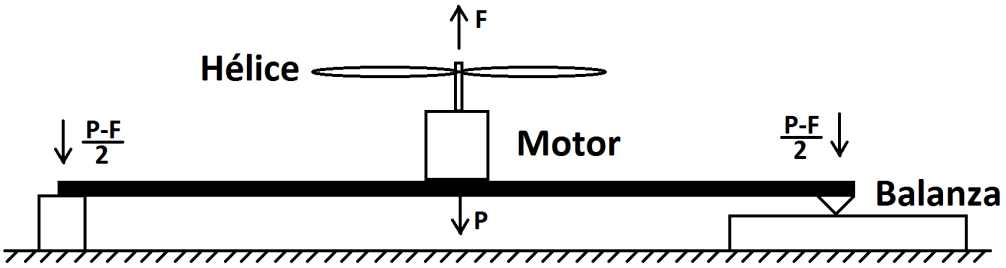


Figura 4.1: Diagrama del banco de medida utilizado para la caracterización de fuerza.

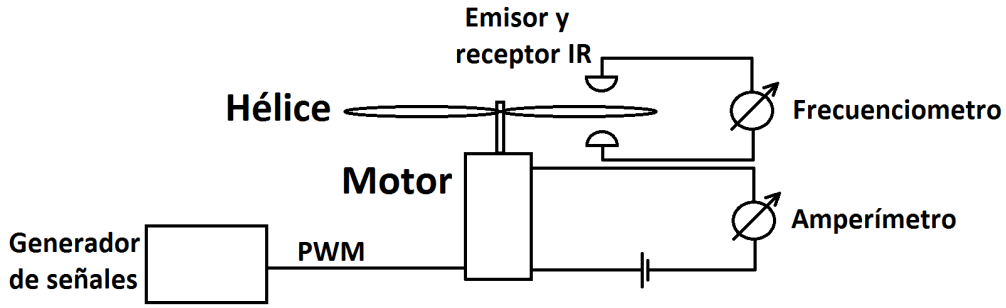


Figura 4.2: Diagrama del banco de medida utilizado para la caracterización de velocidad y consumo.

el fin de cubrir todo el rango de funcionamiento, la señal PWM es variada de a pasos iguales de $40\mu s$, comenzando con $\xi = 1062\mu s$ hasta $\xi = 1800\mu s$.

Se registra el consumo conectando el multímetro en serie a la batería. La fuerza realizada se registra según la lectura de la balanza. La velocidad angular se obtiene midiendo la frecuencia de la señal cuadrada, obteniendo el resultado en rpm al multiplicar la frecuencia obtenida por 60/2 (ver figuras 4.1 y 4.2). La barra de hierro se apoyada sobre la balanza y un punto de apoyo quedando horizontal.

4.4. Resultados

En la gráfica de la figura 4.4 se aprecian las curvas de velocidad, fuerza y consumo en función del comando PWM enviado al ESC.

4.4.1. Velocidad de giro

La relación entre el ancho del pulso PWM ξ y la velocidad de giro ω resulta prácticamente lineal dentro del rango estudiado según la ecuación 4.1.

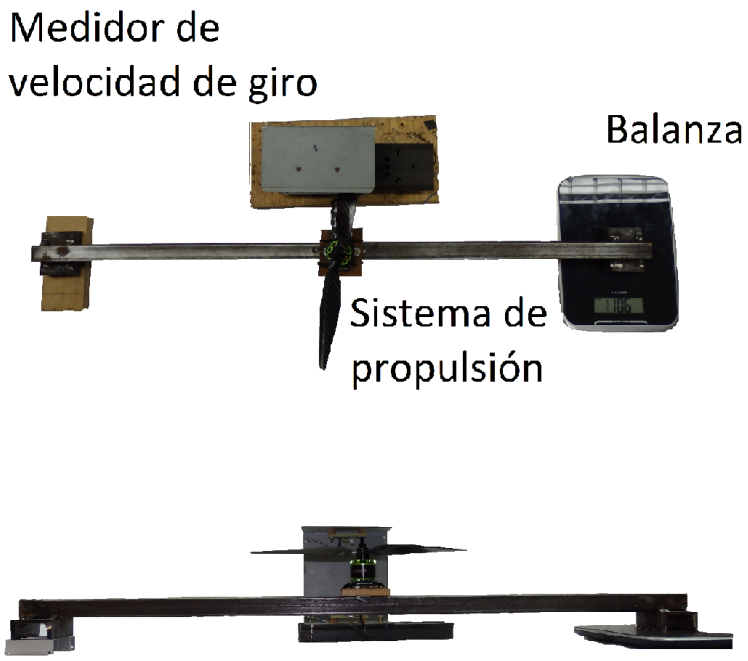


Figura 4.3: Configuración utilizada para la caracterización.

$$\omega(\xi) = 6{,}0554\xi - 5444{,}9 \tag{4.1}$$

4.4.2. Empuje

La fuerza ejercida por el sistema de propulsión crece potencialmente respecto al comando PWM. Realizando una aproximación de orden dos se obtiene la ecuación de ajuste 4.2.

$$Th(\xi) = 0{,}6 \times 10^{-3}\xi^2 - 0{,}63\xi + 47{,}05 \tag{4.2}$$

4.4.3. Consumo

Como se aprecia en la gráfica 4.4 el consumo crece rápidamente en todo el rango de funcionamiento. Considerando que el peso del vehículo es de aproximadamente $1700g$, serían necesarios $4{,}2A$ por motor para mantener al vehículo en el aire.

4.4.4. Punto de operación

El punto de operación del sistema es aquel en el cual la suma de las fuerzas ejercidas por los cuatro motores iguala a la fuerza peso. La tabla 4.1

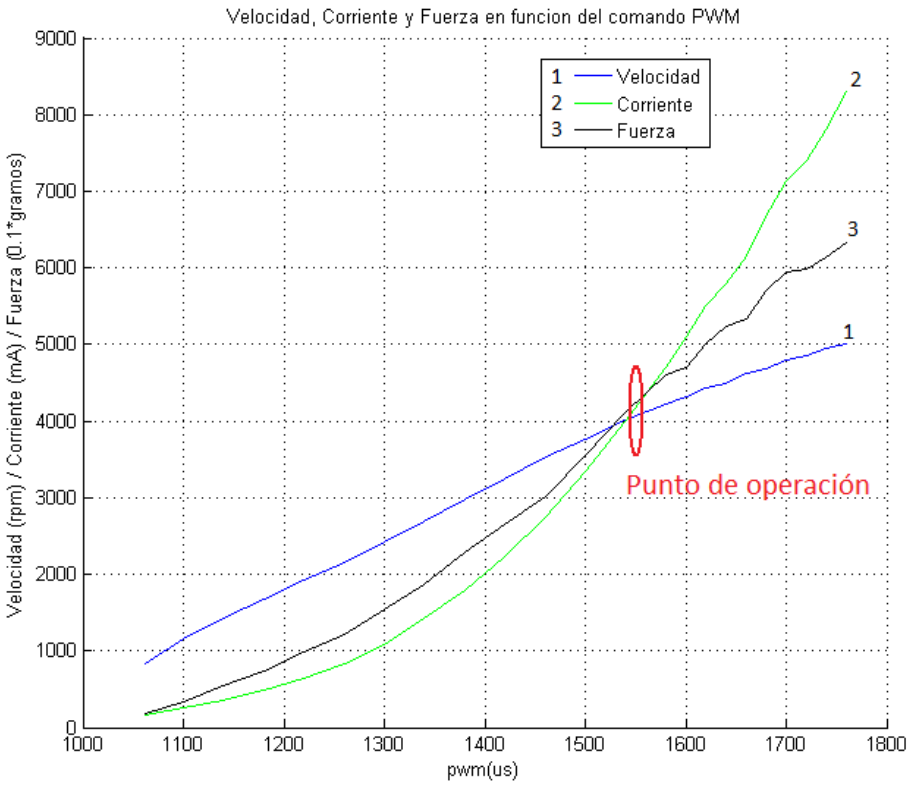


Figura 4.4: Parámetros relevados.

muestra el estado del sistema en el punto de operación.

Comando PWM (μs)	1552
Velocidad de giro (rpm)	4080
Fuerza (gramos)	425
Consumo (A)	4.2A

Tabla 4.1: Punto de operación.

Considerando un consumo por parte de la electrónica de $2A$, el consumo total del sistema asciende a $18{,}8A$. Con una batería de $4500mA/h$ se obtendría una autonomía de 14 minutos aproximadamente.

4.5. Conclusiones

Se comprobó satisfactoriamente el poder de propulsión disponible. Se determina el punto de operación del sistema permitiendo estimar el consumo.

La gráfica de la figura 4.4 es información necesaria para el desarrollo del proyecto, ya que permite conocer los comandos necesarios a enviar a cada ESC en función de la fuerza que se desea imponer.

Capítulo 5

GPS

Los datos obtenidos a partir del módulo GPS son usados para la estimación de la posición y la velocidad instantánea del cuadricóptero. Tanto el algoritmo de seguimiento de trayectorias como el controlador de velocidad de desplazamiento dependen de éstos datos, por lo que resulta imprescindible conocer las características y la calidad de las estimaciones.

5.1. Funcionamiento del GPS

La antena del GPS recibe señal de una red global que consta de veinticuatro satélites cuyas órbitas cubren la totalidad de superficie terrestre. El receptor debe recibir información de al menos cuatro satélites para determinar en qué posición tridimensional se encuentra ubicado. El error en la medida obtenida depende diversos factores[1].

El Módulo GPS utilizado está configurado para brindar datos con una frecuencia de $10Hz$. Las mediciones se realizan siempre al aire libre con una visibilidad de al menos 7 satélites.

5.2. Objetivo de la medida

Se analiza el desempeño del módulo *Adafruit Ultimate GPS Breakout* como dispositivo de medida de los siguientes parámetros en tiempo real.

- Posición absoluta (x, y, z) en quietud.
- Posición absoluta (x, y, z) en velocidad.

[1]Influye el retraso de la señal en la ionosfera y la troposfera, el deterioro y posibles rebotes de la señal, la poca cantidad de satélites visibles y su distribución geométrica , los errores numéricos y los errores que comenten los propios satélites al calcular su posición.

- Orientación o ángulo acimutal[2].
- Módulo de la velocidad sobre la superficie.

5.3. Materiales utilizados

Hardware

- Módulo GPS *Adafruit Ultimate GPS Breakout.*
- PC.
- FTDI (interfaz entre GPS y PC).
- Cinta métrica.
- Varilla de madera de $2m$ de largo.
- Vehículo con velocímetro.

Software

- *Matlab.*
- Parseador de sentencias NMEA[3].
- *Google Earth.*

5.4. Procedimiento y resultados

La práctica se divide dos partes independientes. El análisis de las medidas de posición (x, y, z) se realizan en una cancha de *fútbol 5* de dimensiones conocidas (ver figura 5.1). El análisis de los datos de velocidad y dirección se llevan a cabo en la calle haciendo uso de un vehículo con velocímetro.

Para las medidas en la cancha se posiciona el GPS en el extremo de una varilla de $2m$ de largo. Manteniendo en posición vertical a la varilla, se asegura altura contante en todos los puntos sensados.

Las medidas originales de posición provistas por el GPS vienen dadas en coordenadas geográficas. El resultado se transforma a coordenadas cartesianas según modelo UTM descrito en [10].

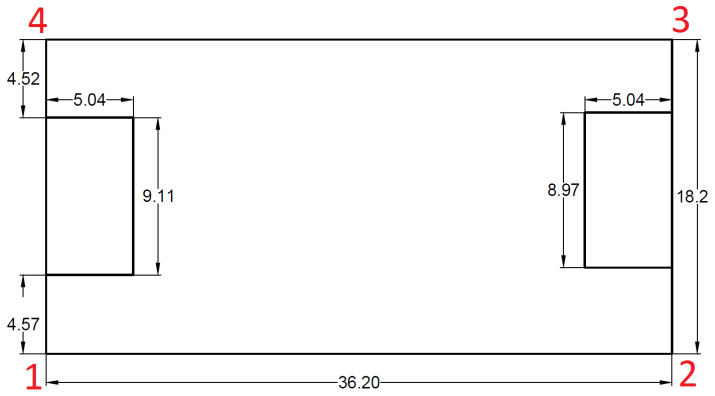


Figura 5.1: Dimensiones de la cancha en metros.

Punto	$\overline{x}$ (m)	σ_x (m)	$\overline{y}$ (m)	σ_y (m)	$\hat{\sigma}$
1	0	2.41	0	3.86	4.62
2	-16.90	1.42	-31.64	3.15	3.62
3	-1.65	1.49	-43.80	2.32	2.76
4	15.87	1.38	-10.31	2.08	2.51

Tabla 5.1: Promedio de las medidas en los extremos de la cancha referenciadas al *punto 1*.

5.4.1. Medidas en puntos fijos

Con el módulo en caliente[4] y la varilla apoyada verticalmente se mide durante cinco minutos en los cuatro vértices de la cancha. La tasa de actualización de datos es de $10Hz$, obteniéndose aproximadamente 3000 muestras por punto. Se fija como origen de coordenadas la estimación para el *punto 1*[5].

Resultado para coordenadas planas $(x.y)$

La figura 5.2 muestra la dispersión de las medidas en torno a los extremos de la cancha, en la tabla 5.1 se estima la ubicación de cada vértice realizando la media de las medidas. Se toma como referencia para los resultados a la estimación obtenida para el *punto 1*. La desviación estándar promedio para los cuatro extremos resulta de 3,38m.

[2]El acimut se mide desde el punto cardinal norte en sentido horario de 0° a 360°.

[3]`http://freenmea.net/decoder`

[4]El GPS necesita aproximadamente 35 segundos luego del encendido para comenzar a brindar datos tan precisos como es posible.

[5]El origen de coordenadas para la zona geográfica UTM *21 H* dista aproximadamente $580Km$ en x y $6140Km$ en y de Montevideo, se toma como origen de coordenadas el *punto 1* para evitar el manejo grandes números.

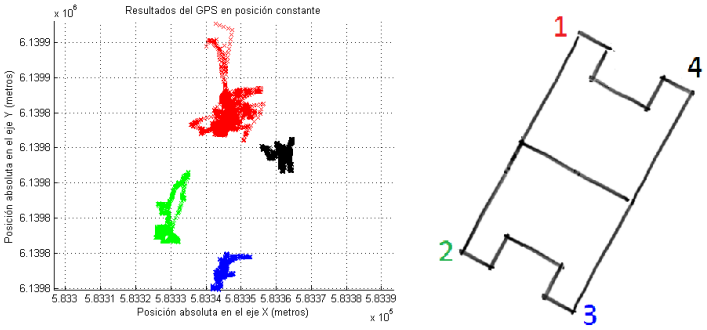


Figura 5.2: Mediada durante 5 minutos en los cuatro extremos de la cancha.

Con el fin de evaluar la correlación entre los datos del GPS y las dimensiones reales de la cancha[6] se realiza el ajuste de la figura 5.3. El rectángulo apreciado en la figura tiene las dimensiones reales de la cancha y está posicionado de forma tal que sus extremos minimizan la distancia con los datos obtenidos, en sentido de mínimos cuadrados. Según datos expuestos en la tabla 5.2 los extremos de ambas canchas distan en promedio $1{,}23m$, demostrando una buena exactitud relativa respecto al tamaño original de la cancha.

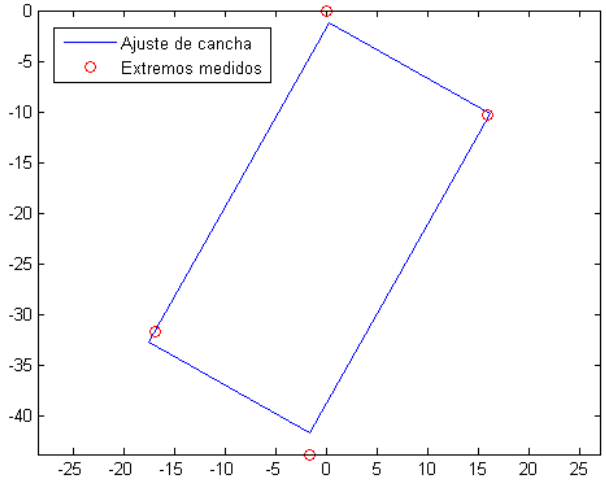


Figura 5.3: Cancha de dimensiones reales, ajustada minimizando distancias entre extremos y puntos estimados.

Resultados de altura

En la tabla 5.3 se aprecian los resultados para las estimaciones de altura sobre el nivel del mar en cada uno de los extremos de la cancha. Si bien los

[6]medidas con una incertidumbre menor a cinco centímetros mediante el uso del metro

Punto	Distancia (m)
1	1.25
2	1.22
3	2.12
4	0.33
Promedio	1.23

Tabla 5.2: Distancias entre datos GPS y extremos de cancha ajustada.

Punto	Altura $\overline{h}$ (m)	σ_h (m)
1	35.74	7.78
2	40.91	10.69
3	30.75	8.62
4	36.17	4.98

Tabla 5.3: Altura en puntos fijos respecto al nivel del mar.

cuatro puntos fueron medidos a la misma altura, los resultados promediados arrojan diferencias de hasta $10m$. A su vez, la desviación estándar para las medidas para cada extremo llega a ser de $10{,}69m$. Claramente no se obtienen datos precisos de la altura.

Conclusiones

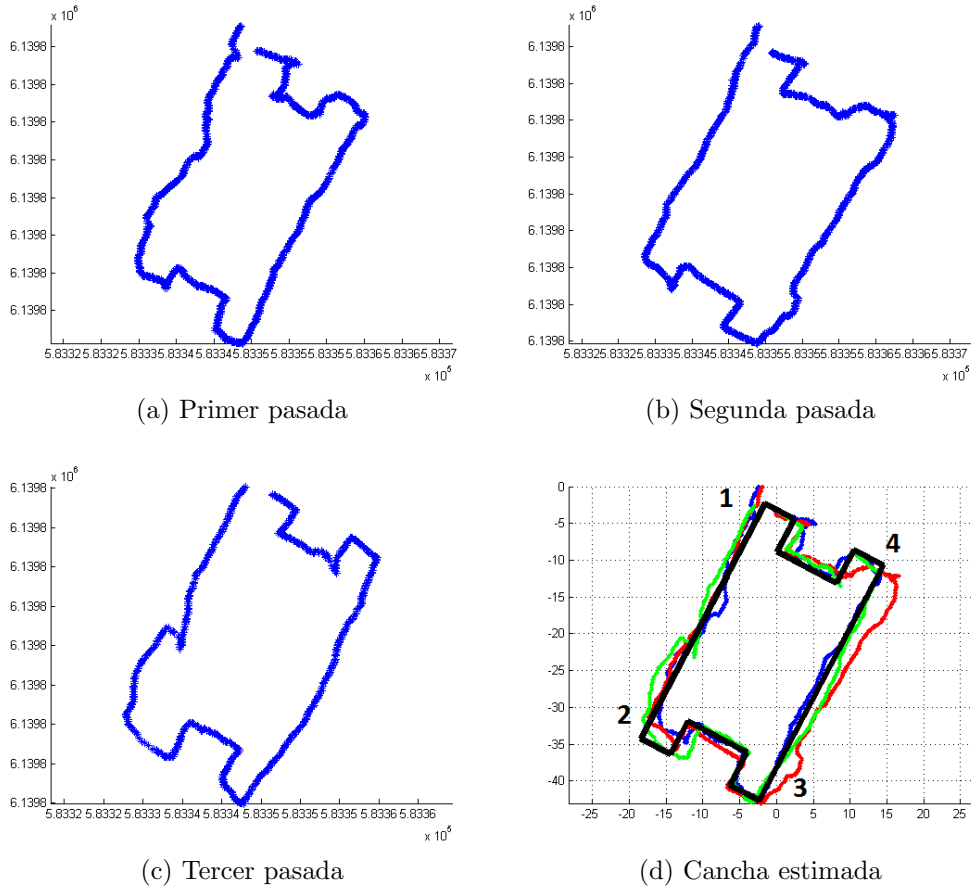
Las medidas de posición brindadas por el GPS en quietud presentan gran dispersión, especialmente al sensar la altura. Sin embargo el promediado durante cinco minutos brinda una estimación adecuada de la posición en el plano.

5.4.2. Medidas en movimiento

El perímetro de la cancha de *fútbol 5* es recorrido montando el GPS sobre la varilla, que es guiada verticalmente sobre las líneas. Se realiza el recorrido en tres ocasiones a altura constante y una velocidad de $3m/s$ aproximadamente. Las medidas sensadas se observan gráficamente en la figura 5.4.

La figura 5.4d muestra la superposición de los cuatro recorridos junto al ajuste de la cancha de dimensiones reales, según el método de mínimos cuadrados. La máxima distancia entre los datos obtenidos y la cancha estimada es de 3,59. En la mayor parte de los casos no se superan los dos metros de distancia.

En la figura 5.5 se superponen ambas canchas ajustadas por mínimos cuadrados a partir de las medidas fijas y dinámicas. Se grafican también los extremos medidos en quietud. Una comparación entre las tres estimaciones



(a) Primer pasada (b) Segunda pasada

(c) Tercer pasada (d) Cancha estimada

Figura 5.4: Posición estimada al recorrer el perímetro de la cancha.

Punto	Cancha estática (m)	Cancha dinámica (m)	Distancia entre canchas (m)
1	1.24	2.72	2.16
2	1.25	3.06	1.81
3	2.13	1.29	1.26
4	0.33	1.40	1.73
Media	1.24	2.12	1.74
Desv. Estándar	0.73	0.90	0.37

Tabla 5.4: Comparación entre vértices y extremos medidos, comparación entre vértices.

(çancha estática", çancha dinámicaz .[ex]tremos medidos") es presentada en la tabla 5.4. La máxima distancia entre ambos ajustes se da en los vértices, siendo de 2,16m en el peor de los casos.

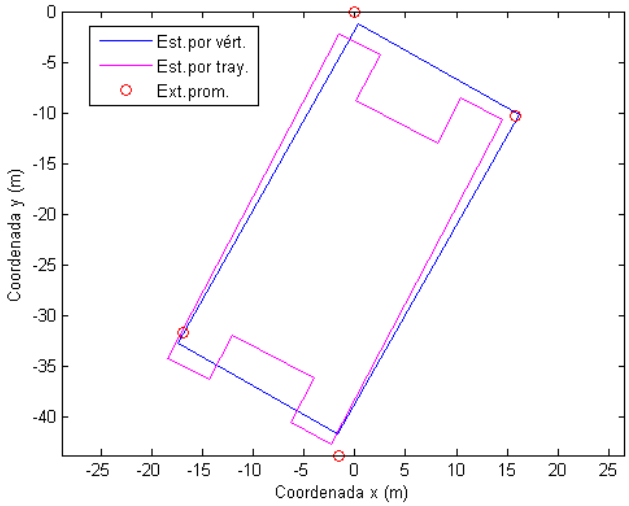


Figura 5.5: Comparación, caso estático y dinámico.

Conclusiones

La exactitud obtenida relativa a las dimensiones reales de la cancha supera las expectativas. La diferencia registrada es menor a dos metros en la mayoría de las medidas. Comparando los resultados obtenidos con las estimaciones en quietud de la sección 5.4.1, se comprueba una fuerte correspondencia distando $2{,}16m$ para el peor caso.

5.4.3. Velocidad y Orientación

No es fácil determinar qué tan bueno es el GPS como estimador de la velocidad, debido a que no se dispone del equipamiento de medida que permita fijar una velocidad constante con precisión. Se opta por realizar un experimento cualitativo que permita obtener una idea de la capacidades del instrumento de medida en condiciones similares a las que será utilizado.

Con la ayuda del velocímetro de un automóvil y el trazado de una calle recta se puede lograr un movimiento rectilíneo a velocidad relativamente constante.

Se coloca el dispositivo horizontalmente sobre el techo de un automóvil y se fija una velocidad y dirección constante. Se maneja a tres distintas velocidades, luego se analiza el valor y la constancia de las medidas. Se realizan dos pasadas a $30Km/h$, una tercer pasada a $15Km/h$ y una última a $35Km/h$. Los cuatro recorridos realizados se aprecian en la figura5.6.

La velocidad media y la desviación estándar obtenida se aprecia en la tabla 5.5.

El valor de las medidas es compatible con la lectura del tacómetro del au-

Figura 5.6: Cuatro recorridos realizados a velocidad y dirección constante.

	Vel. media (Km/h)	Vel. media (m/s)	Desviación std. (m/s)
Caso 1	30.8293	8.5637	0.1689
Caso 2	34.5421	9.5950	0.0914
Caso 3	14.1242	3.9234	0.0973
Caso 4	35.0452	9.7348	0.3638

Tabla 5.5: Velocidad estimada por el GPS.

tomóvil. Los resultados son relativamente constantes, la desviación estándar es de 3,7 % del promedio para el peor caso.

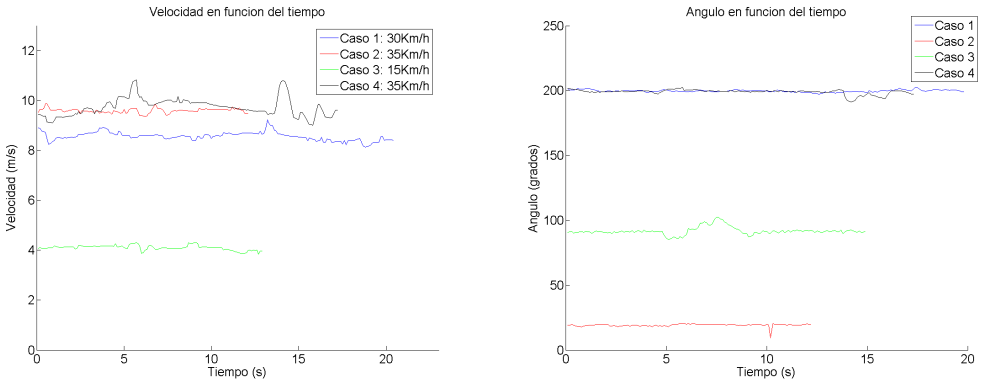
La orientación obtenida se espera constante al igual que la velocidad. La gráfica de la figura 5.7b presenta la dirección en función del tiempo, en la tabla 5.6 se promedian las medidas para cada caso.

Existe una buena precisión en los datos arrojados por el GPS, en el peor de los casos la desviación estándar asciende a los 3° que representa el 3,62 % de la medida.

Es importante remarcar la coincidencia en los resultados del *caso 1* y *caso 4* (menos de un grado de diferencia), ya que ambos fueron realizados con la

	Ángulo medio (grados)	Desviación std. (grados)
Caso 1	199.7065	0.8630
Caso 2	19.2202	1.1031
Caso 3	91.7822	2.9974
Caso 4	198.9507	1.8987

Tabla 5.6: Orientación estimada por el GPS.



(a) Velocidad en función del tiempo. (b) Orientación en función del tiempo.

Figura 5.7: Análisis de la información de velocidad (módulo y orientación) entregada por el GPS.

	Ángulo medio, GPS	Ángulo Google Maps	Diferencia
Caso 1	199.7	198.4	1.3
Caso 2	19.2	19.3	0.1
Caso 3	91.8	92.9	0.1
Caso 4	199.3	199.0	0.3

Tabla 5.7: Comparación con datos de Google Earth.

misma orientación. El *caso 2* se realizó en igual dirección pero desfasado 180°. Realizando la suma $19{,}22° + 180° = 199{,}22°$ se comprueba nuevamente que se mantuvo por abajo de un grado de diferencia.

En la tabla 5.7 se contrastan los resultados estimados con la orientación calculada mediante *Google Earth*.

Existe gran relación entre los datos, demostrando que la dirección estimada por el GPS en movimiento arroja menos de 3° de desviación estándar y menos de 1,5° de error respecto a la medida obtenida en *GoogleEarth*.

5.5. Conclusiones

La calidad de los datos es sensiblemente mejor si la medida se realiza en velocidad. Esto no es un problema ya que el funcionamiento normal es con el vehículo en movimiento. En dichas condiciones se logran medidas con errores del orden de los dos metros.

En quietud la dispersión de los resultados es alta, sin embargo el promedio durante cinco minutos brinda un buen estimativo de la posición en el plano.

Las medidas de altura, tanto en quietud como en movimiento, no satisfacen los requerimientos básicos para el funcionamiento del controlador de altitud.

Capítulo 6

Sensor de proximidad

El uso de sensores de proximidad es de gran utilidad en diversos sistemas móviles, permitiendo reconocer la presencia de objetos en determinada dirección y saber a que distancia se encuentran.

En el presente proyecto se utilizan dos sensores, uno en dirección vertical con intención de conocer la altura de vuelo respecto al suelo y otro en la dirección de movimiento para detectar la presencia de obstáculos.

Se analizaron dos sensores de distinto funcionamiento, un sensor infrarrojo y otro de ultrasonido.

El sensor infrarrojo utiliza un emisor de onda infrarroja ($300GHz - 430THz$)[1] y por lo tanto muy direccional, que al reflejarse sobre una superficie es recibida por un receptor sensible al ángulo de incidencia, permitiendo calcular la distancia conociendo la separación entre emisor y receptor.

Por su lado el sensor de ultrasonido emite una onda ultrasónica de $42KHz$ que al reflejarse sobre una superficie también es recibida, pero en este caso se utiliza la velocidad de propagación de la onda y el tiempo que tardó la misma entre que se emitió y se recibió para calcular la distancia al objeto.

Los sensores utilizados fueron:

- Sensor de ultrasonido MaxBotix MB1340
- Sensor infrarrojo Sharp GP2Y0A21YK0F

6.1. Objetivo

En esta sección se estudian ambos sensores con el fin de conocer su performance real, rango de funcionamiento, tiempos de respuesta, etc. Además se comparan ambas alternativas con el objetivo de conocer cual opción se adapta mejor a cada una de las necesidades planteadas, medición de altura de vuelo y detección de obstáculos.

[1] `http://en.wikipedia.org/wiki/Infrared`

6.2. Sensor infrarrojo

6.2.1. Materiales

Los materiales utilizados para el estudio del sensor infrarrojo fueron los siguientes:

- Sensor infrarrojo Sharp GP2Y0A21YK0F
- Placa de desarrollo *Arduino Due*
- Computadora
- Cinta métrica
- Superficie sólida plana (obstáculo)

La interfaz del sensor consta de tres pines, dos de los cuales son para la alimentación de $5V$ y un tercer pin de salida cuyo voltaje es proporcional a la distancia medida.

Figura 6.1: Sensor infrarrojo.

Según la hoja de datos del fabricante, la medida no es influenciada fácilmente por factores del entorno como podrían ser temperatura ambiente o las diferencias en la reflectividad de los distintos objetos con los que pueda rebotar el haz infrarrojo. Estos sensores son alimentados con $5V$ y tienen como salida un voltaje que se corresponde con la distancia medida según la curva que se muestra en la figura 6.2. Tienen un consumo en operación de unos $33mA$ y el rango de medida está comprendido entre los $10cm$ y los $150cm$ según los datos de la hoja del fabricante. Como se puede apreciar esta relación no es lineal y más importante aún, la función no es inyectiva, ocasionando que medidas menores a los 15cm aproximadamente sean interpretadas erróneamente.

La placa de desarrollo *Arduino Due* se basa en un microcontrolador *Atmel SAM3X8E ARM Cortex-M3 CPU*[2] de $512KBytes$ de memoria flash, $96KBytes$ de RAM y un reloj de $84MHz$. Además posee una salida de $5V$ y entradas analógicas con conversores ADC que permiten hacer uso del sensor.

El objeto utilizado como obstáculo durante la práctica fue una caja de cartón de $63 \times 53{,}5cm$.

6.2.2. Procedimiento

Las conexiones realizadas para el procedimiento de la práctica fueron las siguientes:

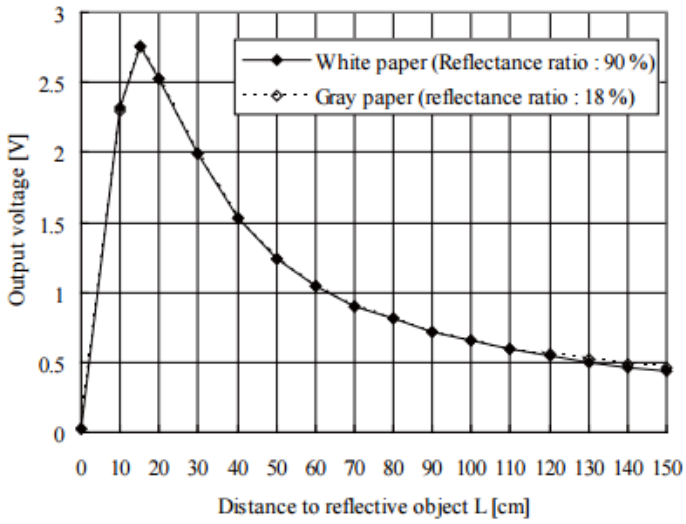


Figura 6.2: Salida analógica en función de la distancia según hoja de datos.

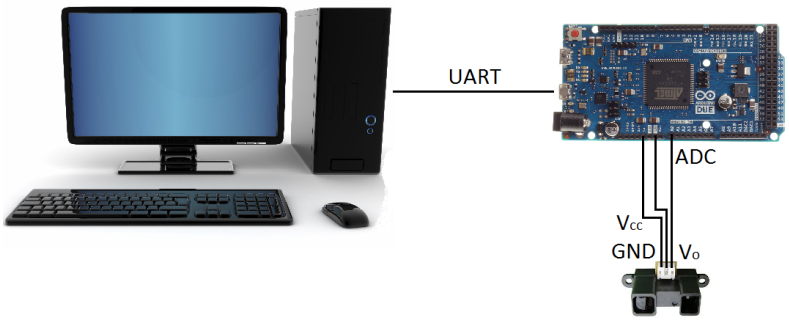


Figura 6.3: Diagrama de conexiones.

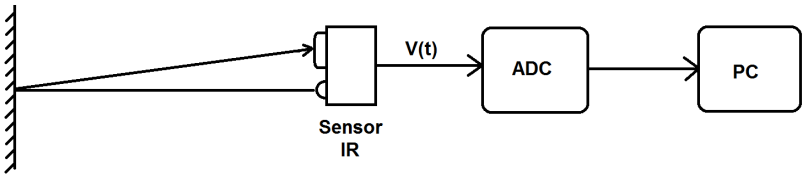


Figura 6.4: Esquema de medición.

Se colocó el sensor en una posición fija horizontal y de frente la superficie plana para que refleje el haz, utilizando la cinta métrica para asegurar las distancias deseadas en cada medición. Por otro lado en la *Arduino Due* se cargó un programa que únicamente convierte la medida del sensor y la envía a la PC mediante una UART. De esta forma se registraron unas 200 mues-

[2]http://www.arduino.cc/en/Main/ArduinoBoardDue

tras del ADC en varias medida de distancia entre $10cm$ y $150cm$. Luego se ajustaron los datos para obtener una expresión analítica que permita calcular cualquier distancia dentro del rango especificado.

Finalmente se realizaron nuevas medidas registrando la distancia calculada y se compararon con las distancias reales para estimar el error que presenta el uso de este sensor.

6.2.3. Resultados

El ADC que dispone el microcontrolador es de 10 bits, por lo que su rango de salida es de 0 a 1023 respondiendo a la siguiente ecuación:

$$N_{ADC} = 1023 \left(\frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}} \right) \tag{6.1}$$

Donde V_{in} es el voltaje de entrada, es decir el voltaje de salida del sensor, y V_{R-} y V_{R+} son voltajes de referencia fijados en $0V$ y $3{,}3V$ respectivamente.

Registrando varias muestras y tomando un promedio de ellas, se obtiene los resultados de la tabla 6.1.

Distancia real (cm)	Salida ADC media
11.8	871
21.8	749
31.8	566
41.8	432
51.8	351
61.8	296
71.8	252
81.8	223
91.8	196
101.8	175
111.8	158
121.8	144
131.8	133
141.8	119
151.8	113
161.8	105

Tabla 6.1: Distancia real contra salida ADC.

Utilizando la ecuación 6.1 podemos obtener una medida indirecta del voltaje de salida y comparar su funcionamiento con lo esperado según la hoja del fabricante.

El resultado puede observarse en la figura 6.5, donde los puntos marcados son la conversión de los valores experimentales y la curva responde a una ecuación de ajuste de los mismos. Puede verse la clara similitud respecto a los datos proporcionados por el fabricante en la figura 6.2, tanto en la forma de la curva como en sus valores máximos y mínimos.

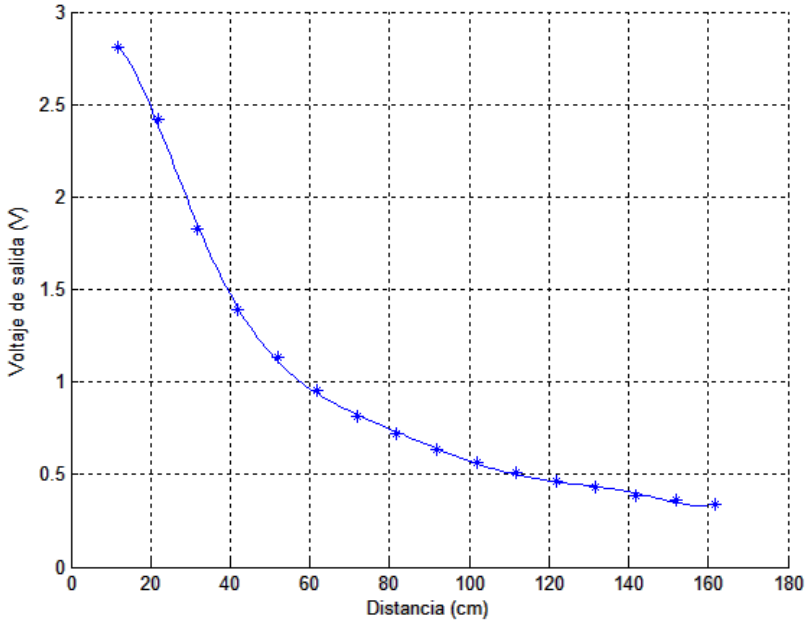


Figura 6.5: Voltaje en función de distancia.

Por otro lado, es necesario obtener una expresión analítica que nos permita transformar la salida del ADC en una medida de distancia. Para ello se utilizó el método de mínimos cuadrados, consiguiendo el siguiente polinomio de tercer grado:

$$d(adc) = -1{,}0724 \times 10^{-6} adc^3 + 0{,}0019 adc^2 - 1{,}1086 adc + 2{,}4928 \qquad (6.2)$$

Donde d representa la distancia que se esta midiendo y adc es la salida del conversor.

La figura 6.6 muestra como dicha curva se ajusta a los valores experimentales. En ella, puede verse un comportamiento poco deseado en la parte final de la curva, perdiendo validez el uso de dicha expresión para mediciones inferiores a los $20cm$. Sin embargo, el fabricante declara que el rango confiable de mediciones comienza justamente en los $20cm$, por lo que no preocupa este comportamiento.

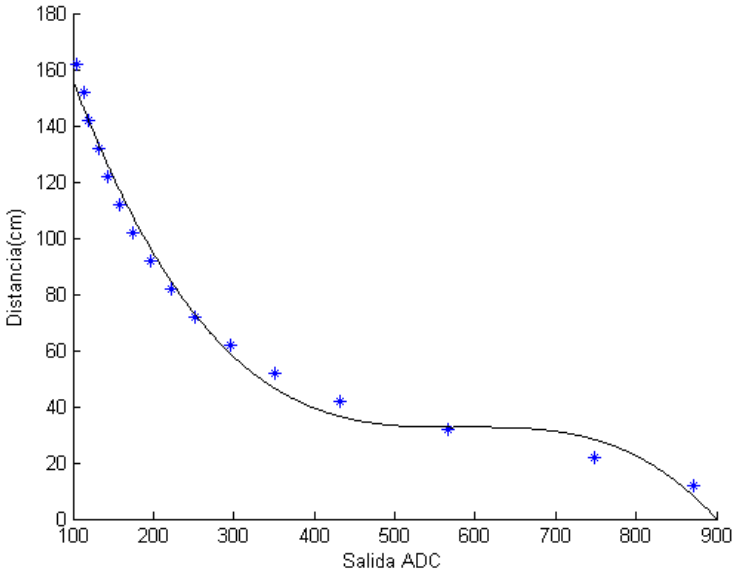


Figura 6.6: Distancia en función de la salida del ADC.

Utilizando dicha expresión se registraron mediciones para distintas distancias, tomando aproximadamente 200 muestras consecutivas y promediandolas. Adicionalmente, para cada set de muestras se registraron las medidas máximas y mínimas y se calculó el error y la desviación estándar.

Medida real (cm)	Promedio (cm)	Error (cm)	Máximo (cm)	Mínimo (cm)	Desviación estándar (cm)
26.8	23.7	3.1	25	21	0.6
46.8	46.2	0.6	48	37	1.3
61.8	62.8	-1.0	67	55	1.9
81.8	80.9	0.9	88	64	2.9
96.8	95.1	1.7	103	76	4.1
116.8	113.5	3.3	126	87	6.4
131.8	127.3	4.5	143	104	6.2
151.8	144.0	7.8	167	123	7.9

Tabla 6.2: Tabla de datos calculados a partir de varias medidas.

Observando los resultados queda claro cómo varía la performance del sensor según la distancia que se este midiendo. Puede verse como el rango de mayor confiabilidad se encuentra entre los $30cm$ y los $100cm$, donde no se registraron errores mayores a los $1{,}7cm$, más que suficiente para la aplicación en cuestión. Incluso el máximo error de $7{,}8cm$ resulta aceptable considerando que se planea volar a más de un metro de altura y los obstáculos deben ser detectados con mayor anticipación.

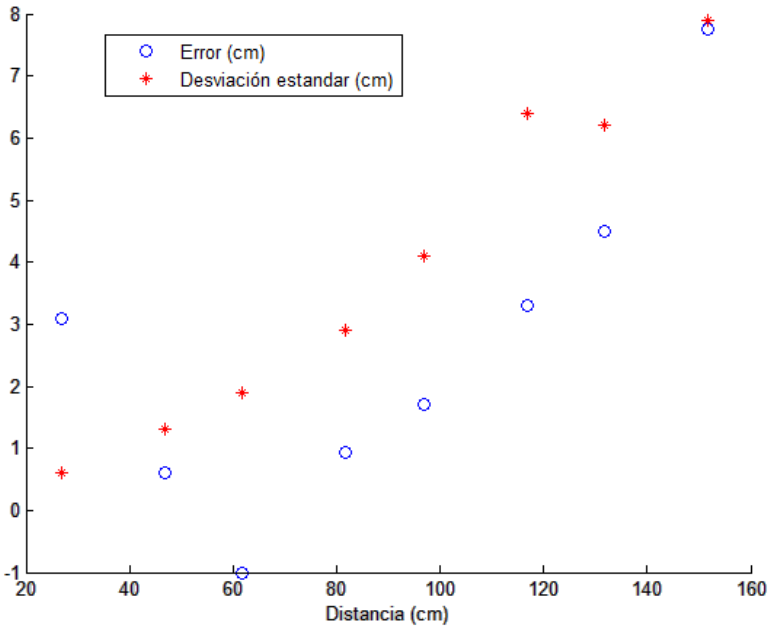


Figura 6.7: Error y desviación estándar.

Sin embargo, debe tenerse en cuenta que para distancias mayores a los $100cm$ es necesario tomar la media de una serie de valores, ya que la precisión de las medidas comienza a empeorar significativamente.

6.3. Sensor de ultrasonido

6.3.1. Materiales

Para estudiar el sensor ultrasónico se utilizaron los siguientes materiales:

- Sensor de ultrasonido *MaxBotix MB1340*
- Placa de desarrollo *Arduino Mega*
- Computadora
- Cinta métrica
- Tabla ancha de madera (obstáculo)

Figura 6.8: Sensor de ultrasonido.

Según la hoja de datos del sensor utilizado, el mismo es capaz de detectar un objeto si este se encuentra entre $1mm$ y $765cm$ pero puede estimar la distancia dentro del rango de $20cm$ a $765cm$ con $1cm$ de resolución. Permite ser alimentada con un voltaje entre $3{,}3V$ y $5V$, con un consumo promedio que varía entre $2{,}1mA$ y $3{,}4mA$ según el voltaje utilizado. Son tres las salidas que tiene disponible: dos análogas y una serial. Una de las señales análogas es proporcional a la distancia a medir, mientras que la segunda entrega la envolvente de onda de la señal. La salida serial es en formato RS232 pero con voltajes comprendidos entre $0V$ y V_{cc}.

La *Arduino Mega* es una placa de desarrollo basada en el microcontrolador *ATmega1280* de $16MHz$ con 128Kbytes de memoria flash, $8KBytes$ de RAM y $4KBytes$ de EEPROM. Al igual que la *Arduino Due*, presenta un conversor ADC con la única diferencia de utilizar un voltaje de referencia de $5V$, lo que permite compatibilidad directa con el sensor.

La tabla utilizada como obstáculo es de madera y de $67cm$ de ancho y $180Xcm$ de alto.

6.3.2. Procedimiento

Dado el amplio rango de funcionamiento y ángulo de apertura del sensor resulta difícil realizar mediciones en condiciones de laboratorio, por lo que se decidió realizar una primera experiencia tomando mediciones hasta $2m$ de distancia y luego verificar los resultados comparando con distancias mayores.

Se colocó el sensor de forma horizontal y la tabla de madera de frente auspiciando de obstáculo. De esta forma se registró la salida del ADC para una serie de muestras, que promediandolas se compararon con las distancias reales medidas con la cinta métrica. A partir de estos datos experimentales se halló una ecuación de ajuste que permite calcular la distancia deseada en función de la salida del ADC.

Una vez realizada dicha experiencia en un rango corto de distancias, se extendieron los resultados tomando mediciones hasta observar que el sensor deja de detectar el obstáculo. De esta forma puede estimarse el rango de detección para un obstáculo con las dimensiones utilizadas y verificarse si el ajuste realizado es válido para todo el rango de uso.

Las conexiones realizadas para las pruebas son análogas a las hechas para estudiar el sensor infrarrojo.

6.3.3. Resultados

Según la hoja de datos del sensor de ultrasonido, el voltaje de su salida es proporcional a la distancia mediante la relación $V_{out} = V_{cc}d/1023$, donde d es la distancia en cm y V_{cc} es el voltaje de alimentación del sensor que en este caso es $5V$. Esto implica que si se utiliza un voltaje de referencia para el ADC igual a V_{cc}, entonces su salida es directamente la distancia dada en cm, por lo que la lectura podría ser directa en un caso ideal.

La tabla 6.3 muestra los resultados para una serie de distancias donde para cada caso se promediaron 200 muestras aproximadamente.

Distancia real (cm)	Salida ADC media
16.3	20
36.3	34
56.3	53
76.3	73
96.3	93
116.3	114
136.3	134
156.3	154
176.3	174
196.3	194

Tabla 6.3: Distancia real contra salida ADC ($20cm$ - $200cm$).

Puede observarse como para medidas menores al rango de funcionamiento especificado por el fabricante la salida obtenida es justamente $20cm$, el mínimo valor del rango. Luego, para medidas mayores se observa que los resultados son coherentes con lo esperado, obteniendo un error máximo de $3,4cm$.

Realizando un ajuste por mínimos cuadrados se halla una expresión analítica para poder comparar los resultados experimentales con lo esperado. Para realizar dicho ajuste se descartó la primer medida realizada ya que la misma cae fuera del rango de funcionamiento normal del sensor. La figura 6.9 muestra los resultados experimentales en azul y la recta de ajuste hallada en negro.

Puede verse claramente que el polinomio hallado se ajusta perfectamente a los datos obtenidos. Idealmente la expresión de dicho polinomio debería ser de primer grado con coeficiente unitario y termino independiente nulo. Sin embargo se observa la presencia de un offset en la medida dado el resultado del ajuste realizado:

$$d = 0{,}992267322215695 \times adc + 3{,}512281041482682 \tag{6.3}$$

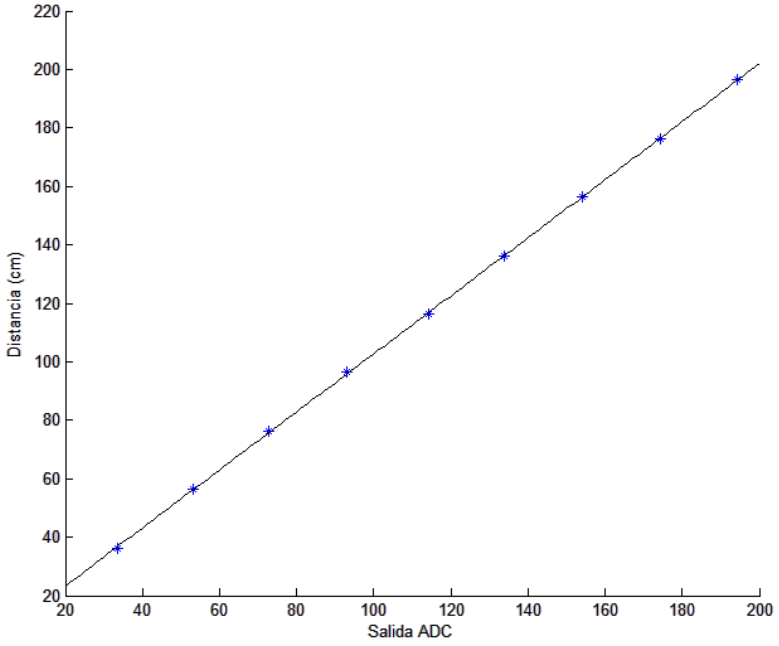


Figura 6.9: Distancia en función de la salida del ADC ($20cm$ - $200cm$).

Para corroborar estos resultados primero se realizaron nuevas mediciones compensando el offset hallado pero dentro del mismo rango de distancias. Luego de una serie de 200 medidas para cada caso se obtuvieron los resultados expuestos en la tabla 6.4.

Distancia (cm)	Media (cm)	Error (cm)	Máximo (cm)	Mínimo (cm)	Desviación estándar (cm)
26.3	27.5	-1.2	29.5	25.5	0.3
46.3	46.5	-0.2	48.5	45.5	0.2
66.3	66.5	-0.2	67.5	65.5	0.3
86.3	86.5	-0.2	96.5	85.5	0.8
106.3	106.6	-0.3	114.5	105.5	0.7
126.3	126.6	-0.3	127.5	125.5	0.4
146.3	147.4	-1.1	151.5	146.5	0.6
166.3	168.4	-2.1	170.5	166.5	0.6
186.3	187.7	-1.4	194.5	186.5	0.8

Tabla 6.4: Tabla de datos calculados a partir de varias medidas.

Los resultados obtenidos son más que satisfactorios, donde el máximo error encontrado es de tan sólo 2,1cm. Además se aprecia una buena precisión del sensor, donde la máxima separación encontrada entre medidas máximas y mínimas es de 11cm.

En la figura 6.10 se observa gráficamente los resultados de error y desviación estándar calculados para cada serie de medidas. Es destacable la

uniformidad que presenta la desviación estándar de las medidas dentro del rango de estudio.

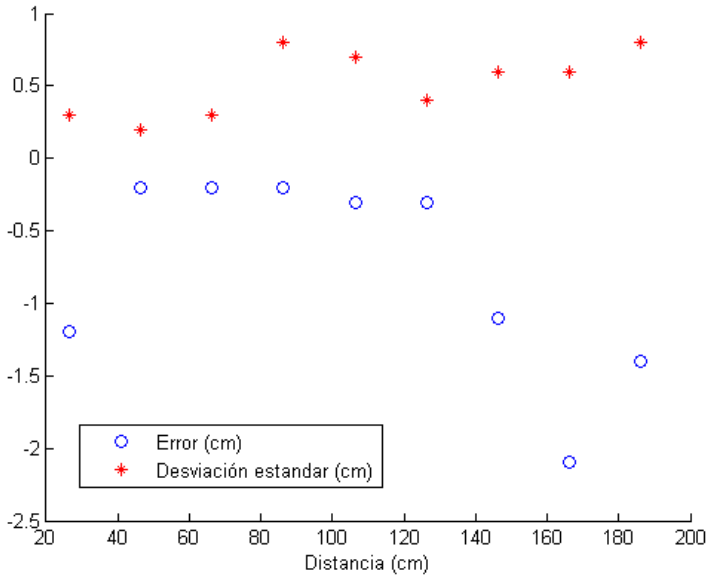


Figura 6.10: Error y desviación estándar ($20cm$ - $200cm$).

Luego de estudiadas las medidas realizadas en el primer rango, se procedió a verificar si los resultados son útiles para distancias mayores a los $2m$.

La máxima distancia posible de medir según el fabricante es de $765cm$, aunque dependiendo las dimensiones del obstáculo puede dejar de detectarse antes de alcanzar dicha lejanía. Por esta razone es necesario tener presente que se utilizó un obstáculo de $67cm$ de ancho.

Se procedió entonces a realizar medidas desde los $200cm$ hasta alcanzar la máxima lejanía de detección, que en estas condiciones se encontró a los $600cm$ aproximadamente. Los resultados se observan en la tabla 6.5 donde las medidas obtenidas ya incluyen el offset hallado anteriormente (ecuación 6.3).

Puede verse que la última medida ya se encuentra en el límite de reconocimiento del obstáculo, ya que muchas medidas caen dentro de un rango aceptable mientras que muchas otras se encuentran en el orden de los $700cm$ alcanzando un máximo de $757{,}5cm$ indicando la ausencia de un objeto.

Si se filtran las muestras obtenidas quitando todas aquellas mayores a los $700cm$ el resultado cambia notoriamente, obteniendo resultados similares al resto de los casos.

Puede observarse en estas medidas la presencia de un offset algo mayor al hallado previamente, ya que prácticamente en todos los casos se obtuvieron

Medida (cm)	Promedio (cm)	Error (cm)	Máximo (cm)	Mínimo (cm)	Desviación estándar (cm)
192	188.1	3.9	194.5	186.5	0.9
212	214.2	-2.2	219.5	211.5	0.9
232	235.3	-3.3	240.5	233.5	0.9
252	254.0	-2.0	265.5	251.5	1.1
272	275.3	-3.3	279.5	273.5	0.8
292	294.3	-2.3	301.5	292.5	1.0
312	319.2	-7.2	321.5	317.5	0.7
332	341.6	-9.6	345.5	339.5	1.0
352	361.2	-9.2	363.5	358.5	0.8
372	379.3	-7.3	388.5	377.5	1.0
392	402.8	-10.8	404.5	400.5	0.7
412	423.4	-11.4	425.5	421.5	0.8
432	438.7	-6.7	448.5	436.5	1.1
452	462.3	-10.3	474.5	460.5	1.2
472	479.7	-7.7	481.5	478.5	0.7
492	498.9	-6.9	503.5	496.5	1.0
512	520.0	-8.0	521.5	517.5	1.1
532	541.1	-9.1	542.5	539.5	0.7
552	564.1	-12.1	568.5	562.5	1.3
572	581.5	-9.5	596.5	579.5	1.4
604	630.7	-26.7	757.5	611.5	44.5

Tabla 6.5: Datos relevados para distancias mayores a los $2m$.

Medida (cm)	Promedio (cm)	Error (cm)	Máximo (cm)	Mínimo (cm)	Desviación estándar (cm)
604	610.6	-6.6	621	608	1.5

Tabla 6.6: Medida filtrada.

resultados mayores al valor medido con la cinta métrica. Sin embargo, debe tenerse en cuenta que el error propio de la medición con cinta métrica comienza a ser mayor para distancias mayores. Por otro lado, cuanto mayor sea la distancia a medir, menos critico resulta obtener una medida con algunos centímetros de error. Por estas razones se decidió mantener el offset hallado para distancias cortas.

Es notorio también que la precisión de las medidas se mantiene respecto a lo estudiado previamente, obteniendo en este caso una variación máxima de $14cm$ para una misma distancia medida. La desviación estándar sigue estando muy cerca de los valores calculados para las primeras medidas.

6.4. Comparación de ambos sensores

Luego de estudiar las características de cada sensor, se realizó una simple comparación entre ellos para decidir cuál se adapta mejor a cada necesidad, tanto para detectar obstáculos como para obtener una estimación de la altura de vuelo.

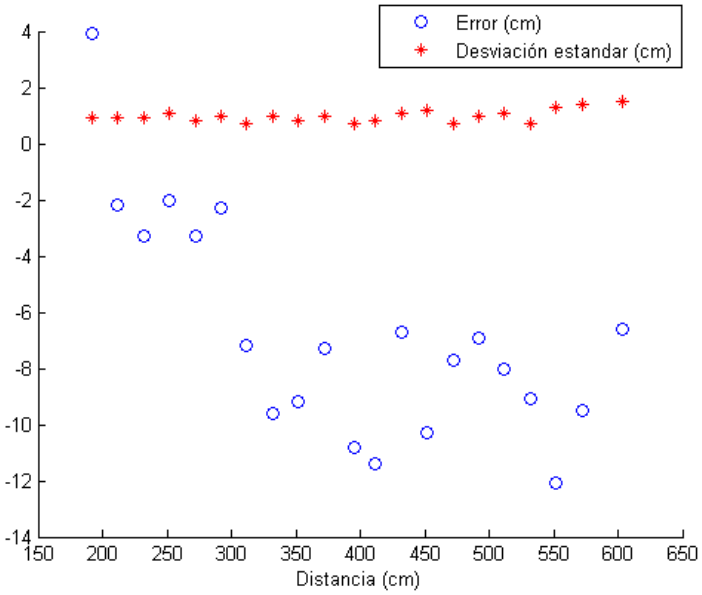


Figura 6.11: Error y desviación estándar ($200cm$ - $600cm$).

Sin duda la mayor diferencia entre estos sensores es el rango de utilización. El sensor infrarrojo tiene un rango demasiado pequeño impidiendo directamente su uso para obtener la altura de vuelo. El sensor ultrasónico asegura obtener medidas con exactitudes aceptables al enfrentarse a un plano de $67cm$ de ancho y $180cm$ de alto, por lo que se cree posible la obtención de medidas de altura de vuelo hasta los $7m$ de altura aproximadamente.

Por otro lado el sensor infrarrojo es más veloz debido la tecnología que utiliza pero es necesario hacer una conversión no lineal para convertir su salida a datos útiles. El sensor de ultrasonido es más lento, alcanzando una frecuencia de $10Hz$ según los datos proporcionados por el fabricante, pero su conversión no implica más que agregar un offset a su salida por lo que casi no requiere recursos más que el conversor ADC.

Desde el punto de vista de precisión de cada uno, sin dudas resulta mucho más confiable el sensor de ultrasonido. Eso puede verse claramente en la desviación estándar de las medidas obtenidas y en los máximos y mínimos registrados para cada caso. Utilizar el sensor infrarrojo impone la necesidad de promediar una serie de medidas antes de obtener un dato útil, mientras que el con el ultrasónico no resulta tan crítico.

Algo destacable del sensor infrarrojo es su baja influencia por factores externos tales como temperatura, humedad, etc. Esto lo hace ideal para el uso en el exterior como es el caso, y más aún teniendo en cuenta que se encontrará en condiciones de grandes vibraciones y expuesto a factores ambientales. Por su parte el sensor de ultrasonido es mucho más sensible a ciertos factores

como las corrientes de aire que provocarán las hélices del cuadricóptero [11]
.

Además el sensor infrarrojo es mucho más direccional permitiendo ser más específico en la posición donde se encuentra el obstáculo. Sin embargo sería necesario utilizar varios sensores para abarcar todo el rango posible de colisiones.

6.5. Conclusiones

Luego de haber considerado ambas opciones, queda claro que el sensor de ultrasonido es el sensor indicado para la medición de la altura de vuelo en despegue y aterrizaje. También se concluye que es preferible utilizar el ultrasonido para detectar obstáculos. Según los resultados obtenidos es de esperar obtener medidas con exactitud y precisión más que aceptables dentro de todo el rango de medida. Además no se aprecia gran variación en este aspecto para distintas distancias medidas.

Capítulo 7

Caracterización del control de actitud de la *CC3D*

En el siguiente capítulo se caracteriza la respuesta del sistema físico controlado por la placa de control de actitud *CC3D*. Se releva información importante para el posterior diseño de los algoritmos y controladores que permiten el vuelo autónomo.

7.1. Caracterización de la respuesta en *Throttle*

Throttle es el nombre que recibe la señal PWM de entrada a la placa *CC3D*, con la cual se regula la suma de las velocidades de los cuatro motores y por consiguiente la fuerza de empuje ejercida por los mismos. En el capítulo 4 se caracteriza la respuesta del sistema de propulsión respecto a la señal PWM recibida por los ESC. Resta determinar la correlación entre la señal PWM de *Throttle* ingresada a la *CC3D* y la señal PWM que ésta envía a los ESC.

7.1.1. Objetivos

Medir la relación entre la señal PWM en la entrada *Thorottle* de la CC3D y la salida PWM_{ESC} a los ESC.

7.1.2. Materiales

- *Beagleboard*.
- *CC3D*.
- Osciloscopio *Tektronix TBS 1062*.

7.1.3. Procedimiento

Se deshabilita el control de actitud en la *CC3D* para lograr que las señales de salida a los motores dependan únicamente del *Throttle* deseado. Luego, se programa la *Beagleboard* para que genere señales PWM de *Throttle* en un rango que va de $1100\mu s$ a $1900\mu s$, recorrido de a pasos de $100\mu s$. Para cada señal generada por la *Beagleboard*, se mide el comando PWM de salida de la *CC3D* utilizando un osciloscopio.

7.1.4. Resultados

Como se aprecia en la figura 7.1, la relación entre ambas señales PWM es lineal según la ecuación 7.1, cuyos coeficientes fueron obtenidos mediante mínimos cuadrados.

$$Throttle = 1{,}4PWM_{ESC} - 606{,}5 \tag{7.1}$$

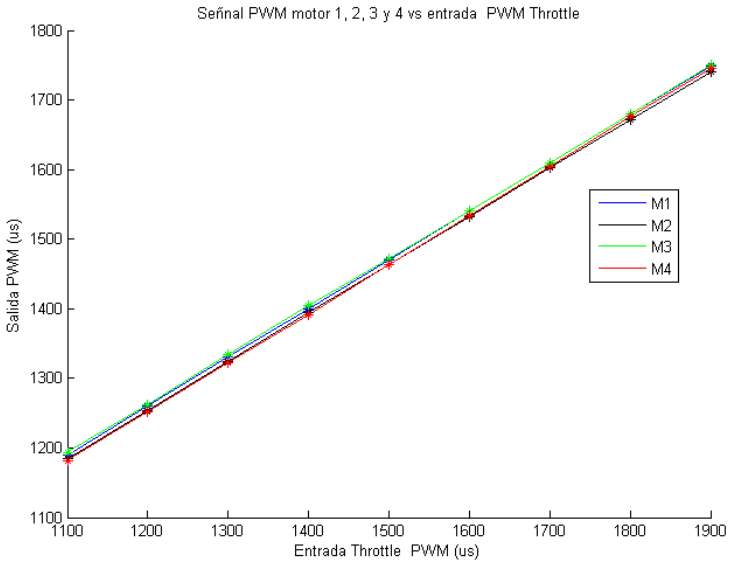


Figura 7.1: Relación entre señal de entrada *Throttle* y salida PWM hacia los motores.

7.1.5. Conclusiones

Se caracteriza satisfactoriamente el comportamiento entre ambas señales, lo que permite comunicar a los motores el empuje deseado a través de la *CC3D*.

7.2. Ángulo *Pitch*

La proyección de la fuerza de empuje sobre el plano está directamente relacionada con el ángulo *Pitch* siempre que se imponga *Roll* nulo. Para diseñar el controlador de velocidad, es necesario caracterizar la forma en que la *CC3D* controla dicho ángulo. La figura 7.3 presenta un diagrama que ilustra el funcionamiento del sistema a estudiar.

7.2.1. Objetivos

- Caracterizar la respuesta estática, es decir, determinar qué angulo se logra en régimen al ingresar determinada señal PWM a la entrada *Pitch* de la *CC3D*. Esto determina la ganancia en continua.

- Caracterizar la respuesta escalón para el ángulo *Pitch* del sistema formado por el cuadricóptero y la *CC3D*. Esto determina la dinámica de la respuesta.

7.2.2. Materiales

- *Beagleboard*.

- Cuadricóptero controlado por la *CC3D*.

- *Canetígiro*[1]

.

7.2.3. Procedimiento

Se cuelga el cuadricóptero equipado con la *CC3D* y la *Beagleboard* haciendo uso del *Canetígiro*, ver figura 7.2.

Para relevar la curva que relaciona la señal PWM enviada y el ángulo obtenido se envía una serie de señales PWM, partiendo del valor correspondiente al ángulo cero e incrementando en pasos de $25\mu s$. La respuesta en *Pitch* para cada entrada es sensada por la *CC3D* cada $50ms$ y los datos almacenados en un *log* por la *Beagleboard*. Luego se relacionan entrada y salida analizando los datos en *MatLab*.

La caracterización de la dinámica de la respuesta se realiza inyectando escalones de 5°, 10° y 20° en la entrada *Pitch* de la *CC3D*.

[1]Instrumento fabricado por el Ing. Rafael Canetti que dota al cuadricóptero de dos grados de libertad: giro respecto al eje z y giro respecto a y_B.

Figura 7.2: Cuadricóptero acoplado al *Canetígiro*.

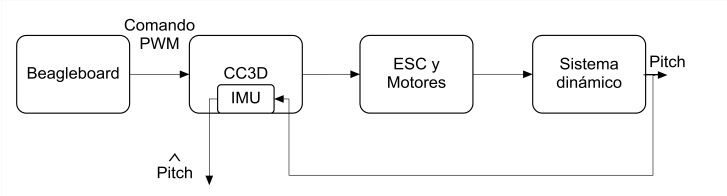


Figura 7.3: Diagrama de funcionamiento.

7.3. Resultados

Señal *PWM* - Ángulo *Pitch* obtenido

La figura 7.4a muestra la medida del ángulo *Pitch* frente a la variación en la entrada PWM. El ancho de los pulsos ingresados se detallan en la tabla 7.1. El ángulo obtenido para cada comando se estima promediando las muestras luego del transitorio.

La relación que se cumple entre la señal PWM de entrada y el ángulo obtenido resulta lineal, tal como se aprecia en la gráfica de la figura 7.4b. Dicha relación queda expresada en la ecuación 7.2

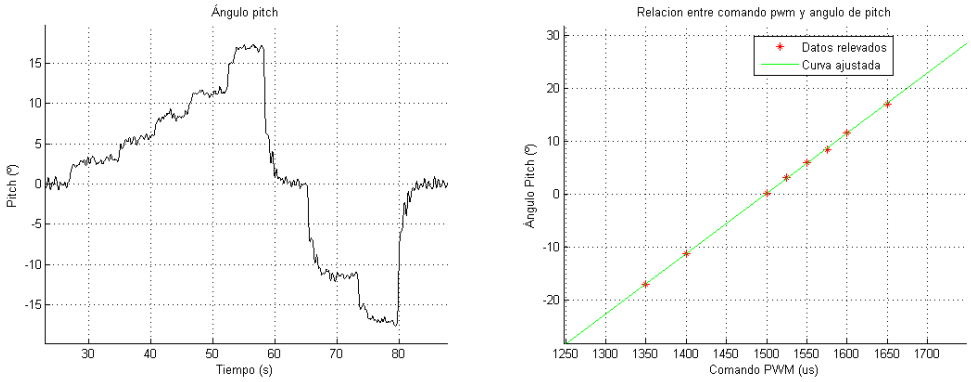
$$PWM_{in} = 8{,}78Pitch + 1500 \tag{7.2}$$

Respuesta al escalón

Los resultados de los escalones de 10° y 20° se pueden ver en las gráficas de la figura 7.5. Se ensayaron dichos valores por su similitud respecto a los

Comando PWM (μs)	Ángulo *Pitch* estimado (º)
1350	-17.1444
1400	-11.3757
1500	0.0164
1525	2.9852
1550	5.8919
1575	8.3435
1600	11.4687
1650	16.9938

Tabla 7.1: Relación entre comando PWM y ángulo *Pitch* obtenido.



(a) Pitch en funcion del tiempo

(b) Relación

Figura 7.4: Relación entre comando PWM y ángulo *Pitch* obtenido.

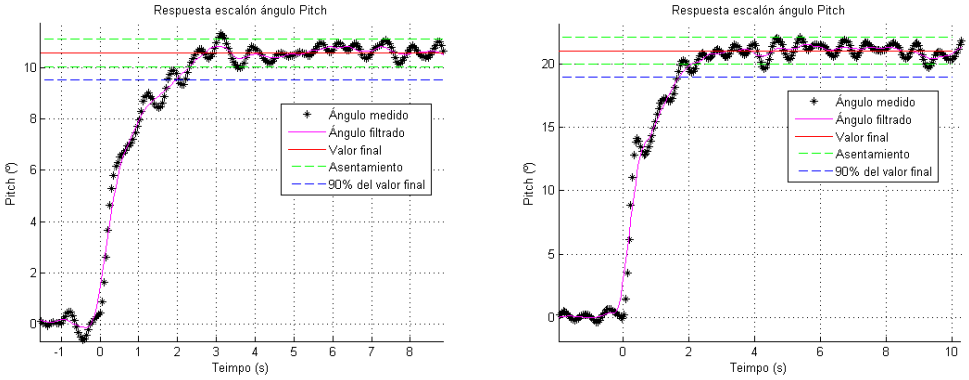
ángulos de *Pitch* utilizados durante el vuelo.

En las gráficas mostradas se observa la presencia de una oscilación cuyo periodo corresponde a 13 muestras o 0,65s aproximadamente. Éste fenómeno se le adjudica al mecanismo de colgado (*Canetígiro*), el cual presenta elasticidad y por ende una oscilación independiente a la respuesta del sistema estudiado. A los efectos de relevar los tiempos característicos de respuesta, se filtran las medidas aplicando un promedio de 13 muestras correspondiente a un período de la oscilación.

En la tabla 7.2 se muestra la caracterización de la respuesta temporal para los escalones relevados.

7.3.1. Conclusiones

Se releva satisfactoriamente la relación entre el comando recibido por la *CC3D* y el ángulo efectivamente logrado por el cuadricóptero, la relación resulta lineal.



(a) Escalón de 10° (b) Escalón de 20°

Figura 7.5: Respuestas escalón ángulo *Pitch*.

Dimensión del escalón (°)	5	10	20	Promedio	Desviación estándar
t_s (s)	2.57	2.31	2.09	2.32	0.24
$t_{0-90\,\%}$ (s)	2.79	1.95	1.71	2.15	0.57
$\tau \approx t_s/3$(s)	0.85	0.77	0.70	0.77	0.08

Tabla 7.2: Respuesta escalón *Pitch*.

Se caracteriza la respuesta temporal del cuadricóptero controlado por la *CC3D*, concluyendo que se puede aproximar mediante un sistema de primer orden.

7.4. Ángulo *Roll*

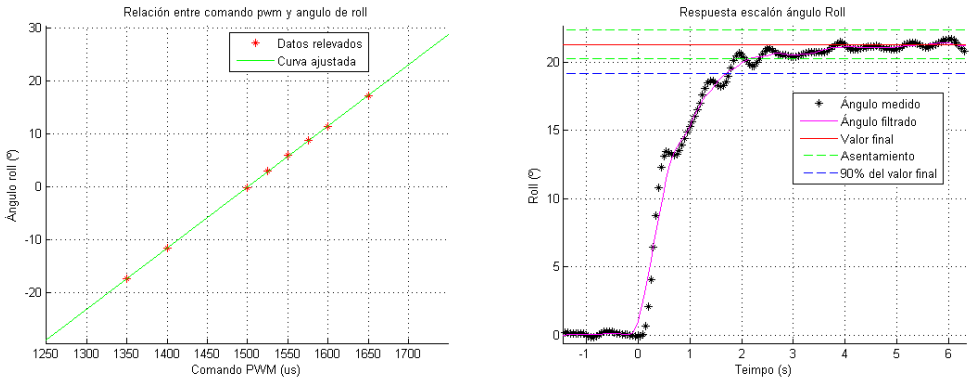
Se espera que la respuesta del cuadricóptero controlado por la *CC3D*, según el ángulo *Roll*, sea muy similar al caso del *Pitch*. La toma de medidas en ambos experimentos se realizó de forma análoga.

7.4.1. Resultados

Señal *PWM* - Ángulo *Roll* obtenido

En la figura 7.6a se aprecia la relación lineal que cumplen el comando PWM y la respuesta del sistema en ángulo *Roll*, relacionados según ecuación 7.3.

$$PWM_{in} = 8{,}67Roll + 1500 \tag{7.3}$$



(a) Comando PWM vs *Roll* (b) Respuesta escalón

Figura 7.6: Respuesta del ángulo *Roll*.

Respuesta al escalón

Como se puede ver en la figura 7.6b, se somete el sistema a escalones de 5°, 10° y 20°. A partir de dicho experimento se obtiene la respuesta temporal que se observa en la tabla 7.3.

Dimensión del escalón (°)	5	10	20	Promedio	Desviación estándar
t_s (s)	2.34	1.87	2.15	2.12	0.23
$t_{0-90\,\%}$ (s)	1.87	1.56	1.67	1.70	0.16
$\tau \approx t_s/3$(s)	0.78	0.62	0.72	0.71	0.08

Tabla 7.3: Respuesta escalón *Roll*.

7.4.2. Conclusiones

Los resultados son similares al caso del ángulo *Pitch*. Dicho comportamiento era el esperado considerando que la dinámica y el control para ambos casos es casi idéntico.

7.5. Caracterización de la respuesta en Yaw

El control ejercido por la *CC3D* para el ángulo *Yaw* actúa únicamente sobre la velocidad angular, es decir, se controla el $\dot{Yaw}$ en lugar del Yaw. Como se ve más adelante, el seguimiento de trayectorias depende de una acción de control que establezca el ángulo *Yaw* requerido, por lo que el control de la *CC3D* por sí solo resulta insuficiente. Por esta razón se caracteriza experimentalmente la respuesta de velocidad de giro según *Yaw* del sistema

cuadricóptero-*CC3D*, obteniendo la información necesaria para el diseño del controlador de Yaw desarrollado en el capítulo 13.

7.5.1. Objetivos

Analizar la respuesta transitoria del control de velocidad de giro según *Yaw* efectuado por la *CC3D* sobre el cuadricóptero.

7.5.2. Materiales

- *Beagleboard*.
- Cuadricóptero controlado por *CC3D*.
- *MatLab*.

7.5.3. Procedimiento

Se cuelga el cuadricóptero de forma tal de restringir el movimiento en todos los ejes exceptuando el giro en torno al eje vertical. Se ingresan escalones en la entrada *Yaw*[2] de la *CC3D*. La magnitud de los escalones es de $40°/s$ y $120°/s$.

Se programa en la *Beagleboard* las rutinas que se encargan tanto de enviar las señales escalón mediante una señal PWM como de recibir y almacenar los datos sensados por la IMU integrada en la *CC3D*.

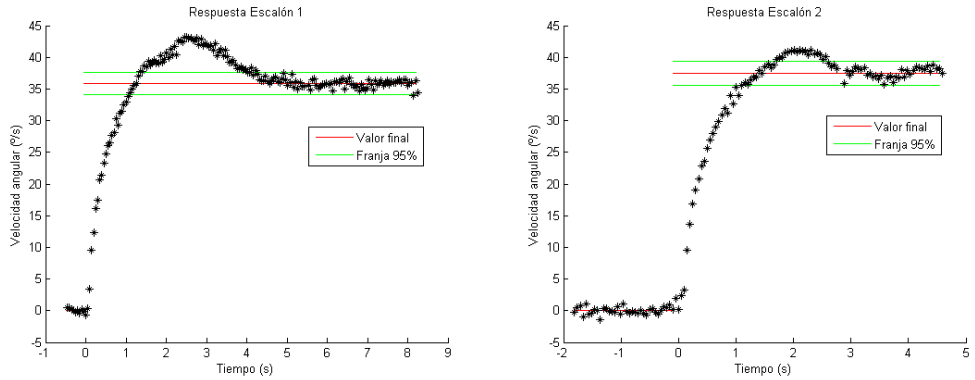
7.5.4. Resultados

Si bien las respuestas a los escalones presentan un sobretiro promedio de 15 % (ver figuras 7.7 y 7.8), se realiza una aproximación a un sistema de primer orden. Éste modelo, cuya transferencia en *Laplace* se aprecia en la ecuación 7.4, resulta suficiente para el desarrollo del control de posición angular, tal como se comprueba en el capítulo 13.

Se promedian los resultados de los cuatro escalones (ver tabla 7.4) para determinar un estimativo de la constante de tiempo del sistema: $T_s = 0{,}41s$.

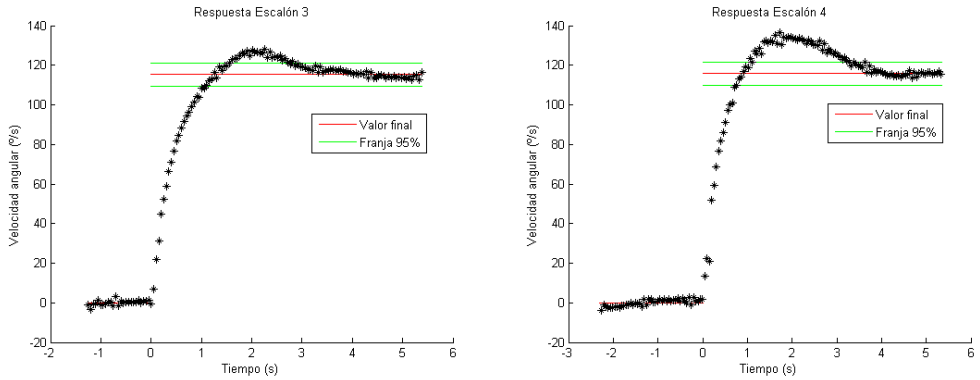
$$H(s) = \frac{1}{1 + T_s s} \tag{7.4}$$

[2]La señal de entrada *Yaw* de la *CC3D* recibe velocidades angulares a ser logradas



(a) Escalón 1 (b) Escalón 2

Figura 7.7: Respuestas escalón de $40°/s$.



(a) Escalón 1 (b) Escalón 2

Figura 7.8: Respuestas escalón de $120°/s$.

	Escalón 1	Escalón 2	Escalón 3	Escalón 4	Promedio	Desv. estándar
Magnitud del escalón (°/s)	40	40	120	120		
t_{0-100} (s)	1.25	1.51	1.26	0.91	1.23	0.25
$t_s(s)$	4.22	2.60	2.87	3.28	3.24	0.71
$t_p(s)$	2.46	2.11	2.26	1.72	2.14	0.31
$t_{63,2\%}(s)$	0.45	0.43	0.42	0.34	0.41	0.05
$M_p(\%)$	20.40	9.73	11.04	18.10	14.82	5.23

Tabla 7.4: Constantes de la respuesta transitoria.

7.5.5. Conclusiones

El control de velocidad angular de *Yaw* resulta en un sistema con sobretiro que claramente es de orden superior a uno. Se realiza una aproximación a un sistema de primer orden para simplificar el desarrollo del controlador de posición angular, sin perder información del retardo y la ganancia en continua

de la planta.

Parte III

Algoritmos de autonomía

Capítulo 8

Generación de trayectorias

Un aspecto central en la autonomía de un cuadricóptero es lograr que el mismo sea capaz de generar una trayectoria a seguir para cumplir con determinado objetivo de vuelo. Se debe contar con la capacidad de diseñar el recorrido que satisfaga las consignas indicadas por el usuario.

La autonomía que se pretende dar al vehículo permitiría, por ejemplo, recorrer un perímetro con motivos de vigilancia o realizar cierto tipo de mediciones en determinados puntos predefinidos. Una forma de lograr esto es implementar un algoritmo que calcule un camino óptimo cumpliendo los requisitos impuestos por el usuario.

8.1. Objetivo

El objetivo específico es diseñar un algoritmo que a partir de una serie de *waypoints* ingresados por el usuario y el ángulo acimutal con el cual éstos deben ser atravesados, genere una trayectoria que recorra cada punto en la dirección indicada.

Los *waypoints* quedan determinados por las coordenadas cartesianas (x, y, z) relativas al sistema de referencia inercial S_I, originado en la posición inicial del vehículo. La dirección con la que se debe abarcar cada *waypoint* se define según el ángulo acimutal, es decir, referido al punto cardinal Norte en el sentido de las agujas del reloj.

8.1.1. Restricciones

Al momento de desarrollar el algoritmo se manejaba una hipótesis heredada del proyecto uQuad! [16] donde se restringían los posibles movimientos del vehículo únicamente a rectas y circunferencias. Si bien esto no es un impedimento para el modelo físico utilizado en el presente proyecto, se mantiene dicha restricción a la hora de generar las trayectorias. Sin embargo, para el

seguimiento de las mismas se pueden realizar otro tipo de maniobras (esto se explica en detalle en el capítulo 9).

Las trayectorias generadas están compuestas por la unión de rectas y círculos de radio r constante. Esto restringe los posibles caminos entre dos *waypoints* consecutivos dados sus respectivos ángulos de salida y llegada.

Existe una restricción en cuanto al radio mínimo de las circunferencias. Si bien los cuadricópteros gozan de una gran maniobrabilidad, el error presente en la estimación de posición impone límites más restrictivos. Dadas las características del GPS a utilizar (estudiadas en el capítulo 5) se decide trabajar con un radio de curvatura mínimo de $10m$.

Finalmente, queda restringir la distancia mínima entre dos *waypoints* consecutivos, que si bien limita el espectro de aplicaciones posibles, es necesario debido nuevamente al sistema de posicionamiento GPS. Se considera que para obtener buenos resultados durante el vuelo los *waypoints* deben distar al menos $50m$ entre sí.

8.2. Desarrollo del algoritmo

Una vez establecidos los *waypoints* por donde debe pasar el vehículo y sus orientaciones, es necesario diseñar una curva apropiada que los una. Se consideraron ciertas simplificaciones que permiten descomponer el problema en partes, estas son:

- Reducir el problema a dos *waypoints* únicamente, luego pueden concatenarse varias trayectorias para lograr el objetivo final.
- Suponer un entorno libre de obstáculos, la evasión de los mismos puede atacarse independientemente generando nuevas trayectorias luego de la detección.
- Reducir el problema al plano horizontal tomando *waypoints* de dimensión dos (x, y). La altura es controlada de forma independiente por el controlador de altitud, ver capítulo 11.

8.2.1. Curvas de Dubins

La teoría desarrollada por Lester Eli Dubins en 1957 [9] es directamente aplicable a la situación de interés del presente trabajo e incluso ya ha sido utilizada en aplicaciones similares como en [22] y [21]. En dicha teoría se muestra que dados dos puntos del plano (x_1, y_1) y (x_2, y_2); con una orientación asociada a cada uno de ellos θ_1 y θ_2, y dado un radio mínimo de curvatura del vehículo r, entonces existe una curva continuamente diferenciable de distancia mínima que une ambos puntos en las direcciones adecuadas.

Esta curva puede ser de la forma CSC o CCC, donde S representa una trayectoria recta y C un arco de circunferencia de radio r. Este último puede existir en dos variantes según el sentido en el que se lo recorra, denominándolo R o L si es recorrido en sentido horario o antihorario respectivamente.

De esta forma se deduce que para cada par de *waypoints* existen seis posibles configuraciones para determinar la trayectoria de largo mínimo que los una:

- RSR
- RSL
- LSL
- LSR
- LRL
- RLR

Este tipo de curvas cumplen con las restricciones planteadas, ya que limitan las posibles sub-trayectorias únicamente a rectas y arcos de circunferencia. También cumplen el objetivo presentado, uniendo *waypoints* que contemplan la posición angular de partida y llegada y minimizando el largo de la trayectoria.

Utilizando las ideas de Dubins, una solución al problema sería considerar todas las posibles configuraciones para cada caso concreto y finalmente seleccionar la que minimiza el largo total de la trayectoria. Sin embargo, Andrei M. Shkel y Vladimir J. Lumelsky en [18] logran reducir los casos de prueba utilizando la información de los ángulos de partida y llegada. En su trabajo también se demuestra que bajo ciertas condiciones[1] la solución que minimiza el largo de la trayectoria siempre es del tipo CSC, descartando en todos los casos las curvas del tipo CCC. Esto reduce las opciones a las curvas RSR, LSL, LSR y RSL.

Aplicar estas técnicas permite utilizar el algoritmo en forma realmente eficiente, ya que al disminuir la cantidad de configuraciones posibles se reducen las operaciones necesarias y por lo tanto los costos computacionales. Incluso para algunos casos alcanza con conocer los ángulos de partida y llegada para determinar la curva de Dubins correcta.

En el desarrollo del algoritmo se normalizan las distancias tomando al radio r como unidad. La distancia entre *waypoints* resulta entonces $d = D/r$, donde D es la distancia real.

[1] La distancia entre *waypoints* debe ser mayor a cuatro veces el radio de curvatura para asegurar el correcto funcionamiento en todos los casos. Esta condición se cumple siempre para los valores elegidos.

Para cada sub-trayectoria R, L y S existe una transformación que mapea su origen y ángulo inicial con sus coordenadas finales y ángulo de llegada, a partir de la información del largo recorrido. Estas se definen como $S_v(x, y, \theta)$, $L_v(x, y, \theta)$ y $R_v(x, y, \theta)$, donde v representa el largo de la sub-trayectoria.

$$\begin{aligned} S_v(x, y, \theta) &= (x + v\cos(\theta), y + v\sin(\theta), \theta) \\ L_v(x, y, \theta) &= (x + \sin(\theta + v) - \sin(\theta), y - \cos(\theta + v) + \cos(\theta), \theta + v) \\ R_v(x, y, \theta) &= (x - \sin(\theta - v) + \sin(\theta), y + \cos(\theta - v) - \cos(\theta), \theta - v) \end{aligned} \quad (8.1)$$

Si se conocen los largos de todas las sub-trayectorias, concatenando estas transformaciones se obtiene el *waypoint* de llegada para cualquier combinación CSC. Sin embargo, este no es el caso ya que se conocen los *waypoints* y sus orientaciones pero no las distancias a recorrer.

Con el fin de simplificar las operaciones se considera un nuevo sistema de coordenadas cartesianas, tal que su origen coincida con el *waypoint* de partida y el *waypoint* de llegada tenga coordenadas $(d,\ 0)$. Esto no es más que una traslación y rotación respecto al sistema de referencia inercial S_I, el resultado se puede ver en la figura 8.1.

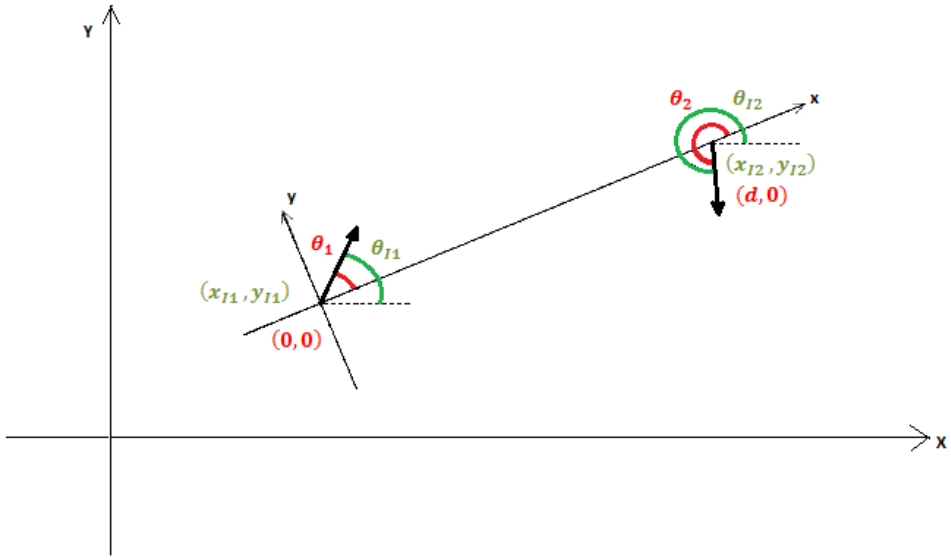


Figura 8.1: Sistema de coordenadas utilizado.

A modo de ejemplo se supone una configuración RSL. Concatenando las transformaciones correspondientes se obtiene la relación entre el *waypoint* inicial $(0, 0, \theta_1)$ y el final $(d, 0, \theta_2)$:

$$L_q(S_p(R_t(0, 0, \theta_1))) = (d, 0, \theta_2) \quad (8.2)$$

De dicha relación se desprende el sistema de ecuaciones 8.3, donde t, p y q representan los largos de las sub-trayectorias R, S y L respectivamente.

$$\begin{cases} p\cos(\theta_1 - t) - 2\sin(\theta_1 - t) + \sin(\theta_1) + \sin(\theta_2) = d \\ p\sin(\theta_1 - t) + 2\cos(\theta_1 - t) - \cos(\theta_1) - \cos(\theta_2) = 0 \\ \theta_1 - t + q = \theta_2 \end{cases} \tag{8.3}$$

Resolviendo el sistema se obtienen los largos de todas las sub-trayectorias que componen la trayectoria RSL en función de d, θ_1 y θ_2.

RSL:

$$\begin{cases} t_{rsl} = \theta_1 - \arctan\left(\dfrac{\cos(\theta_1) + \cos(\theta_2)}{d - \sin(\theta_1) - \sin(\theta_2)}\right) + mod2\pi\left(\arctan\left(\dfrac{2}{p_{rsl}}\right)\right) \\ p_{rsl} = \sqrt{d^2 - 2 + 2\cos(\theta_1 - \theta_2) - 2d(\sin(\theta_1) + \sin(\theta_2))} \\ q_{rsl} = \theta_2 - \arctan\left(\dfrac{\cos(\theta_1) + \cos(\theta_2)}{d - \sin(\theta_1) - \sin(\theta_2)}\right) + mod2\pi\left(\arctan\left(\dfrac{2}{p_{rsl}}\right)\right) \end{cases} \tag{8.4}$$

Análogamente se obtienen las soluciones para el resto de las configuraciones LSL, RSR y LSR.

LSL:

$$\begin{cases} t_{lsl} = -\theta_1 + mod2\pi\left(\arctan\left(\dfrac{\cos(\theta_2) - \cos(\theta_1)}{d + \sin(\theta_1) - \sin(\theta_2)}\right)\right) \\ p_{lsl} = \sqrt{d^2 + 2 - 2\cos(\theta_1 - \theta_2) + 2d(\sin(\theta_1) - \sin(\theta_2))} \\ q_{lsl} = \theta_2 - mod2\pi\left(\arctan\left(\dfrac{\cos(\theta_2) - \cos(\theta_1)}{d + \sin(\theta_1) - \sin(\theta_2)}\right)\right) \end{cases} \tag{8.5}$$

RSR:

$$\begin{cases} t_{rsr} = \theta_1 - mod2\pi\left(\arctan\left(\dfrac{\cos(\theta_1) - \cos(\theta_2)}{d - \sin(\theta_1) + \sin(\theta_2)}\right)\right) \\ p_{rsr} = \sqrt{d^2 + 2 - 2\cos(\theta_1 - \theta_2) + 2d(\sin(\theta_2) - \sin(\theta_1))} \\ q_{rsr} = -\theta_2 + mod2\pi\left(\arctan\left(\dfrac{\cos(\theta_1) - \cos(\theta_2)}{d - \sin(\theta_1) + \sin(\theta_2)}\right)\right) \end{cases} \tag{8.6}$$

LSR:

$$\begin{cases} t_{lsr} = mod2\pi\left(-\theta_1 + \arctan\left(\dfrac{-\cos(\theta_1)-\cos(\theta_2)}{d+\sin(\theta_1)+\sin(\theta_2)}\right) - \arctan\left(\dfrac{-2}{p_{lsr}}\right)\right) \\ p_{lsr} = \sqrt{d^2 - 2 + 2\cos(\theta_1-\theta_2) + 2d(\sin(\theta_1)+\sin(\theta_2))} \\ q_{lsr} = -\theta_2 + \arctan\left(\dfrac{-\cos(\theta_1)-\cos(\theta_2)}{d+\sin(\theta_1)+\sin(\theta_2)}\right) - mod2\pi\left(\arctan\left(\dfrac{-2}{p_{lsr}}\right)\right) \end{cases} \tag{8.7}$$

Sumando los largos de las sub-trayectorias se obtiene el largo de la trayectoria total, por lo que calculando todos los casos se puede decidir cual minimiza su largo. Sin embargo, se utilizó la clasificación realizada en [18], cuyo resultado se expone a continuación.

Observando en que cuadrantes se encuentran los ángulos de partida y llegada respecto al nuevo sistemas de coordenadas, se reducen las posibles trayectorias según se observa en la tabla 8.1, donde se muestra para cada caso las condiciones que determinan el tipo de curva deseada.

θ_1 \ θ_1	**1**	**2**	**3**	**4**
1	**RSL**	**si** $S_{12} < 0 \Rightarrow$ **RSR** **si** $S_{12} > 0 \Rightarrow$ **RSL**	**si** $S_{13} < 0 \Rightarrow$ **RSR** **si** $S_{13} > 0 \Rightarrow$ **LSR**	**si** $S_{14}^1 > 0 \Rightarrow$ **LSR** **si** $S_{14}^2 > 0 \Rightarrow$ **RSL** **sino RSR**
2	**si** $S_{21} < 0 \Rightarrow$ **LSL** **si** $S_{21} > 0 \Rightarrow$ **RSL**	**si** $S_{22}^1 < 0 \Rightarrow$ **LSL** **si** $S_{22}^1 > 0 \Rightarrow$ **RSL** **si** $S_{22}^2 < 0 \Rightarrow$ **RSR** **si** $S_{22}^2 > 0 \Rightarrow$ **RSL**	**RSR**	**si** $S_{24} < 0 \Rightarrow$ **RSR** **si** $S_{24} > 0 \Rightarrow$ **RSL**
3	**si** $S_{31} < 0 \Rightarrow$ **LSL** **si** $S_{31} > 0 \Rightarrow$ **LSR**	**LSL**	**si** $S_{33}^1 < 0 \Rightarrow$ **RSR** **si** $S_{33}^1 > 0 \Rightarrow$ **LSR** **si** $S_{33}^2 < 0 \Rightarrow$ **LSL** **si** $S_{33}^2 > 0 \Rightarrow$ **LSR**	**si** $S_{34} < 0 \Rightarrow$ **RSR** **si** $S_{34} > 0 \Rightarrow$ **LSR**
4	**si** $S_{41}^1 > 0 \Rightarrow$ **RSL** **si** $S_{41}^2 > 0 \Rightarrow$ **LSR** **sino LSL**	**si** $S_{42} < 0 \Rightarrow$ **LSL** **si** $S_{42} > 0 \Rightarrow$ **RSL**	**si** $S_{43} < 0 \Rightarrow$ **LSL** **si** $S_{43} > 0 \Rightarrow$ **LSR**	**LSR**

Tabla 8.1: Tabla de decisión de tipo de curva según Shkel et al. [18].

Como puede observarse, existen cuatro casos en los que el tipo de curva de Dubins queda directamente determinado, mientras que en los doce casos restantes solo hace falta ejecutar a lo sumo dos operaciones y verificar cuál es la condición que se cumple dentro del cuadro. Las funciones S_{ij} y S_{ij}^k, que condicionan el resultado según su signo, se definen a partir de dos funciones $f(a_1, a_2, a_3)$ y $g(a)$, cuyos parámetros de entrada son los largos de las sub-trayectorias.

Las funciones $f(a_1, a_2, a_3)$, $g(a)$, S_{ij} y S_{ij}^k son [18]:

$$
\begin{aligned}
f(a_1, a_2, a_3) &= a_1 - a_2 - 2(a_3 - \pi) \\
g(a) &= a - \pi
\end{aligned}
\tag{8.8}
$$

$$
\begin{aligned}
S_{12} &= f(p_{rsr}, p_{rsl}, q_{rsl}) \\
S_{13} &= g(t_{rsr}) \\
S^1_{14} &= g(t_{rsr}) \\
S^2_{14} &= g(q_{rsr}) \\
S_{21} &= f(p_{lsl}, p_{rsl}, t_{rsl}) \\
Si\ \alpha > \beta \Rightarrow S^1_{22} &= f(p_{lsl}, p_{rsl}, t_{rsl}) \\
Si\ \alpha < \beta \Rightarrow S^2_{22} &= f(p_{rsr}, p_{rsl}, q_{rsl}) \\
S_{24} &= g(q_{rsr}) \\
S_{31} &= g(q_{lsl}) \\
Si\ \alpha < \beta \Rightarrow S^1_{33} &= f(p_{rsr}, p_{lsr}, t_{lsr}) \\
Si\ \alpha > \beta \Rightarrow S^2_{33} &= f(p_{lsl}, p_{lsr}, q_{lsr}) \\
S_{34} &= f(p_{rsr}, p_{lsr}, t_{lsr}) \\
S^1_{41} &= g(t_{lsl}) \\
S^2_{41} &= g(q_{lsl}) \\
S_{42} &= g(t_{lsl}) \\
S_{43} &= f(p_{lsl}, p_{lsr}, q_{lsr})
\end{aligned}
\tag{8.9}
$$

Una vez determinado el tipo de curva que minimiza el largo total de la trayectoria y fijado el radio mínimo de curvatura del cuadricóptero, queda totalmente definida la trayectoria dentro del plano (x, y), dado que ya se conocen los largos de cada sub-trayectoria y todos los sentidos de giro.

Al ser el radio de curvatura utilizado en el desarrollo anterior unitario, los largos de las trayectorias circulares representan directamente el ángulo barrido durante las mismas. Luego, para obtener los largos reales de cada sub-trayectoria se debe multiplicar el valor obtenido por el radio r.

8.2.2. Extensión de curvas a tres dimensiones

Una de las simplificaciones realizadas fue reducir el problema al plano (x, y), pero es necesario extender la solución al espacio tridimensional. Una opción es la adoptada por Lin y Saripalli en [13], esta consiste en establecer una velocidad constante según el eje z asegurando que se llega a la altura deseada al mismo tiempo en que se alcanza la posición (x, y) objetivo. La

posición vertical de cada punto de la trayectoria queda terminada según la ecuación 8.10, donde z_i y z_f son las alturas de los *waypoints* iniciales y finales respectivamente, L es el largo de la trayectoria entre ambos *waypoints* y (x, y) es el punto de la trayectoria en el plano que se corresponde con el z que se está calculando.

$$z = z_i + \frac{l(x,y)}{L}(z_f - z_i) \tag{8.10}$$

8.3. Resumen

A continuación se resume el algoritmo desarrollado que permite definir una trayectoria a seguir dados un *waypoint* inicial (x_1, y_1, z_1) y un *waypoint* final (x_2, y_2, z_2), junto a sus respectivas orientaciones θ_{I1} y θ_{I2}.

1. Tomar los datos de cada *waypoint* (x, y) y θ y redefinirlos en el nuevo sistema de coordenadas. Como resultado se obtiene el *waypoint* inicial en $(0, 0)$ y el final en $(d, 0)$, donde d es la distancia entre ambos *waypoints* normalizada respecto al radio de curvatura r. Las nuevas orientaciones θ_1 y θ_2 resultan de la diferencia entre los ángulos θ_{I1} y θ_{I2} y la rotación del nuevo sistema de coordenadas respecto al original (ver figura 8.1).

2. Determinar la clase de curva de Dubins deseada en función los cuadrantes a los que pertenecen θ_1 y θ_2.

3. En función de la clase determinada, la información del cuadrante correspondiente de la tabla 8.1 y las ecuaciones 8.9 determinar el tipo de curva de Dubins.

4. Según la curva determinada utilizar las ecuaciones 8.4, 8.5, 8.6 o 8.7 para hallar los largos normalizados de las sub-trayectorias t_{ijk}, p_{ijk} y q_{ijk}.

5. Multiplicar los valores hallados por el radio de curvatura r. Esto da como resultado los largos reales de las sub-trayectorias.

6. Teniendo el tipo de curva, los largos de las trayectorias y los *waypoints* con sus respectivas orientaciones $(x_1,\ y_1,\ z_1,\ \theta_{I1})$ y $(x_2,\ y_2,\ z_2,\ \theta_{I2})$ se tiene completo conocimiento de la trayectoria hallada en el plano.

7. Utilizar la ecuación 8.10 y las alturas dadas dadas por cada *waypoint* para extender el resultado anterior al espacio tridimensional.

8.4. Conclusiones

Con el algoritmo presentado es posible dotar de autonomía de vuelo al cuadricóptero, ya que le permite autogenerar trayectorias que recorran una serie de *waypoints* orientados en el espacio. Además, dichas trayectorias son curvas continuamente diferenciables y de largo mínimo.

El algoritmo contempla las restricciones dadas, limitando las posibles trayectorias a líneas rectas y arcos de circunferencia. Este asegura su funcionamiento siempre y cuando se cumpla la condición $C_{il} \cup C_{ir} \cap C_{fl} \cup C_{fr} = \emptyset$, donde los C_{ij} representan las posibles circunferencias iniciales y finales. Esto implica que la distancia mínima en el peor caso es de $4r$, restricción que se cumple para los valores elegidos.

Se logra cumplir con el objetivo de vuelo planteado, ya que el algoritmo permite especificar tanto la posición tridimensional (x, y, z) como la orientación para cada *waypoint*. Además pueden concatenarse varias trayectorias generadas de forma independiente asegurando tanto la continuidad como la diferenciabilidad de la curva total, ampliando las posibilidades de uso.

El algoritmo es sencillo y logra generar la curva deseada en pocos pasos, lo que permite una rápida ejecución. Si bien la trayectoria podría generarse antes de comenzar el vuelo restándole importancia a este factor, resulta útil en caso de necesitar regenerar la trayectoria, ya sea por haberse alejado demasiado de la original o por encontrarse con un obstáculo.

Capítulo 9

Seguimiento de trayectorias

Una vez que se cuenta con una trayectoria a recorrer, es necesario implementar un algoritmo de seguimiento. Esto implica obtener una altura y ángulo *Yaw* deseado a partir de la estimación del estado actual y de la trayectoria a seguir.

La figura 9.1 resume cómo el algoritmo de seguimiento de trayectorias interacciona con el resto del sistema.

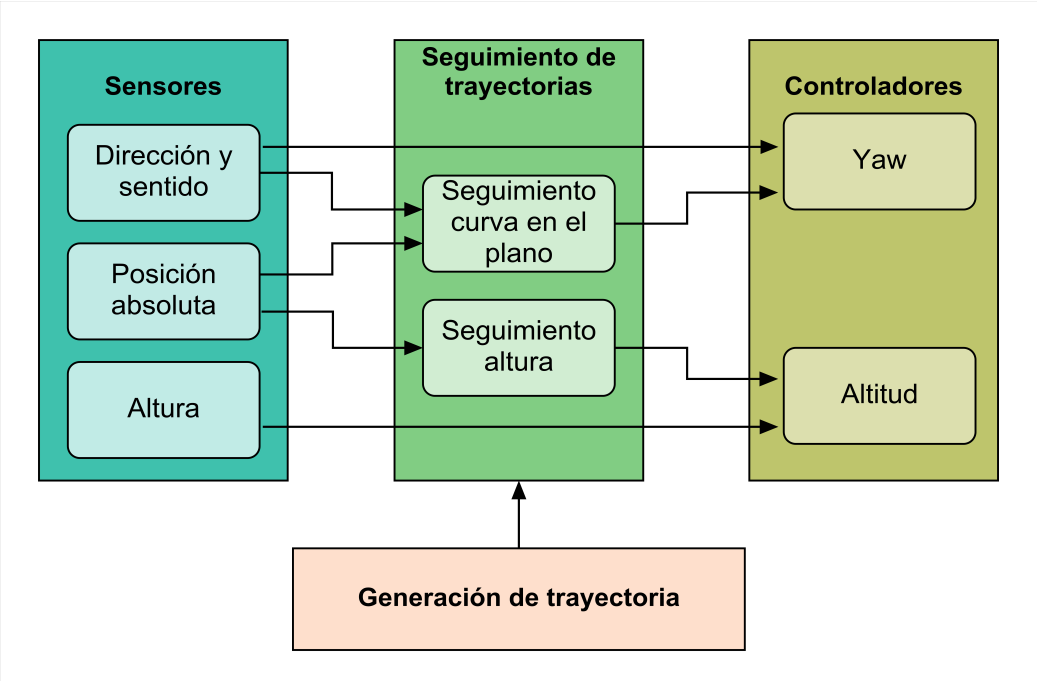


Figura 9.1: Diagrama del proceso de seguimiento de trayectorias.

9.1. Seguimiento de posición en el plano

Dado que el ángulo *Roll* se considera nulo durante todo el recorrido, el cambio en la dirección de vuelo estará influenciado únicamente por el estado del ángulo *Yaw*. Por lo tanto, la estrategia de seguimiento de la curva plana generada utiliza como variable de control al ángulo *Yaw*. A partir de la trayectoria a seguir y conociendo la posición en un instante dado, puede calcularse el *Yaw* deseado que permita acercarse a la trayectoria.

En primera instancia se opta por discretizar la trayectoria generada en *sub-waypoints* y seguir cada uno de ellos en forma consecutiva. El ángulo *Yaw* deseado se calcula entonces como el ángulo entre el eje x del S_I y la recta definida entre la posición actual y el *sub-waypoint* que se esta siguiendo. A su vez, el algoritmo debe verificar si alcanzó el *sub-waypoint* que estaba siguiendo para evaluar si debe cambiar al objetivo siguiente o permanecer en el actual.

Se presenta el psudocódigo que implementa dicho algoritmo, donde *swp* es el *sub-waypoint* a seguir y p es la posición del cuadricóptero dada por el GPS. Se utiliza un valor *UMBRAL* que representa la distancia máxima para la cual se considera alcanzado el *sub-waypoint* objetivo.

$d = \|p - swp\|$;
if $d <$ *UMBRAL* **then**
  $swp =$ *Proximo sub-waypoint*;
end
Yaw deseado $= atan2(p, swp)$;

Algorithm 1: Pseudocódigo de la primera aproximación al algoritmo de seguimiento.

La información que el algoritmo necesita es una lista ordenada de *sub-waypoints* obtenidos a partir de la discretización de la trayectoria. El intervalo entre muestras, es decir la distancia entre dos way-points consecutivos, es un parámetro a elección determinante en la eficiencia del algoritmo. Esto introduce un posible problema con el algoritmo tal como está planteado.

Supongamos que el cuadricóptero se encuentra realizando un vuelo siguiendo determinado *sub-waypoint*. Podría suceder que ante determinada acción de control el cuadricóptero se pase del *sub-waypoint* que estaba siguiendo sin cumplir la condición umbral, provocando que los próximos ciclos de control exijan volver hacia atrás en lugar de seguir adelante. Por esta razón se siguió la idea presentada en [17], donde conociendo la trayectoria que se desea seguir se fijan los *sub-waypoints* de forma dinámica en cada loop de ejecución. De esta forma se evita retroceder en el seguimiento de la trayectoria.

9.1.1. Seguimiento de trayectorias rectilineas

La estrategia para ubicar dinámicamente el *sub-waypoint* a seguir consiste en determinar la proyección ortogonal de la posición actual sobre la trayectoria y fijar el *sub-waypoint* determinada distancia más adelante.

La figura 9.2 muestra gráficamente la estrategia de posicionamiento del *sub-waypoint*. La recta que se desea seguir esta determinada por los puntos W_i y W_{i+1}, p indica la posición medida por el GPS, q es la proyección de p sobre la recta y s es el *sub-waypoint* ubicado una distancia δ por delante.

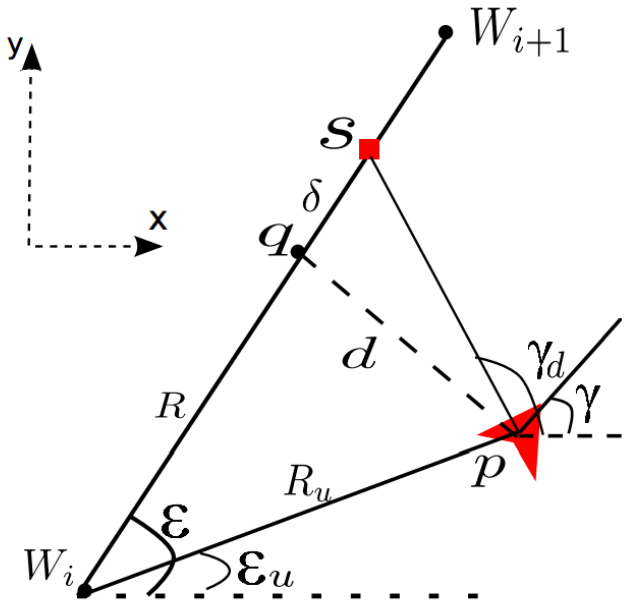


Figura 9.2: Seguimiento trayectorias rectilineas según [17].

Entonces, para ejecutar el algoritmo es necesario conocer los dos puntos que definen la recta y tener una medida nueva del GPS. Con esa información se calcula el *sub-waypoint* según la ecuación 9.1.

$$\begin{aligned} R_u &= \|W_i - p\| \\ \varepsilon_u &= atan2(W_i, p) \\ \varepsilon &= atan2(Wi, W_{i+1}) \\ R &= \sqrt{R_u^2 - (R_u sen(\varepsilon - \varepsilon_u))} \\ \gamma_d &= atan2(p, s) \\ s &= (W_{i\hat{x}} + (R+\delta)\cos(\varepsilon), W_{i\hat{y}} + (R+\delta)\sin(\varepsilon)) \end{aligned} \tag{9.1}$$

Una vez hallado el *sub-waypoint* s se calcula el *Yaw* deseado.

$$\gamma_d = atan2(p, s) \tag{9.2}$$

El parámetro δ representa qué tan adelante se desea posicionar el *sub-waypoint* respecto a la proyección q. Variaciones en este parámetro provocan variaciones en la respuesta del algoritmo, tanto en términos de tiempo de convergencia como en su respuesta oscilatoria. Para valores de δ grandes la convergencia será lenta pero no se percibirán grandes oscilaciones, mientras que valores pequeños generaran el comportamiento opuesto.

9.1.2. Seguimiento trayectorias circulares

Para el caso del seguimiento de trayectorias circulares se utiliza la misma idea, ubicando el *sub-waypoint* sobre la trayectoria en función de la proyección de la posición estimada sobre la misma. Por lo tanto, también en este caso existe un parámetro λ a ajustar, solo que este representa el ángulo $\widehat{qOs}$ (ver figura 9.3). Si se multiplica este ángulo por el radio de la circunferencia se obtiene la longitud del arco comprendido entre el *sub-waypoint* y la proyección ortogonal, lo que es equivalente al caso anterior.

La figura 9.3 muestra la trigonometría involucrada para el cálculo de s a partir de la posición p dada por el GPS y la trayectoria a seguir (definida a partir del centro O y el radio r). Adicionalmente, en este caso es necesario conocer el sentido en que se desea recorrer la circunferencia.

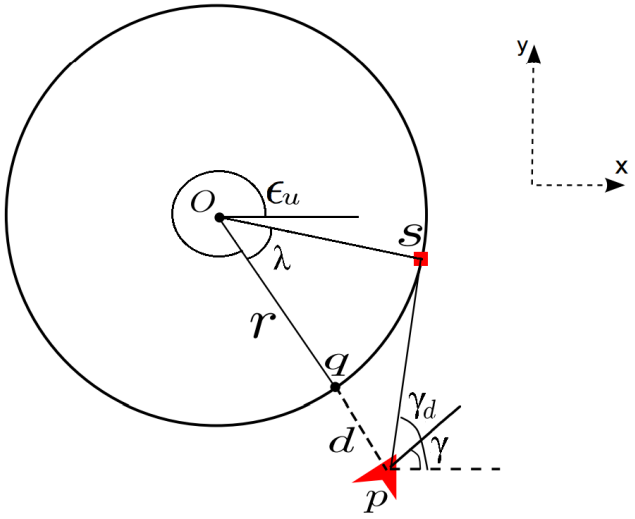


Figura 9.3: Seguimiento trayectorias circulares según [17].

Se calcula el *sub-waypoint* s según la siguiente ecuación.

$$\begin{aligned} \epsilon_u &= atan2(O, p) \\ s &= (O_{\widehat{x}} + r\cos(\epsilon_u \pm \lambda), O_{\widehat{y}} + r\sin(\epsilon_u \pm \lambda)) \end{aligned} \tag{9.3}$$

Finalmente se calcula *Yaw* deseado.

$$\gamma_d = atan2(p, s) \tag{9.4}$$

El signo de λ determina el sentido en que se recorre la circunferencia, siendo en sentido horario cuando su signo es negativo y sentido antihorario para el caso de signo positivo.

9.1.3. Diseño de los parámetros de seguimiento

Para fijar los parámetros δ y λ se optó por generar simulaciones utilizando $MatLab$, donde se ejecuta el seguimiento de ambas trayectorias variando dichos parámetros a modo de comparación. Esto también permite estudiar el comportamiento del algoritmo cambiando la condición inicial en posición, velocidad y ángulo *Yaw*.

Dado que el algoritmo se alimenta de la posición dada por del GPS, resulta necesario simular el movimiento del cuadricóptero que depende, entre otras cosas, de la posición angular dada por *Yaw*.

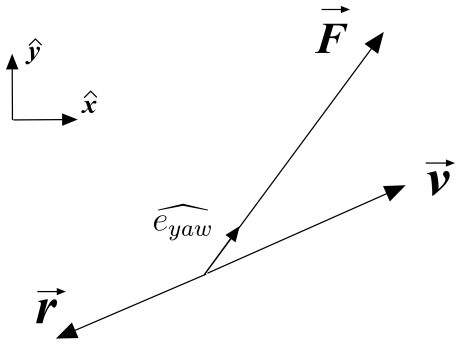


Figura 9.4: Diagrama de fuerzas.

En la figura 9.4 se observa el diagrama de fuerzas que actúan sobre el cuadricóptero proyectadas sobre el plano (x, y), donde $\vec{v}$ es velocidad instantánea del cuadricóptero, $\vec{r}$ es la fuerza de rozamiento contra el aire y $\vec{F}$ es la proyección de la fuerza ejercida por los motores/hélices a causa de la inclinación dada por el *Pitch*. Por su lado $\widehat{x}$ y $\widehat{y}$ son versores del S_I y $\widehat{e_{yaw}}$ es un versor con dirección según el ángulo *Yaw*.

Considerando entonces que la fuerza ejercida por los motores/hélices será del orden de la fuerza peso obtenemos la ecuación de movimiento 9.5, donde ψ es el ángulo *Pitch* y B es el coeficiente de rozamiento entre el cuadricóptero y el aire.

$$m\vec{a} = mg\sin(\psi)\widehat{e_{yaw}} - B\vec{v} \tag{9.5}$$

Si se considera la implementación en tiempo discreto se obtienen las siguientes ecuaciones para calcular la posición y velocidad dada una condición inicial x_0 y v_0.

$$\begin{cases} a_{\widehat{x}i} = g\sin(\psi_i)\cos(\gamma_i) - \frac{B}{m}v_{\widehat{x}i} \\ a_{\widehat{y}i} = g\sin(\psi_i)\sin(\gamma_i) - \frac{B}{m}v_{\widehat{y}i} \end{cases}$$

$$\begin{cases} v_{\widehat{x}i+1} = v_{\widehat{x}i} + a_{\widehat{x}i}T \\ v_{\widehat{y}i+1} = v_{\widehat{y}i} + a_{\widehat{y}i}T \end{cases}$$

$$\begin{cases} x_{\widehat{x}i+1} = x_{\widehat{x}i} + v_{\widehat{x}i}T \\ x_{\widehat{y}i+1} = x_{\widehat{y}i} + v_{\widehat{y}i}T \end{cases}$$

Donde T es el período de muestreo y γ_i y ψ_i son los ángulos *Yaw* y *Pitch* en el instante *i-ésimo* respectivamente.

El valor de B se estimó en $0{,}6Kg/s$ a partir de los datos utilizados en las simulaciones realizadas por De Moor et al. en [15], teniendo en cuenta las diferencias físicas existentes entre dicho trabajo y el presente.

El valor de T se fija en $0{,}1s$, ya que además de coincidir con el período de muestro real del GPS, presenta errores de integración aceptables.

Durante el vuelo normal γ_i es controlado por la $CC3D$, por lo que para simular su medida se utiliza el resultado de la respuesta al escalón estudiada en el capítulo 13. Allí se aproxima como un sistema de primer orden con constante de tiempo $0{,}41s$.

ψ es en realidad variable, pero para simplificar la simulación se fija de tal forma que en vuelo rectilíneo implique una velocidad deseada de $3m/s$ una vez alcanzado el régimen.

$$\psi = atan\left(\frac{Bv}{mg}\right) = 0{,}1075rad \tag{9.6}$$

Una vez implementada la simulación del GPS y la actualización del ángulo *Yaw* en base a los parámetros mencionados, se implementa la simulación del algoritmo de seguimiento tanto para circunferencias como para rectas. El programa se encarga de generar las trayectoria ideal y graficar sobre ellas las simulaciones de vuelo partiendo de una posición inicial seleccionable y velocidad nula.

Elección de δ

La figura 9.5 muestra como se comporta el algoritmo al variar su parámetro de seguimiento. En ella se observa que existe un compromiso entre la amplitud de oscilación y el tiempo de convergencia a la trayectoria deseada.

En la figura 9.6 se observa la distancia a la trayectoria ideal en función del tiempo para cada caso. Está claro que cuanto mayor sea el valor de δ mayor será el tiempo de levantamiento pero menor su sobretiro.

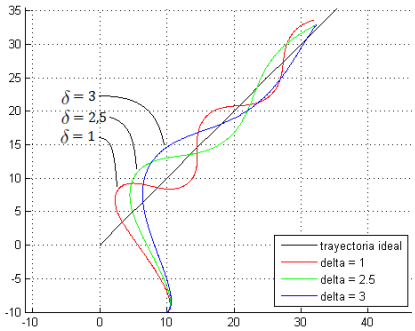


Figura 9.5: Seguimiento rectilíneo $\delta = 3{,}4m$.

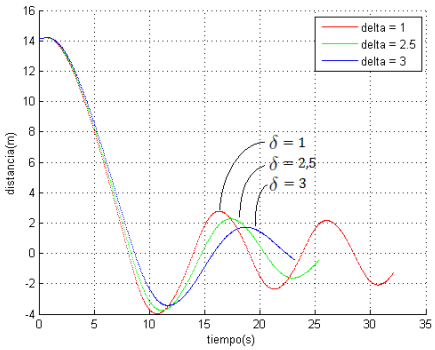


Figura 9.6: Distancia a la trayectoria ideal en función del tiempo.

Luego de realizar varias pruebas estudiando los distintos casos, se opta por fijar $\delta = 3{,}4m$. Las figuras 9.7 y 9.8 muestran la trayectoria simulada y la distancia a la trayectoria ideal respectivamente, partiendo de una posición inicial que dista $14{,}14m$ del punto de partida ideal y con ángulo de *Yaw* nulo.

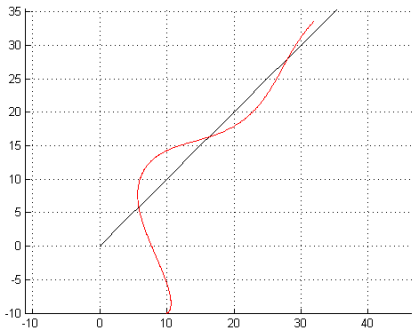


Figura 9.7: Seguimiento rectilíneo $\delta = 3{,}4m$.

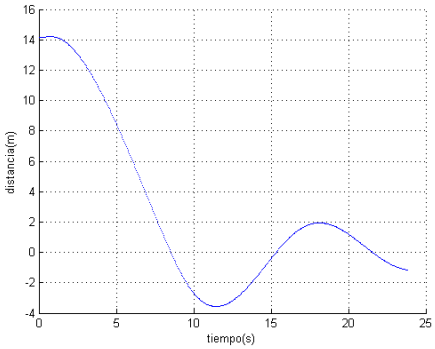


Figura 9.8: Distancia a la trayectoria ideal en función del tiempo.

Puede observarse que para dicha condición inicial se obtiene un sobretiro de aproximadamente $\%25$ y un retardo de $8s$ en alcanzar la trayectoria por primera vez. Sin embargo, hay que considerar que variaciones en la condición inicial implican cambios en la respuesta del algoritmo. A modo de ejemplo se presenta otra simulación de la misma trayectoria con los mismos parámetros de seguimiento pero partiendo de una posición inicial que dista $7{,}07m$ (ver figuras 9.9 y 9.10 respectivamente). Para esta situación se observa un sobretiro de casi el $\%40$ y un retardo de $5s$.

Finalmente se realiza una simulación partiendo de la condición inicial ideal (ver figura 9.11), donde se observa un perfecto seguimiento de la trayectoria en tal caso.

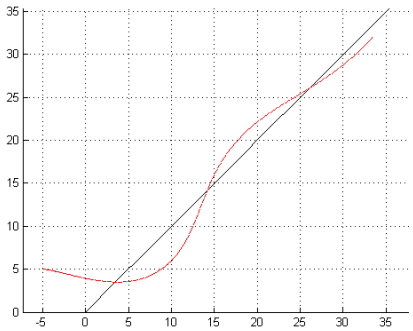


Figura 9.9: Seguimiento rectilíneo con nueva condición inicial.

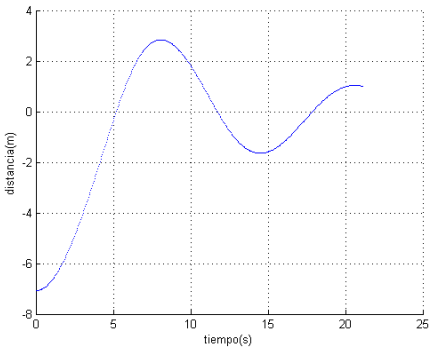


Figura 9.10: Distancia a la trayectoria ideal en función del tiempo con nueva condición inicial.

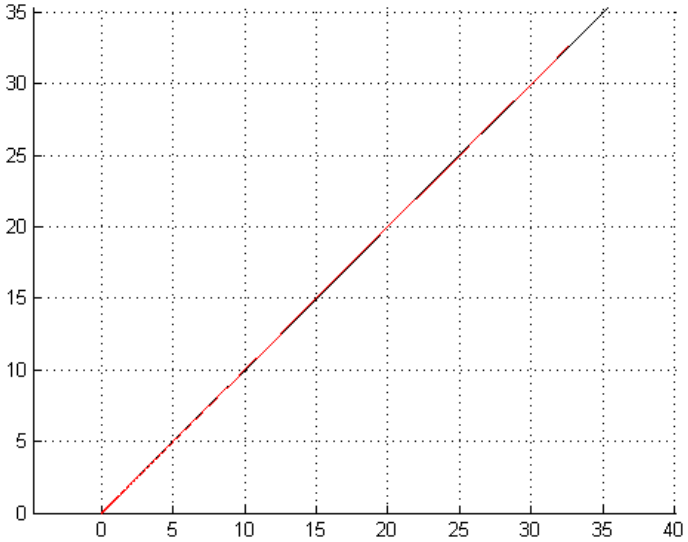


Figura 9.11: Seguimiento rectilíneo partiendo de posición inicial ideal.

Elección de λ

A diferencia del seguimiento de trayectorias rectilineas, el parámetro de diseño λ representa un ángulo, por lo que la respuesta del algoritmo es periódico entre $\lambda = 0$ y $\lambda = 2\pi$.

Para valores de λ pequeños la respuesta es oscilatoria, ya que la dirección entre la posición del UAV y el *sub-waypoint* será prácticamente ortogonal a la trayectoria. A medida que se incrementa su valor, el tiempo de convergencia comienza a acelerarse pero el radio al cual converge varía como se aprecia en las figuras 9.12 y 9.13.

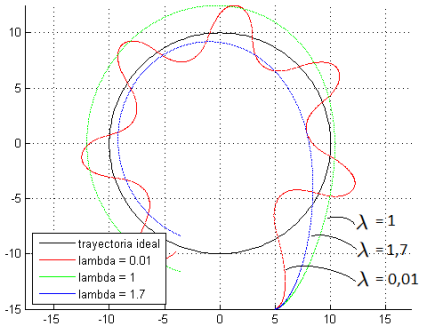


Figura 9.12: Seguimiento circular $\lambda = 1{,}4rad$.

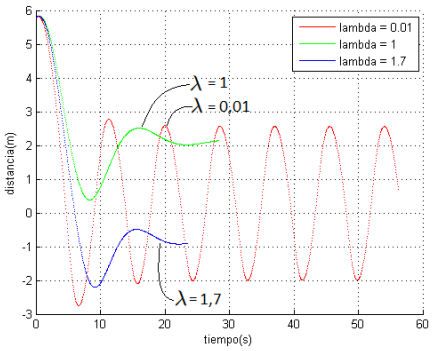


Figura 9.13: Distancia a la circular ideal en función del tiempo.

Por lo tanto, la elección de λ debe realizarse en función del radio utilizado durante la generación de trayectorias, en este caso $r = 10m$.

Realizando varias simulaciones se fija el parámetro en $\lambda = 1{,}52rad$, donde se obtiene una respuesta con error asintótico nulo respecto al radio utilizado, como se observa en las figuras 9.14 y 9.15.

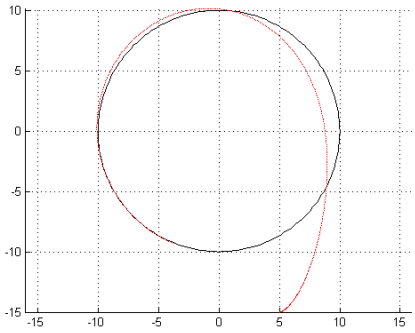


Figura 9.14: Seguimiento circular $\lambda = 1{,}4rad$.

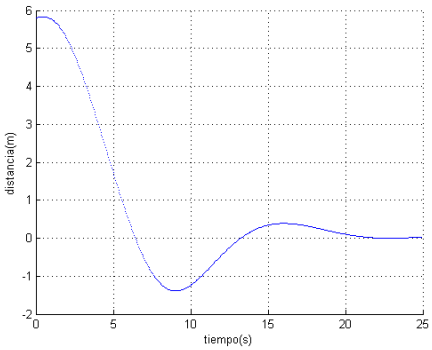


Figura 9.15: Distancia a la trayectoria ideal en función del tiempo.

9.2. Seguimiento de posición vertical

Como se explica en el capítulo 8, la trayectoria vertical es lineal entre pares de *waypoints*.

Para asegurar coordinación entre el seguimiento en el plano y el vertical y lograr que ambos alcancen sus objetivos con poca diferencia de tiempo, se utilizan los *sub-waypoints* generados extendiendolos al espacio tridimensional según la ecuación 8.10.

$$z = z_i + \frac{l(x,y)}{L}(z_f - z_i) \tag{8.10}$$

9.3. Resumen

A continuación se resume el algoritmo de seguimiento de trayectorias, el cual se compone en realidad de dos algoritmos según el tipo de curva a seguir en cada instante.

Seguimiento subtrayectoria rectiliea

Los datos necesarios para realizar el seguimiento son la posición estimada del cuadricóptero $p = (x_{quad}, y_{quad})$ y dos puntos pertenecientes a la recta a seguir $W_i = (x_i, y_i)$ y $W_f = (x_f, y_f)$.

1. Calcular la distancia el punto inicial de la sub-trayectoria y la posición actual.

$$R_u = \|W_i - p\| \tag{9.7}$$

2. Calcular los ángulos ε y ε_u.

$$\begin{aligned} \varepsilon &= atan2(W_i, W_f) \\ \varepsilon_u &= atan2(W_i, p) \end{aligned} \tag{9.8}$$

3. Calcular la distancia entre el punto inicial de la sub-trayectoria y la proyección ortogonal de la posición actual sobre la misma.

$$R = \sqrt{R_u^2 - (R_u \sin(\varepsilon - \varepsilon_u))} \tag{9.9}$$

4. Definir el *sub-waypoint* s a seguir.

$$s = (W_{i\widehat{x}} + (R + \delta)\cos(\varepsilon), W_{i\widehat{y}} + (R + \delta)\sin(\varepsilon)) \tag{9.10}$$

5. Calcular el ángulo de *Yaw* deseado como el ángulo entre el eje x del sistema de coordenadas S_I y la recta que definen los puntos p y s.

$$\gamma_d = atan2(p, s) \tag{9.11}$$

Seguimiento subtrayectoria circular

Para el caso del seguimiento de circunferencias los datos necesarios son la posición actual del cuadricóptero $p = (x, y)$ y el centro y radio de la sub-trayectoria $O = (x_O, y_O)$ y r.

1. Calcular la posición angular del cuadricóptero relativo al centro de la circunferencia.
$$\epsilon_u = atan2(O, p) \tag{9.12}$$

2. Definir el *sub-waypoint* s a seguir. El signo de λ depende del sentido en que se quiera recorrer, siendo positivo para un seguimiento en sentido horario y negativo para un seguimiento antihorario.
$$s = (O_{\widehat{x}} + r\cos(\epsilon_u \pm \lambda), O_{\widehat{y}} + r\sin(\epsilon_u \pm \lambda)) \tag{9.13}$$

3. Calcular el ángulo de *Yaw* deseado como el ángulo entre el eje x del sistema de coordenadas S_I y la recta que definen los puntos p y s.
$$\varepsilon_d = atan2(p, s) \tag{9.14}$$

9.4. Conclusiones

El algoritmo de seguimiento desarrollado contempla ambos tipos de sub-trayectorias, tanto rectilíneas como circulares.

Además se destaca la presencia de parámetros independientes entre cada sub-trayectoria, permitiendo realizar su ajuste también de forma independiente.

Para los valores de los parámetros seleccionados se observa un perfecto seguimiento si se parte en condiciones ideales. Esto asegura que, a menos de imprevistos como ráfagas de viento, la lejanía a la trayectoria ideal debería mantenerse dentro de un rango aceptable. Esto no implica que el seguimiento vaya a mantenerse perfecto durante todo el vuelo, ya que el modelo utilizado para la simulación del GPS es bastante simplificado además de no estar considerándose el error propio del sistema de posicionamiento.

Capítulo 10

Evasión de obstáculos

El algoritmo de generación de trayectorias explicado en el capítulo 8 no contempla un ambiente en presencia de obstáculos, pero sin duda es necesario lograr detectarlos y evitarlos en pos de incrementar la autonomía del cuadricóptero. Utilizando el sensor de proximidad descrito en el capítulo 6 se logra la detección, permitiendo responder de forma adecuada evitando la colisión.

10.1. Restricciones

El problema general de evitar cualquier clase de obstáculos es complejo y no necesariamente tiene solución, se consideran las siguientes restricciones al problema.

- Los obstáculos no superan los $2m$ de ancho.
- La evasión se realiza sólo en el plano horizontal.
- La distancia entre dos obstáculos consecutivos es de al menos $10m$.

10.2. Detección de obstáculos

Las trayectorias consisten en tramos rectos y arcos de circunferencias, permitiendo realizar el sensado únicamente la dirección de avance.

Para realizar la detección se utiliza el sensor de ultrasonido antes mencionado, capaz de detectar objetos a una distancia de $6m$. Se toma como umbral de peligro los $5m$ de distancia, distancia suficiente para evadir obstáculos de dimensiones de $2m$ de ancho.

Durante el vuelo normal debe sensarse permanentemente la dirección de vuelo, ejecutando normalmente el algoritmo seguimiento de trayectorias

mientras no se detecte peligro de colisión. Dado que medidas aisladas del sensor pueden tener errores considerables, se decide mantener un registro de las últimas seis medidas obtenidas y considerar detectado un obstáculo cuando al menos cuatro de ellas son menores al umbral.

10.3. Algoritmo de evasión

Una vez detectado el obstáculo es necesario dejar de lado el seguimiento de trayectorias para tomar acciones de vuelo que permitan evadirlo.

Se opta por tomar acciones de evasión sin forzar al vehículo a detenerse, ésto se debe a que la calidad de los datos del GPS se deteriora al detener el vehículo. Sí se implementa una disminución de la velocidad de vuelo a la mitad.

Dado que se consideran obstáculos relativamente chicos y aislados, pueden evadirse realizando una curva cuyo radio de curvatura deberá ser menor al fijado para la generación de trayectorias. Con este fin, aparte de variar el ángulo de *Yaw*, se incrementa la capacidad de giro imponiendo ángulo de *Roll* no nulo. Ésta acción deriva en la presencia de una fuerza perpendicular al sentido de avance (eje y_B) que actúa como fuerza centrípeta. Se fija *Roll* como el doble del ángulo *Pitch* actual.

El ángulo *Yaw* se varía 2° en cada loop. El sentido de giro debe ser coherente con la inclinación según según *Roll* y se define de la siguiente forma. Si al momento de detectar el obstáculo se estaba recorriendo una subtrayectoria circular con sentido horario, entonces el giro debe ser en sentido antihorario. Análogamente, se debe realizar un giro en sentido horario si se encontraba recorriendo una circunferencia en sentido antihorario. Para el caso de detectar el obstáculo durante el seguimiento de rectas no se tiene preferencia por ningún sentido, por lo que se fijará por azar.

10.4. Finalización de evasión

Si previamente el cuadricóptero se encontraba evadiendo un obstáculo pero el sensor de ultrasonido ya no lo detecta, debe retomarse el objetivo dado por los *waypoints* ingresados por el usuario.

Volver al seguimiento que se estaba ejecutando previo a la detección del obstáculo puede resultar inconveniente, ya que la evasión puede producir una lejanía considerable a la trayectoria original. Por esta razón se considera mejor opción volver a realizar la generación de trayectorias considerando la posición y orientación en ese instante como *waypoint* inicial y agregar todos los *waypoints* dados por el usuario que no se hayan alcanzado aún.

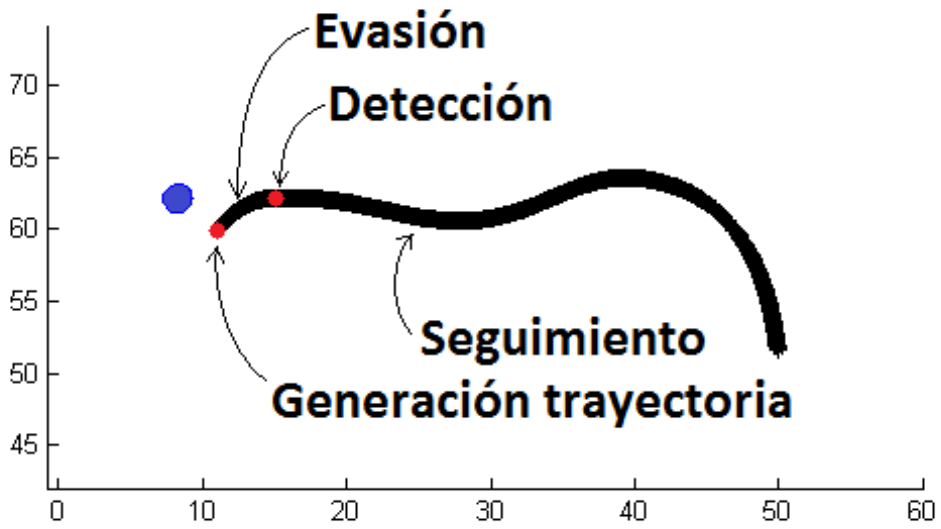


Figura 10.1: Evasión de obstáculos.

10.5. Resumen

A continuación se resume el algoritmo de evasión de obstáculos considerado.

1. Obtener una medida del sensor de ultrasonido y observarla junto a las últimas cinco medidas anteriores. Si al menos cuatro de ellas son menores a los $5m$ encender una bandera de peligro. A su vez debe implementarse otra bandera guarde el estado anterior de la primera.

2. Si la bandera de peligro está apagada pero la anterior se encontraba encendida, regenerar la trayectoria a partir de la posición actual y abandonar el algoritmo de evasión. De lo contrario continuar.

3. Verificar qué sub-trayectoria se encontraba siguiendo previo a la detección del obstáculo y en función de eso decidir un sentido de giro.

   - L: Seleccionar sentido horario.
   - R: Seleccionar sentido antihorario.
   - S: El sentido seleccionado es indiferente, se toma horario.

4. Fijar la velocidad deseada de vuelo en $1{,}5m/s$ y el *Roll* deseado al doble del *Pitch* según el sentido seleccionado anteriormente.

5. En función del sentido seleccionado, incrementar o disminuir el ángulo *Yaw* deseado 2° respecto al anterior.

Parte IV

Controladores

Capítulo 11

Control de altitud

Con el objetivo de realizar el seguimiento de cierta altura deseada z_d a lo largo del recorrido, se diseña e implementa un controlador del tipo PID.

La placa *CC3D* dispone de una entrada denominada *Throttle* mediante la cual se modifica la suma de velocidades que se exige a los motores. La señal *Throttle* influye directamente en fuerza de empuje Th ejercida verticalmente, por lo que altura de vuelo puede ser controlada mediante dicha señal.

11.1. Modelado

Para diseñar el controlador de altura de vuelo se realizó un modelo simplificado para relacionar la fuerza de empuje respecto a la posición vertical.

Una primera hipótesis de trabajo es que los ángulos *Pitch* y *Roll* serán en todo instante suficientemente pequeños como para considerar que la orientación del vector de fuerza ejercida por las hélices será vertical a efectos prácticos. Se estima que ninguno de estos ángulos superará en ningún momento los 15º, por lo que en el peor de los casos se trabaja con una componente igual a $cos(15°)^2 Th$, superando el 90 % del empuje original.

Considerando entonces al cuadricóptero como una masa puntual, utilizamos la primer cardinal obteniendo la ecuación de movimiento de la ecuación 11.1. Donde m es la masa del cuadricóptero, z es la altura de vuelo, Th es la fuerza de empuje, P es la fuerza peso y b es el coeficiente de viscosidad.

$$m\ddot{z} = Th - P - b\dot{z} \tag{11.1}$$

Despreciando la fuerza de rozamiento con el aire se obtiene la ecuación de movimiento 11.2.

$$m\ddot{z} = Th - P \tag{11.2}$$

Considerando la transformada de *Laplace* en pequeña señal, se obtiene la función de transferencia de la ecuación 11.3.

$$H(s) = \frac{Z(s)}{Th(s)} = \frac{1}{ms^2} \tag{11.3}$$

11.2. Diseño del controlador

Se considera el diagrama de bloques de la figura 11.1, donde $C(s)$ es el controlador a diseñar, $z_d(t)$ es la altura a seguir, $z(t)$ es la altura estimada a partir de los sensores del sistema, $e(t) = z_d(t) - z(t)$ es la señal de error y $Th(t)$ es la fuerza de empuje que se aplica al sistema $H(s)$.

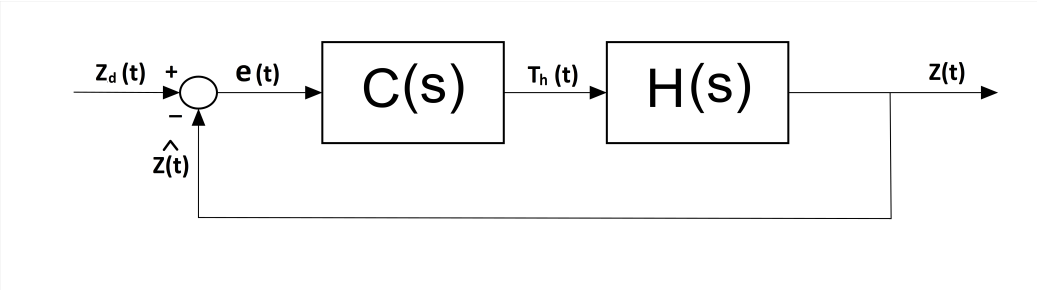


Figura 11.1: Diagrama de bloques control de posición vertical.

La transferencia del sistema en lazo cerrado se expresa en la ecuación 11.4.

$$G_{cl}(s) = \frac{C(s)H(s)}{1 + C(s)H(s)} \tag{11.4}$$

El controlador a utilizar será del tipo PID cuya expresión genérica es la de la ecuación 11.5.

$$C(s) = K_P + \frac{K_I}{s} + K_D s \tag{11.5}$$

Las constantes K_P, K_I y K_D se determinaran con el objetivo de obtener el menor sobretiro posible y un tiempo de asentamiento de al menos $10s$. La exigencia no es superior debido a que el largo de la trayectoria en el plano (x, y) se considera suficientemente mayor a la distancia vertical entre waypoints consecutivos. Además, tenemos una limitante física debido a la saturación de los motores.

Control proporcional

En primera instancia se estudia el caso más sencillo, utilizar un controlador únicamente proporcional. La transferencia en lazo cerrado queda según la ecuación 11.6.

$$G_{cl}(s) = \frac{K_P}{ms^2 + K_P} \tag{11.6}$$

El sistema presenta dos polos complejos en $\pm j\sqrt{\frac{K_P}{m}}$. El lugar geométrico de los polos al variar K_P se muestra en la figura 11.2, donde los polos comienzan en $s = 0$ para $K_P \to 0$ y se alejan por el eje imaginario al incrementar el valor de K_P. El sistema resulta inestable para cualquier valor de K_P,la respuesta temporal será oscilatoria sin amortiguación.

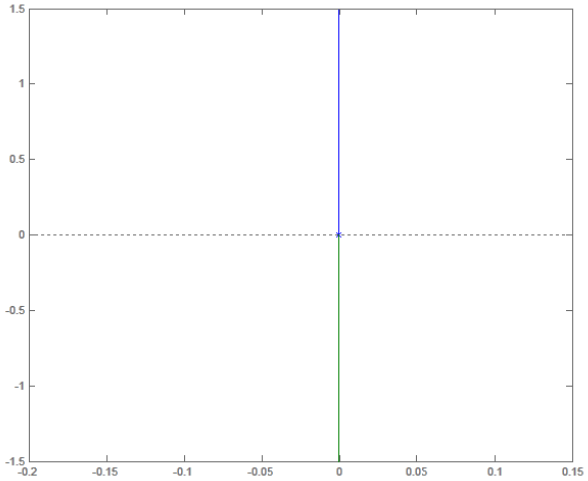


Figura 11.2: Lugar Geométrico de los polos en lazo cerrado variando K_P.

Según el método de *Evans*, el lugar geométrico comienza para $K_P \to 0$ en los polos del sistema en lazo abierto y terminan para $K_P \to \infty$ en los ceros del sistema en lazo cerrado. Por lo tanto, resulta necesario agregar un cero al sistema, de forma que las ramas que se van por el eje imaginario cambien su trayectoria dando posibles K_P que aseguren la estabilidad del sistema. Para lograr esto debe agregarse un termino derivativo al controlador.

Control proporcional derivativo

En este caso la expresión del controlador toma la forma de la ecuación 11.7. Las constantes del controlador se reescribieron como $T_D = K_D/K_P$ y $K = K_P/m$ para facilitar su estudio.

$$G_{cl}(s) = \frac{K_P + K_D s}{ms^2 + K_P + K_D s} = \frac{K(1 + T_D s)}{s^2 + K(1 + T_D s)} \tag{11.7}$$

El sistema en lazo abierto presenta un cero en $s = -1/T_D$ y dos polos en $s = 0$. Fijando T_D en un valor y estudiando el lugar geométrico de los polos al variar K se obtiene el resultado mostrado en la figura 11.3. El punto múltiple

$s = -2/T_D$ se halla calculando las raíces de la derivada de la transferencia en lazo abierto $G_{ol} = C(s)H(s)$.

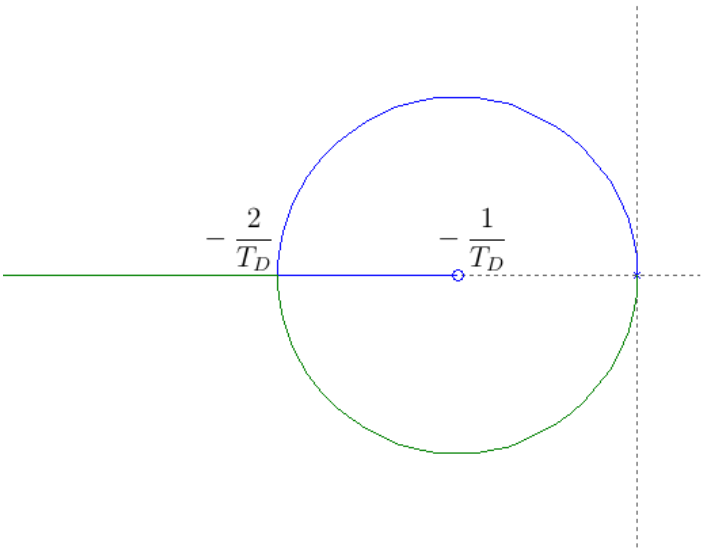


Figura 11.3: Lugar Geométrico de los polos en lazo cerrado fijando K_D y variando K.

Observar que para cualquier $K, T_D > 0$ el sistema es estable. Sin embargo, la presencia de un cero en el sistema de segundo orden implica la existencia de un sobretiro cuyo valor depende de la relación entre las ambas constantes.

La transferencia del sistema en lazo cerrado se puede reescribir como la suma de dos términos según la ecuación 11.8.

El primer término es un sistema de segundo orden sin ceros, cuya respuesta temporal no presenta sobretiro si se cumple que $K \geq 4/T_D^2$.

El segundo término no es más que la derivada del primero multiplicado por un factor de T_D. En la figura 11.4 se observa la respuesta al escalón de cada término y el resultado total, utilizando constantes $K = 1$ y $T_D = 2$.

$$G_{cl}(s) = \frac{K}{s^2 + K(1 + T_D s)} + \frac{K T_D s}{s^2 + K(1 + T_D s)} \tag{11.8}$$

Existe un compromiso entre el tiempo de asentamiento y el sobretiro de la respuesta temporal. Ubicando polos en el punto múltiple (ver figura 11.3), el tiempo de asentamiento resulta aproximadamente $t_s = 3 * (T_D/2) = 3s$ y $K = 4/T_D^2 = 1$. Las constantes del controlador resultan $K_P = m$ y $K_D = 0{,}5$.

La masa del cuadricóptero influye en dichos resultados, su valor puede oscilar entre los $1{,}7Kg$ y los $2Kg$ dependiendo de la carga extra transportada. En la figura 11.5 se aprecia la variación en el lugar geométrico al modificar la masa dentro rango antes dicho.

Finalmente se opta por un controlador PD con las siguientes constantes de control.

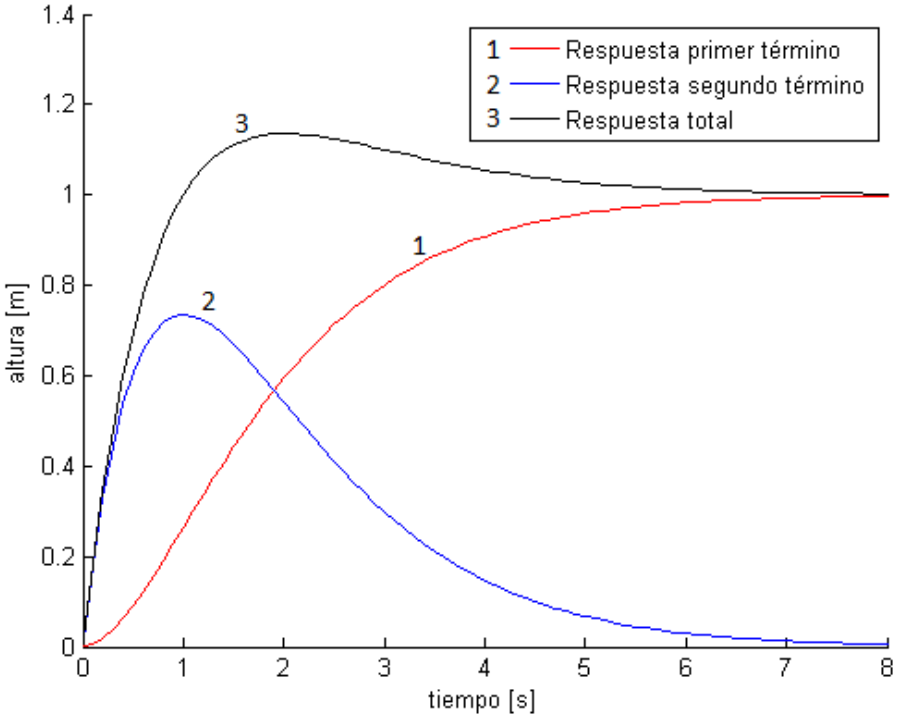


Figura 11.4: Respuesta al escalón del sistema controlado con $K = 1$ y $T_D = 2$.

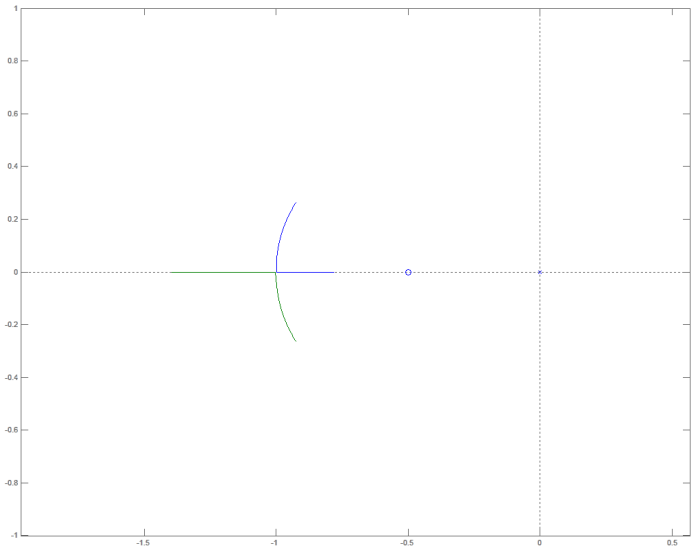


Figura 11.5: Lugar geométrico de los polos del sistema al variar la masa de carga.

$$\left.\begin{array}{l} K = 1 \\ T_D = 2 \end{array}\right\} \quad \Longrightarrow \quad \left\{\begin{array}{l} K_P = 1{,}85 \\ K_D = 3{,}7 \end{array}\right. \qquad (11.9)$$

11.3. Aspectos prácticos

11.3.1. Acondicionamiento de la señal de control

El controlador diseñado maneja como señal de control al empuje en *Newtons* ejercido por el sistema de propulsión. Sin embargo, al momento de la implementación, el controlador tendrá como salida una señal PWM a inyectar en la entrada *Throttle* de la placa CC3D. Resulta necesario conocer la relación que se cumple entre la señal *Throttle* y el empuje efectivamente logrado. En la sección 7.1 se haya la relación entre el comando *Thrust* recibido por la CC3D y el comando PWM que ésta envía a los drivers, ecuación 11.10. En el capítulo 4 se relaciona el comando PWM recibido por el driver con el empuje efectuado, ver ecuación 11.11. Mediante ambos resultados se implementa el bloque M^{-1} (ecuación 11.12) para acondicionar la señal de control. El diagrama de la figura 11.6 ilustra el proceso antes descrito.

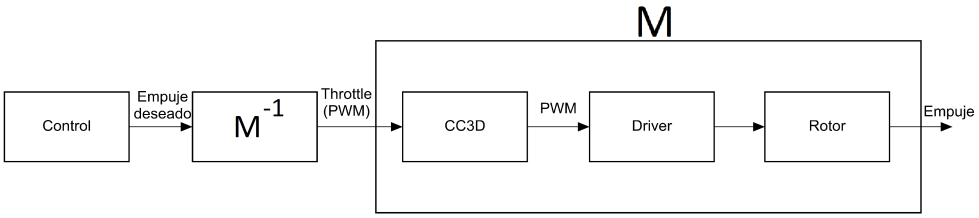


Figura 11.6: Acondicionamiento de la señal de control.

$$Throttle = 1{,}4PWM_{ESC} - 606{,}5 \tag{11.10}$$

$$PWM_{ESC} = -6{,}8Th^2 + 149{,}3Th + 1069{,}3 \tag{11.11}$$

$$Throttle(Th) = -9{,}7Th^2 + 214{,}1Th + 926{,}7 \tag{11.12}$$

11.3.2. Saturación de los motores

Si la diferencia entre la altura deseada y la estimada es muy grande, la señal de error también lo será por lo que la acción de control puede llegar a saturar a los motores.

Utilizando *Simulink* de *MatLab* se simula la señal de *Thrust* una respuesta escalón 11.7. Puede verse como en los primeros instantes de tiempo la fuerza de empuje exigida por el controlador alcanza los $75N$ aproximadamente. Sin embargo, en el capítulo 4 se concluye que la fuerza de empuje lograda está entre los $3{,}14N$ y $23{,}52N$ aproximadamente. Vale la pena destacar que

la fuerza necesaria para mantener el cuadricóptero en una posición vertical fija, considerando que posee una carga media, es de $18{,}13N$ por lo que nos encontramos dentro de los parámetros deseados.

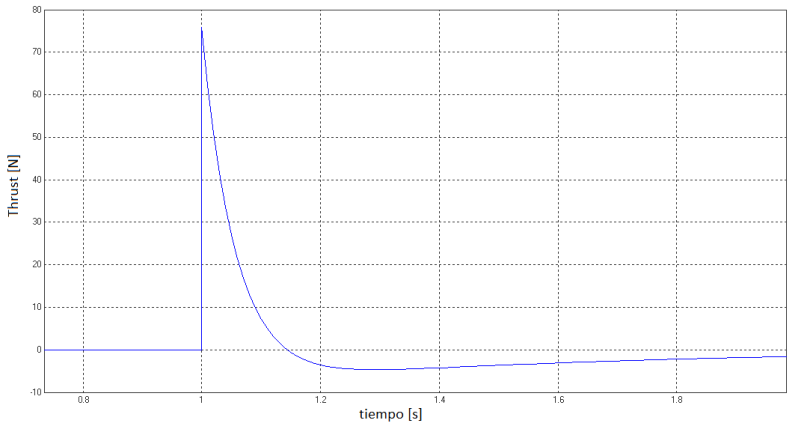


Figura 11.7: Señal de Thrust en la respuesta al escalón.

11.4. Conclusiones

Se logra diseñar y simular e implementar un controlador del altitud del tipo PD, quedando como trabajo a futuro las pruebas y ajustes en el sistema físico.

Capítulo 12

Control de velocidad

Durante los vuelos se pretende mantener una velocidad constante v_d, por lo que debe diseñarse un controlador apropiado que lo permita.

Imponiendo un *pitch* distinto de cero, se logra una componente horizontal de la fuerza ejercida por los motores, dando lugar a un cambio en la aceleración. El movimiento del cuadricóptero en estas condiciones es de aceleración no uniforme, ya que la fuerza de fricción con el aire varía conforme con la velocidad hasta alcanzar un punto de equilibrio. Luego de alcanzar dicho estado, el movimiento se mantiene a velocidad constante idealmente.

12.1. Modelado

Para realizar un modelo sencillo se utilizan las ideas planteadas durante la simulación del seguimiento de trayectorias, suponiendo un movimiento plano y sin viento.

$$m\vec{\dot{v}} = mg\sin(\psi)\widehat{yaw} - B\vec{v} \tag{12.1}$$

Donde $\vec{v}$ es la velocidad de vuelo, $\widehat{yaw}$ es el versor orientado según el ángulo Yaw, m es la masa del cuadricóptero, g es la aceleración gravitacional y B es el coeficiente de fricción.

Si se considera un vuelo en linea recta, los vectores $\vec{v}$, $\vec{\dot{v}}$ y el versor $\widehat{yaw}$ son colineales. Por otro lado, es sabido que el ángulo de Pitch siempre será suficientemente pequeño como para realizar la aproximación $\sin(\psi) \approx \psi$. Aplicando estas simplificaciones a la ecuación anterior se deduce la siguiente relación entre el Pitch impuesto y la velocidad de vuelo.

$$m\dot{v} = mg\psi - Bv \tag{12.2}$$

Dicha ecuación describe un modelo útil para el diseño del controlador de velocidad. La transferencia entre las dichas variables presenta un único polo en $s = B/m = 0{,}32$

$$\frac{V(s)}{\psi(s)} = H(s) = \frac{g}{s + \frac{B}{m}} \tag{12.3}$$

12.2. Diseño del controlador

Puede verse en la ecuación 12.3 que el sistema en lazo abierto se modela como un sistema de primer orden con un polo real negativo. Por lo tanto, idealmente el sistema es estable y sin sobretiro en su respuesta al escalón.

Control proporcional

Dadas las características del modelo, es claro que el control proporcional es suficiente para obtener una buena performance en la respuesta del sistema.

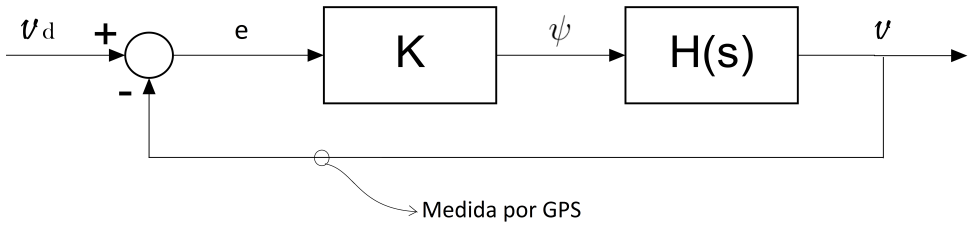


Figura 12.1: Sistema realimentado control proporcional.

La transferencia en lazo cerrado para este caso toma la siguiente forma.

$$G_{CL}(s) = \frac{Kg}{s + \frac{B}{m} + Kg} \tag{12.4}$$

El sistema continúa presentando un único polo real negativo, cuyo valor varía dependiendo del valor seleccionado para la constante de proporcionalidad K según se observa en la figura 12.2

El problema presente en el sistema realimentado de la figura 12.1 es la presencia de un error asintótico no nulo. La solución implementada consiste en el agregado de un pre-filtro que compense la ganancia en continua, es decir, un bloque de constante $\frac{B}{Kgm} + 1$.

Por otro lado, debe considerarse el retardo producido por el control de Pitch de la *CC3D*. Según lo estudiado en el capítulo 7 la constante de tiempo de dicho control es de aproximadamente $0{,}77s$, por lo que debe agregarse un bloque que lo modele.

En estas condiciones el sistema realimentado agrega un segundo polo en $s = -1{,}23$ modificando el lugar geométrico al variar la constante de control proporcional (ver figura 12.4).

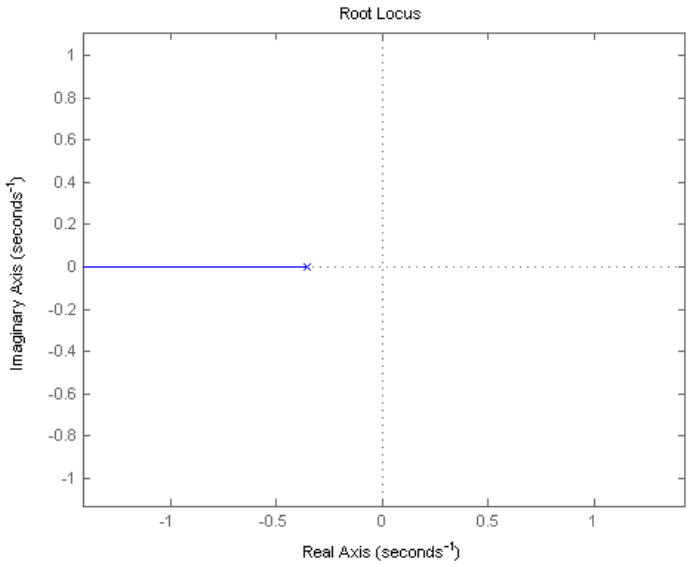


Figura 12.2: Lugar geométrico de los polos del lazo cerrado.

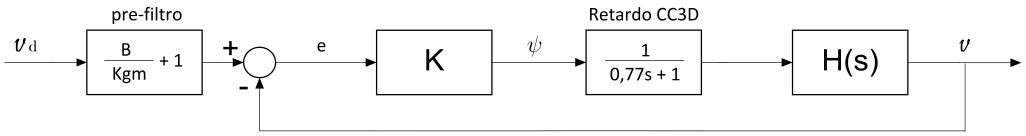


Figura 12.3: Sistema realimentado con pre-filtro, controlador proporcional y retardo de la *CC3D*.

Se diseña el valor de K de forma tal que los polos del sistema en lazo cerrado se ubiquen en el punto múltiple $s = 0{,}812$. Por lo tanto su valor se fija en $K = 0{,}019$.

Para este caso la respuesta al escalón no presenta sobretiro y el tiempo de asentamiento es de $5{,}8s$ como se puede ver en la figura 12.5.

12.3. Conclusiones

Si bien el diseño se basa en un modelo teórico bajo ciertas simplificaciones, se considera el retardo dado por la *CC3D* al controlar el *Pitch*. Además, según las experiencias descritas en el capítulo 7, el control de *Roll* es satisfactorio en términos generales. Esto nos permite asumir que el control de velocidad actuará de forma aceptable durante un vuelo real.

Los resultados obtenidos en la respuesta al escalón del lazo cerrado presentan tiempos de asentamiento suficientes, ya que no se considera crítico alcanzar la velocidad crucero con mayor rapidez. También se observa la au-

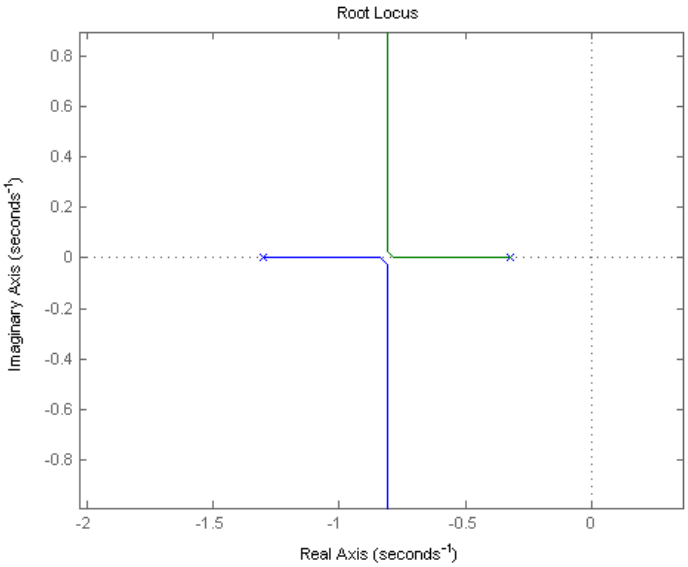


Figura 12.4: Lugar geométrico de los polos del lazo cerrado completo.

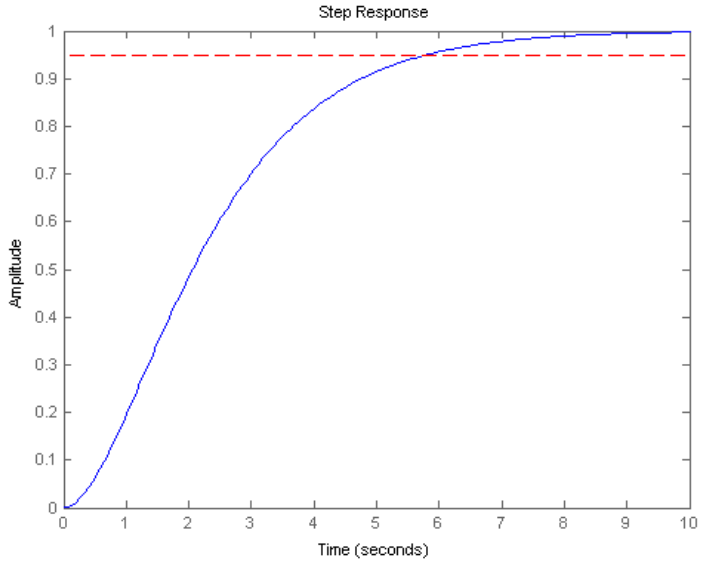


Figura 12.5: Respuesta al escalón del diseño final.

sencia de sobretiro considerando una carga media.

Finalmente se observa cómo el pre-filtro soluciona el problema del error asintótico no nulo.

Capítulo 13

Control de ángulo Yaw

13.1. Introducción

El ángulo *Yaw* determina la dirección de la componente horizontal del empuje ejercida por los rotores siempre que se imponga *Roll* nulo. Disponer del control del mismo resulta imprescindible para realizar el seguimiento de trayectorias. Se implementa un controlador del tipo *PID* para éste fin.

El sistema $H'(s)$ a controlar consiste en el cuadricóptero controlado en actitud por la placa *CC3D*, ésta ya implementa un control de velocidad angular de *Yaw*. La respuesta de la planta está caracterizada en la sección 7.5, donde se concluye que un modelo simplificado de primer orden brinda información suficiente para el diseño del controlador.

La ecuación 13.1 corresponde a la transferencia del sistema $H'(s)$, el término $(1/s)$ integra la salida original del sistema $\dot{Yaw}$. El lazo de control se aprecia en la figura 13.1, la realimentación $\widehat{Yaw}$ se estima a partir de medidas sensoriales de la IMU que viene incluida en placa *CC3D*.

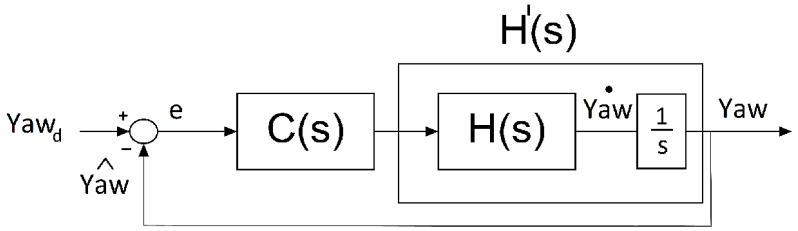


Figura 13.1: Lazo de control de *Yaw*.

$$H'(s) = H(s)\frac{1}{s} = \frac{1}{s(1 + T_s s)} \tag{13.1}$$

La constante de tiempo del sistema según lo visto en la sección 7.5 resulta $T_s = 0{,}41s$.

13.2. Diseño del controlador

El objetivo consiste en diseñar e implementar un controlador para el sistema de la ecuación 13.1, teniendo como objetivo la rapidez en respuesta y la estabilidad. En primer lugar se abarca el caso más sencillo, el controlador proporcional.

13.2.1. Controlador proporcional

La transferencia del lazo cerrado para el caso del controlador proporcional queda representado en la ecuación 13.3. A partir del análisis del lugar geométrico de las raíces (figura 13.2), se concluye en primer lugar que se obtiene un sistema estable que la constante de proporcionalidad debe ser tal que los polos queden en el entorno al punto múltiple. De esta forma se obtiene el sistema de respuesta menor tiempo de asentamiento posible manteniendo sobretiro nulo.

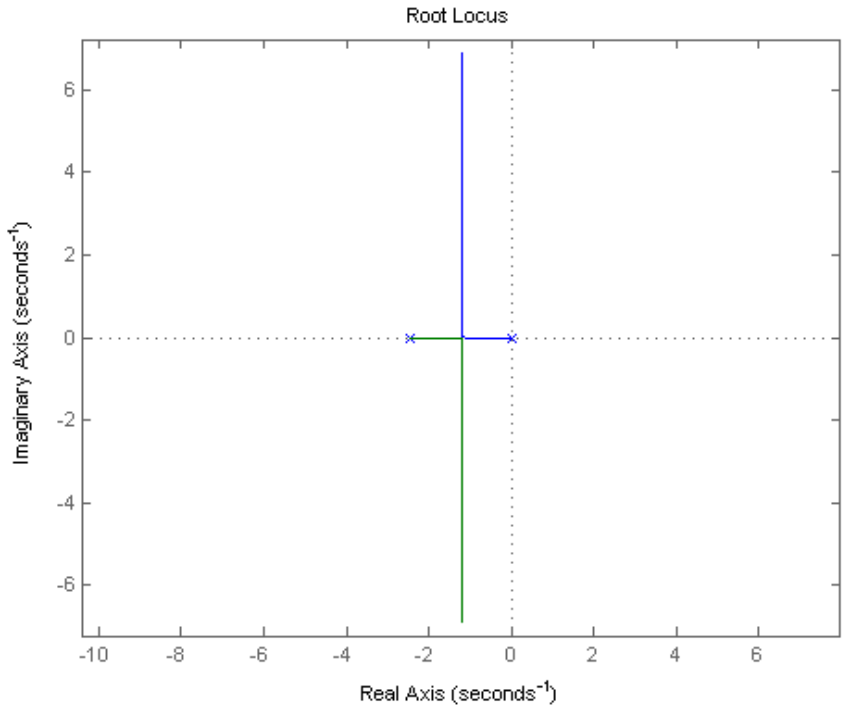


Figura 13.2: Control proporcional, lugar geométrico de las raíces.

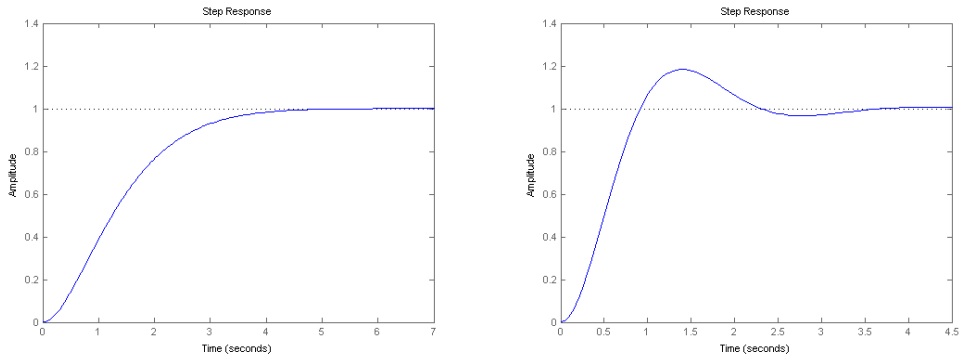
k_p	$t_{0-90\,\%}(s)$	M_p (%)
0.7	2.71	0
2.7	0.81	18

Tabla 13.1: Control proporcional,respuesta al escalón simulada.

$$C^P(s) = k_p \tag{13.2}$$

$$G^P_{cl}(s) = \frac{C^P(s)H'(s)}{1 + C^P(s)H'(s)} = \frac{k_p}{T_s s^2 + s + k_p} \tag{13.3}$$

Se simula la respuesta escalón para dos ganancias $k_p = 0{,}7$, que sitúa a los polos en el entorno al punto múltiple y $k_p = 2{,}7$ que induce sobretiro pero disminuye considerablemente el tiempo de levantamiento, ver tabla 13.2 y figuras 13.3a y 13.3b.



(a) Respuesta escalón, ganancia 0.7. (b) Respuesta escalón, ganancia 2.7.

Figura 13.3: Simulación control proporcional de constante $k_p = 0{,}7$.

13.2.2. Controlador Proporcional Derivativo

Al agregar un término derivativo se espera una disminución del el efecto de sobretiro, debido a que el controlador tendrá en cuenta la velocidad a la que se llega al objetivo. El controlador PD queda determinado en la transferencia de la ecuación 13.4.

$$C^{PD}(s) = k_p(1 + T_d s) \tag{13.4}$$

La constante derivativa $T_d = 0{,}4$ se elige de forma tal que el polo ingresado quede cercano y a la izquierda del polo de la planta ($T_s = 0{,}41s$).

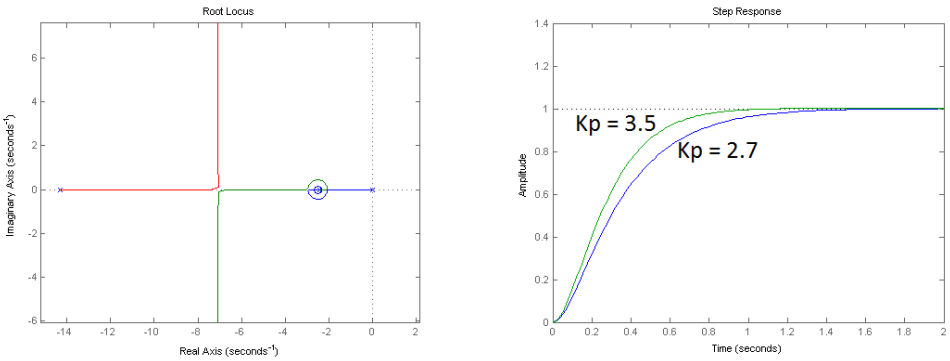
Adicionalmente se aplica un filtro pasa-bajos $F(s)$ (ecuación 13.5) a la salida del controlador de manera de suavizar la señal de control evitando saturar los motores en los primeros instantes de la respuesta escalón. Implica el agregado de un polo en $-14{,}3$ que resulta irrelevante al momento de analizar el diagrama de ceros y polos.

$$F(s) = \frac{1}{(1 + T_f s)} \tag{13.5}$$

La transferencia en *Laplace* del lazo cerrado adopta la expresión de la ecuación 13.6.

$$G_{cl}^{PD}(s) = \frac{C^{PD}(s)F(s)H'(s)}{1 + C^{PD}(s)F(s)H'(s)} \tag{13.6}$$

El lugar geométrico de los polos del lazo cerrado se aprecia en la figura 13.4a. El sistema cuenta con tres polos y un cero, para ganancias entre 2.7 Y 3.5 los polos permanecen en el eje real. Se simulan ambos extremos obteniendo las respuestas escalón de la figura 13.4b.



(a) Lugar geométrico de las raíces.

(b) Simulación de la respuesta escalón para ganancias 2.7 y 3.5.

Figura 13.4: Diseño del controlador PD.

La tabla 13.2 muestra los tiempos de levantamiento de cero a noventa porciento según la ganancia del controlador. En ambos casos no se registra sobretiro debido a que los polos del lazo cerrado se encuentran en la recta real.

13.3. Pruebas sobre el sistema real

13.3.1. Procedimiento

k_P	$t_{0-90\,\%}(s)$
2.7	0.75
3.5	0.57

Tabla 13.2: Control *PD*,respuesta al escalón simulada.

k_p	$t_{0-90\,\%}$	$t_{0-90\,\%}$ simulado	$M_p(\,\%)$	$M_p(\,\%)$ simulado
0,7	2.16	2.71	0.51	0.00
2,7	0.87	0.81	24.30	18.00

Tabla 13.3: Respuesta al escalón real y simulada, controlador proporcional.

Las pruebas se realizan sobre cuadricóptero colgado como se aprecia en la figura 13.5. El sistema tiene libertad giro en torno al eje z (ángulo *Yaw*), tanto *Roll* como *Pitch* permanecen fijos. El control y la recopilación de los datos (*logs*) se ejecutan en la *Beagleboard* a bordo, de manera similar a la que actúa durante un vuelo real.

A continuación se analizan las respuestas escalón para los controladores diseñados.

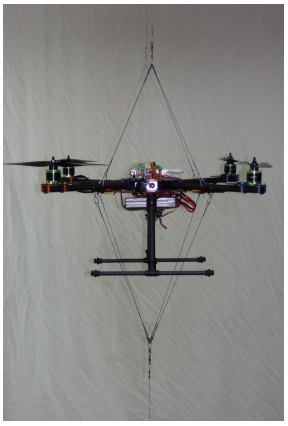


Figura 13.5: Libertad de giro únicamente en *Yaw*.

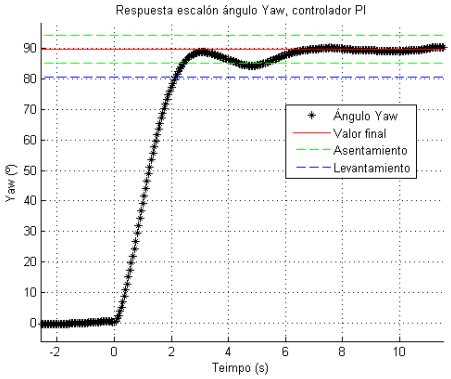
13.3.2. Controlador proporcional

La figura 13.6a muestra la respuesta del sistema con control proporcional de constante $k_p = 0{,}7$. La dimensión del escalón de entrada es de 90°.

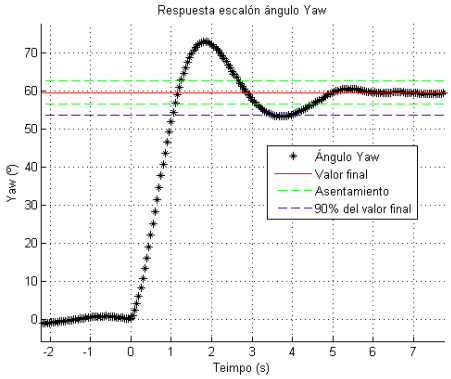
La diferencia que se observa entre la respuesta real y la simulación de la figura 13.3 se explica principalmente por la simplificación a un modelo de primer orden de la planta. Estas diferencias no implican que el controlador no haya superado las expectativas de diseño. Se obtiene un tiempo de levantamiento menor al de la simulación y un sobretiro prácticamente nulo, ver tabla 13.3.

Como era de esperar, aumentar la ganancia del controlador implica un incremento en la rapidez en la respuesta y el sobretiro.

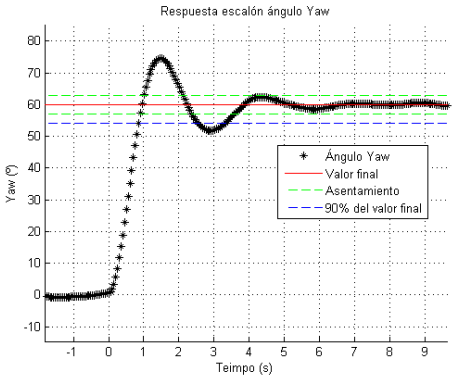
La figura 13.6 se analiza la respuesta al aumentar la ganancia del controlador. Los resultados presentados en la tabla 13.4 muestran el compromiso existente entre el aumento de velocidad de respuesta y el sobretiro.



(a) Controlador de ganancia 0.7.



(b) Controlador de ganancia 1.5.



(c) Controlador de ganancia 2.7.

Figura 13.6: Respuestas escalón, controlador proporcional.

k_p	$t_{0-90\,\%}$	t_s	$M_p(\,\%)$
0,7	2.16	5.35	0.51
1,5	1.05	4.57	21.01
2,7	0.87	3.59	24.30

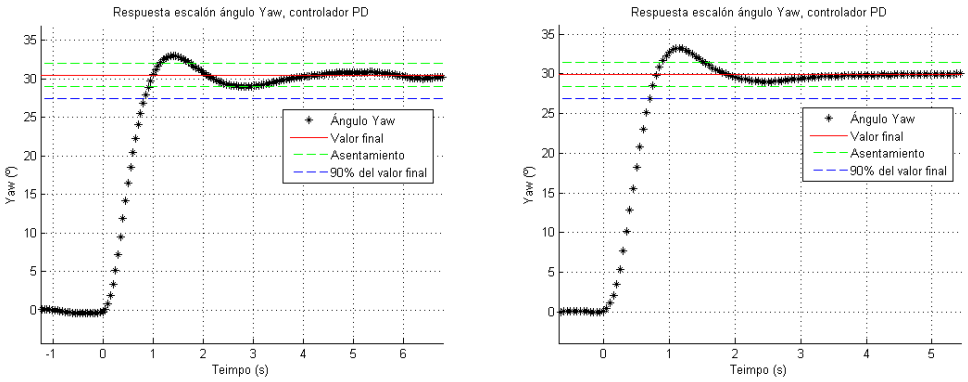
Tabla 13.4: Tiempos de respuesta y sobretiro al variar la ganancia del controlador proporcional.

13.3.3. Controlador Proporcional Derivativo

Se prueban dos controladores, uno de ganancia 2,7 y otro de ganancia 3,5.

Para el controlador PD de ganancia 2,7, la respuesta resulta 1,5s más rápida que la del controlador proporcional. Sin embargo, a diferencia de la respuesta simulada, se observa un sobretiro considerable. El origen del mismo se le atribuye nuevamente al modelo simplificado de la planta.

Aumentando la ganancia se obtiene una respuesta 10ms más rápida, pero a su vez se aumenta el sobretiro al 11 %.



(a) Controlador de ganancia 2.7. (b) Controlador de ganancia 3.5.

Figura 13.7: Respuestas escalón, controlador PD.

k_p	$t_{0-90\,\%}$	$t_{0-90\,\%}$ simulado	t_s	$M_p(\,\%)$
2,7	0.64	0.75	1.74	8.10
3,5	0.51	0.57	1.55	11.00

Tabla 13.5: Parámetros de la respuesta escalón, controlador *PD*.

Aspecto práctico

Se descarta el uso del controlador PD debido a los motivos expuestos a continuación. La señal de control concentra una gran cantidad de energía en los primeros instantes de acción, permitiendo influir de forma significativa sobre el sistema logrando la respuesta rápida esperada. El problema radica en que la acción inicial influye de forma drástica en la dinámica del sistema y en el funcionamiento de los motores. Los tirones y oscilaciones bruscas observadas nos son aceptables durante el vuelo. Ésta respuesta brusca puede ser aplacada mediante la el uso de un filtro pasa-bajos a la salida del controlador, sin embargo se estaría perdiendo la esencia del controlador. Resulta más sencillo el uso del controlador proporcional cuya perfomance es aceptable para el seguimiento de trayectorias, tal como se comprueba más adelante.

13.4. Conclusiones

Si bien la respuesta al escalón más rápida y de menor sobretiro se logra mediante el uso de un controlador proporcional derivativo, éste produce señales de control demasiado bruscas para el sistema físico. Mediante el controlador proporcional se puede llegar a lograr un tiempo de asentamiento de $3{,}6s$ a costa de un sobretiro de $24\,\%$.

Parte V

Implementación

Capítulo 14

Montaje del sistema electrónico

Este capítulo describe las cuestiones prácticas del armado del cuadricóptero en lo que respecta al montaje de las distintas piezas de electrónica y su interconexión.

En la figura 14.1 se presenta un diagrama ilustrativo de las conexiónes de datos entre todas las partes. En la figura 14.3 se muestran exclusivamente las conexiones de alimentación.

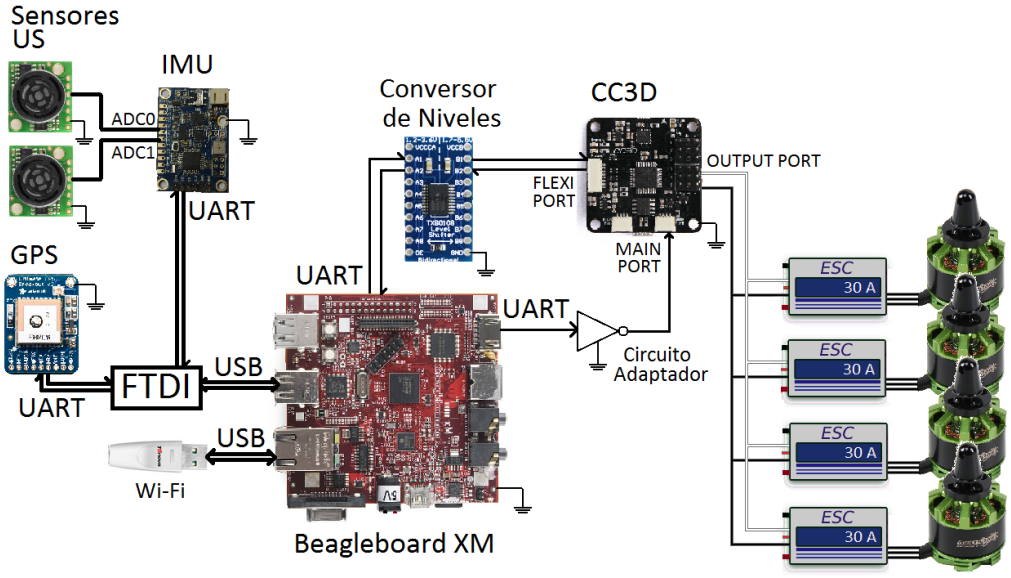


Figura 14.1: Diagrama de conexionado de datos.

14.1. Conexiones de la *Beagleboard*

La *Beagleboard* es la pieza central de la inteligencia del cuadricóptero y como tal centraliza toda la información. Esto implica una serie de conexiones

físicas con los demás componentes. A continuación se listan las distintas piezas de hardware con las que se comunica la *Beagleboard* y se detallan características de la conexión:

- ***CC3D*:**

  1. Se realiza una primera conexión para enviar los comandos de actitud y throttle deseados. Esta se realiza entre la UART2 de la *Beagleboard* y el Main Port de la *CC3D*. Es necesario adaptar la señal de salida de la primera para que tome las características de la señal que espera la segunda.
  2. La segunda conexión establece la comunicación de Telemetría entre ambas placas[1] para recibir las medidas de actitud. Esta conexión es entre la UART1 de la *Beagleboard* y el Flexi Port de la *CC3D*. En este caso también se deben adaptar los niveles lógicos de las señales.

- **IMU y GPS:** Se conectan a través de un FTDI a los puertos USB de la *Beagleboard*.

- **Dongle Wi-Fi:** Se conecta directamente a cualquiera de los puertos USB disponibles.

La implementación de la electrónica necesaria para realizar las conversiones de niveles lógicos se detalla en la sección 14.3.

14.2. Conexiones de la *CC3D*

Además de las conexiones con la *Beagleboard*, la *CC3D* se conecta con los cuatro ESCs para comandar los motores y recibe la alimentación del BEC de uno de ellos. Los detalles de las conexiones se presentan en la tabla 14.1.

CC3D (Output port)	**ESC (PWM in)**
Pin -	GND (cable negro)
Pin ·	Signal (cable amarillo)
Pin +	Signal (cable rojo)

Tabla 14.1: Conexiones entre *CC3D* y ESCs.

[1] Se implementó únicamente la decodificación del mensaje que contiene la actitud estimada. Pero es posible obtener más datos de la *CC3D* si se implementa la decodificación completa del protocolo que esta utiliza (Protocolo *UAVTalk*, ver sección 15.6.1).

14.3. Electrónica de conversión de niveles

En esta sección se describe la implementación de la electrónica necesaria para realizar las conversiones de niveles lógicos necesarias para la interconexión de las distintas placas.

14.3.1. Circuito Acondicionador

Para que la *CC3D* pueda recibir adecuadamente los mensajes *Futaba SBUS* de la *Beagleboard* es necesario adaptar los niveles lógicos de las señales, ya que el Main Port de la primera trabaja a 5V y la UART de la segunda a 1.8V. Además, debido a las características del protocolo *Futaba SBUS* (Ver sección 15.6.2) la señal de salida de la *Beagleboard* debe ser invertida. Por estas razones se debió implementar el circuito de la figura 14.2.

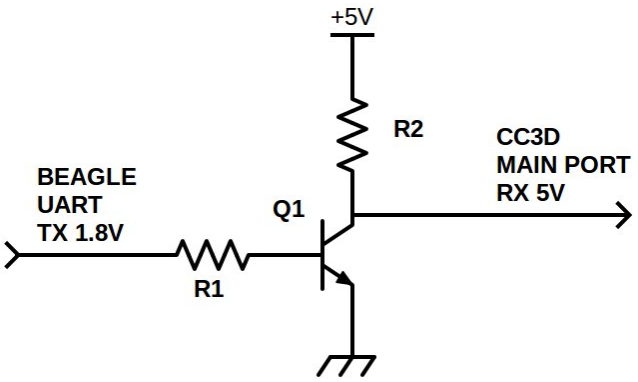


Figura 14.2: Circuito para adaptar la señal SBUS.

14.3.2. Conversor de niveles

Se debe conectar una UART de la *Beagleboard* con el Flexi Port de la *CC3D*. Dado que este último funciona a 3.3V y la *Beagleboard* a 1.8V, se utiliza el conversor de niveles *Adafruit TXB0108* para interconectarlos.

14.4. Alimentación

Tanto la electrónica como los motores se alimentan de la Batería LiPo. Dado que esta fuente es de 11.1V nominales, es necesario adaptar su tension de salida para los distintos componentes. En la tabla 14.2 se presentan todos los elementos presentes en el cuadricóptero con sus respectivas fuentes de alimentación y sus salidas de tensión.

Elemento	Fuente de alimentación		Salidas de tensión
Regulador Switcheado	Batería LiPo	11.1V	5V
Beagleboard	Regulador Switcheado	5V	5V ∨ 1.8V
CC3D	BEC de ESC	5V	5V
ESCs	Batería LiPo	5V	5V
IMU	FTDI	3.3V	Ninguna
GPS	FTDI	3.3V	Ninguna
Sensores Ultrasonido	BEC de ESC	5V	Ninguna
LD33V	*Beagleboard*	5V	3.3V
Conversor de niveles	*Beagleboard* LD33V	1.8V 3.3V	Ninguna
Circuito Inversor	*CC3D*	5V	Ninguna

Tabla 14.2: Voltaje de alimentación de cada elemento

Para mayor claridad respecto del conexionado referirse al diagrama de la figura 14.3.

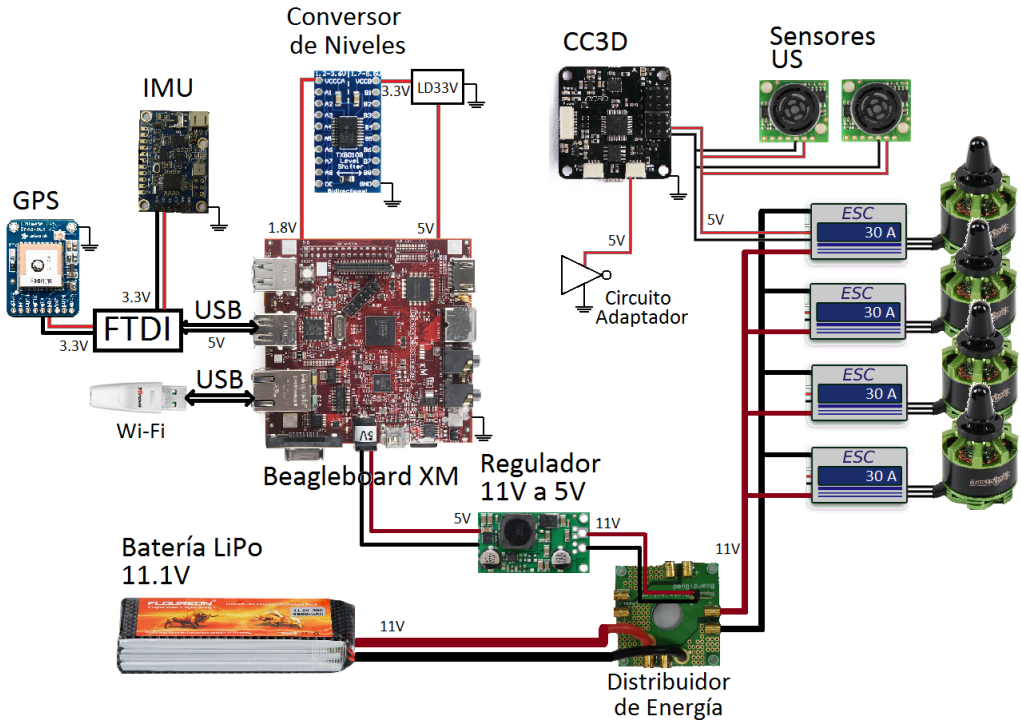


Figura 14.3: Diagrama de alimentación.

Capítulo 15

Software

En este capítulo se describen el diseño e implementación del software y algunas características de las plataformas utilizadas. La información respecto a la obtención, compilación y ejecución del código, así como los detalles de la comunicación con el sistema se encuentran en el anexo A.

15.1. Estructura general del software

El software implementa la autonomía de vuelo del cuadricóptero. Esto es básicamente generar la trayectoria y tomar las acciones de control necesarias para que el cuadricóptero la siga, sumado a la comunicación entre los distintos procesadores y sensores utilizados. Se tomaron en cuenta las siguientes consideraciones a la hora del diseño:

- Estar escrito en lenguaje `C`.
- Deberá ser compilado y ejecutado en la *Beagleboard*. Esta cuenta con un sistema operativo embebido *Linux* de distribución *Ångström*[1].
- Estructura modularizada donde cada módulo (implementado como una librería) cumpla una función específica.

A modo introductorio, se presentan las principales funciones del software:

- Generación de la trayectoria en base a los waypoints por los que se desea pasar.
- Seguimiento de la trayectoria generada (control de posicion, velocidad y orientación).

[1]En el Anexo A se encuentra información sobre la compilación y configuración del sistema operativo.

- Comunicación con la *CC3D* a través de un puerto serie para el envío de comandos de actitud deseada y lectura de la actitud estimada. Para garantizar los tiempos de ejecución, el programa que se encarga de enviar los comandos a la *CC3D* corre por separado del programa principal.
- Lectura de datos de la IMU (contiene sensores de ultrasonido y barómetro) y GPS a través de un puerto serie.
- Escritura en memoria no volátil de información generada durante el vuelo.

La naturaleza de la aplicación requiere que el software se ejecute en tiempo real, lo que se dificulta cuando se tienen varias operaciones de lectura/escritura de puertos serie. Para evitar demoras se implementaron dichas operaciones de forma que nunca bloqueen la ejecución del programa. También se implementaron algunas de las funciones de lectura/escritura en procesos independientes, tanto para evitar demoras como para asegurar tiempos críticos[2].

Existen dos programas que corren en hilos separados del principal y son llamados por este durante la etapa de incialización. Estos son el demonio[3] SBUSD y el demonio *GPSD*[4]. El primero es el responsable de enviar los comandos a la *CC3D* a través de un puerto serie, mientras que el segundo se encarga de recibir y decodificar la información provista por el GPS y facilitarla al usuario.

Se implementaron también ciertas modificaciones en el software original[5] de la IMU. Este está escrito en lenguaje `Arduino` y se encarga de recolectar las medidas de varios sensores para luego enviarlas a través de un pueto serie. Las modificaciones en el software de la IMU incluyen la incorporación de la lectura de las medidas de los sensores de ultrasonido y la modificación del período con el que se envían los datos.

15.2. Programa principal

15.2.1. Inicialización

Durante la inicialización se prepara el sistema para el vuelo, para esto se abren todos los puertos de comunicación necesarios, se ejecutan los procesos

[2]Este es el caso del protocolo *Futaba SBUS* utilizado para comandar la *CC3D* (ver sección 15.6.2).

[3]Un demonio (el término en inglés es daemon) es un proceso informático que se ejecuta en segundo plano.

[4]`http://catb.org/gpsd/`

[5]El software original de la IMU fue implementado por los fabricantes (*ckdevices*) y modificado posteriormente por el grupo u*Quad!* [16].

independientes al programa principal y se genera la trayectoria inicial.

Se pueden configurar ciertos modos de operación modificando el archivo `quadcop_config.h`. Las modificaciones se utilizan para facilitar el debugging del software o para probar algunas funcionalidades por separado. Las configuraciones disponibles son:

- `DEBUG:` Habilita mensajes al usuario para facilitar el depurado del software.
- `PC_TEST:` Deshabilita algunas características del software que funcionan únicamente en la *Beagleboard* para permitir la ejecución del programa en cualquier PC con *linux*.
- `SIMULATE_GPS:` Deshabilita las conexiones con el GPS y simula la variación de posición del cuadricóptero en base al modelo físico.
- `DISABLE_UAVTALK:` Deshabilita la comunicación de Telemetría con la *CC3D*.
- `FAKE_YAW:` Si la Telemetría es deshabilitada, se puede simular el Yaw usando un modelo del sistema controlado.
- `SOCKET_TEST:` Crea un servidor TCP para entablar comunicación con otra PC y enviarle los datos de posición instantánea. Esto permite graficar en tiempo real la posición del cuadricóptero utilizando por ejemplo *Matlab*.

Los elementos más relevantes de la inicialización del programa principal son:

- El generador de trayectorias requiere del archivo de texto `way_points_in.txt` del cual lee los waypoints para generar la trayectoria.
- El GPS debe ser conectado a un puerto USB de modo que el sistema operativo lo mapee al archivo `/dev/ttyUSB0`. Luego, escribiendo los comandos necesarios en dicho archivo, el dispositivo GPS puede ser configurado para funcionar a 10Hz y 57600 baudios. Finalmente, se ejecuta el programa *GPSD* para comenzar a recibir los datos.
- El proceso SBUSD debe ser ejecutado en una etapa inicial del programa y se le debe especificar el puerto serie en donde se encuentra conectada la *CC3D*. En el caso de estar conectada en la UART1, la ruta en el sistema operativo sería `/dev/ttyO0`. Se implementó un proceso independiente para enviar los comandos a la *CC3D* para garantizar los tiempos requeridos por el protocolo *S-BUS*.

- La IMU se conecta a través de un FTDI para ser mapeada por el sistema operativo en el archivo `/dev/ttyUSB1`. Por defecto la IMU envía las medidas en formato `ASCII`, pero tiene implementada la opción de enviarlas de forma binaria si se le envía el caracter `'!'`. Dado que se implementó la decodificación de las medidas en formato binario[6], durante la inicialización se le envía a la IMU el caracter mencionado.

15.2.2. *Loop* principal

El *loop* principal tiene una duración constante de 50ms y se puede dividir en tres grandes bloques, recolección de medidas, ejecución del algoritmo de seguimiento y toma de acciones de control. Si embargo, no todas las medidas tienen la misma cadencia, por lo que hay tareas que no se realizan en todos los *loops*.

Las siguientes medidas están presentes cada 50ms: ángulos de Euler, altura y distancia en la dirección del vuelo. La cadencia de dichas medidas permite realizar el control de Yaw, el control de altura y la detección de obstáculos todos los *loops*. Pero los datos de posición y velocidad (dados por el GPS) están presentes cada 100ms, por lo que el algoritmo de seguimiento de trayectorias y el control de velocidad se realizan cada intervalos de dicho tiempo, es decir, se saltean un *loop*.

En la figura 15.1 se puede ver un diagrama de flujo del *loop* principal, donde se ilustran las dos variantes del mismo según su duración y las tareas asociadas a cada uno de ellos. También se muestran los dos modos de operación, que son el vuelo normal, donde se sigue la trayectoria generada, y el modo evasión de obstáculos. Cada tarea se basa en funciones de una librería en particular, a continuación se detallan las diferentes tareas y se nombran las librerías (escritas entre `< >`) asociadas a cada una de ellas:

- **Medidas actitud** `<UAVTalk>`: La *CC3D* envía mensajes con las medidas de actitud cada 50ms y estos son decodificados mediante el protocolo UAVTalk. Esta envía cada cierto tiempo otros mensajes además de la Telemetría (ver sección 15.6.1). Dichos mensajes deben ser decodificados y descartados rápidamente para no perder medidas ni demorar el *loop* principal. La librería *UAVTalk* también se encarga de calcular y descontar el offset de la medida de Yaw.

- **Medidas IMU** `<imu_comm>`: Esta librería se encarga de decodificar los datos que la IMU envía cada 50ms. Dado que estos datos son medidas crudas de los sensores (*e.g.*, cuentas de ADC), la librería también se encarga de convertir los datos a las variables de interés.

[6]Se optó por utilizar el formato binario porque reduce considerablemente la cantidad de bytes que deben ser leidos.

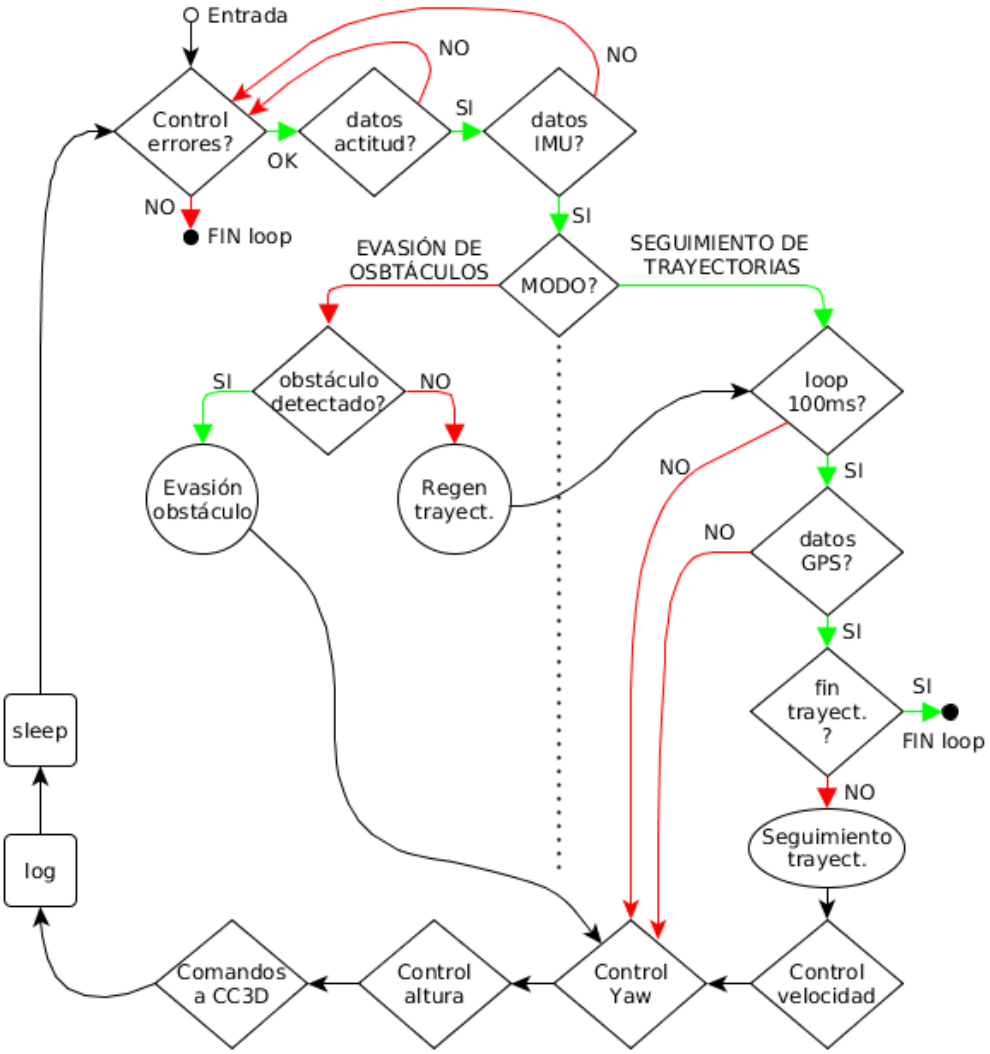


Figura 15.1: Diagrama de flujo del *loop* principal.

- **Medida GPS** `<gps_comm>`: A intervalos de 100ms se le solicita al programa *GPSD* los datos actualizados del GPS. Luego se deben convertir los valores de latitud y longitud a posición en (`x,y`). De todo esto se encarga la librería *gps_comm*.

- **Seguimiento de trayectorias y modo normal** `<path_following>`: El seguimiento, que está a cargo del módulo *path_follower*, necesita datos actualizados de posición para poder funcionar. Por lo tanto, este se ejecuta solamente si hay datos nuevos del GPS. Este módulo es el encargado de proveer el ángulo de Yaw deseado que alimenta el controlador de Yaw.

- **Detección de obstáculos y modo evasión** `<obstacle>`: La función de esta librería es detectar los posibles obstáculos en base a las medidas del sensor de proximidad entregadas por la IMU. La evasión comienza cuando el obstáculo es detectado y dura mientras este se encuentre en el rango de detección del sensor. Al finalizar la evasión, se regenera la trayectoria y se retorna al modo normal de operación.

- **Regeneración de trayectoria** `<path_planning>`: Se llama a la misma función que genera la trayectoria inicial pero en este caso se le pasa como parámetros la posición actual y los waypoints que aún no fueron alcanzados.

- **Control:** `<control_altura>`, `<control_velocidad>`, `<control_yaw>`: Los controladores están implementados en librerías separadas. Los controles de Yaw y de altura se ejecutan cada 50ms y se encargan de seguir las señales (generadas por el seguimiento de trayectorias) de Yaw deseado y de altura deseada respectivamente. En cambio, el control de velocidad se ejecuta cada 100ms ya que requiere de medidas del GPS.

- **Comandos a la *CC3D*** `<uquad_kernel_msgq>`, `<futaba_sbus>`: Luego que las señales de control son convertidas a comandos PWM, estos son enviados a través de mensajes de kernel al proceso SBUSD. Luego, este se encargará de generar el mensaje SBUS y enviárselo a la *CC3D* a través de un puerto serie.

- **Log y control de tiempos:** Antes de finalizar el *loop* se escriben en memoria no volátil una serie de variables de interés. Por último, se controla cuanto tiempo pasó desde el comienzo del *loop* y se pone el programa en modo sleep hasta completar los 50ms.

15.3. Módulos de control

Dada la completa independencia de los controladores entre sí, se implementan los módulos de control en librerías separadas, es decir, una librería por cada variable a controlar. A continuación se detallan los aspectos más relevantes de la implementación de los controladores.

15.3.1. Control de Yaw

Recibe la medida de *Yaw* de la *CC3D* y el ángulo de Yaw deseado calculado por el seguimiento de trayectorias. Se puede optar por control proporcional o proporcional derivador, para esto se debe configurar el parámetro `CONTROL_YAW_ADD_DERIVATIVE` en el archivo `control_yaw.h`.

Control proporcional

El control proporcional se implementó mediante la siguiente ecuación:

$$u = K_P(\theta_d - \hat{\theta}) \tag{15.1}$$

Donde θ_d es el ángulo de *Yaw* deseado, dado por el módulo *path_following*, $\hat{\theta}$ es la medida del *Yaw* leida de la *CC3D* y K_P es la constante de proporcional diseñada en el capítulo 13.

Control derivador

Para el computo de la derivada de la señal de error se realiza un cociente incremental, pero dado que esto puede generar ruido indeseado, se promedian algunas muestras antes del cálculo. La operación realizada es la siguiente:

$$\dot{e_k} = \frac{mean(e_k, .., e_{k-n+1}) - mean(e_{k-n}, .., e_{k-2n-1})}{nT} \tag{15.2}$$

Donde $2n$ es la cantidad de muestras utilizadas en el cálculo y T es el tiempo de muestreo.

El control derivativo puede generar grandes incrementos en la señal de control si la señal a seguir cambia abruptamente. Para atenuar dichos saltos, se implementó un filtro pasabajos de primer orden a la salida del controlador:

$$y_k = \alpha x_k + (1 - \alpha) y_{k_1} \tag{15.3}$$

Aún con estas consideraciones es posible que la señal de control supere los niveles permitidos por la *CC3D*. En estos casos, se limitan las señales antes de ser enviadas.

Un aspecto importante de la implementación es cómo se adapta la salida del controlador para ser enviada a la *CC3D*. Esto es necesario ya que la primera tiene unidades de velocidad angular (radianes por segundo en este caso), mientras que la segunda recibe señales PWM[7]. Para realizar la conversión se utilizó la siguiente ecuación:

$$PWM_{\dot{\theta}} = \frac{25}{11}\dot{\theta} + 1500 \tag{15.4}$$

Donde $PWM_{\dot{\theta}}$ es el comando enviado a la *CC3D* y θ la salida del controlador. Los datos necesarios para deducir la conversión dada por la ecuación 15.4 fueron obtenidos de la documentación de *OpenPilot* [8].

[7]Las señales PWM son codificadas mediante el protocolo *Futaba S-BUS* (ver sección 15.6.2).

15.3.2. Control de altura

La implementación del control de altura es análoga a la del control de yaw en modo PD. Para realimentar la altura se utiliza el barómetro o el sensor de ultrasonido dependiendo del caso. A continuación se describe por separado la implementación de la parte proporcional y la parte derivativa del controlador.

Control proporcional

$$u_P = K_P(h_d - \hat{h}) \tag{15.5}$$

Donde h_d es la altura deseada, dado por el seguimiento de trayectorias, $\hat{h}$ es la medida de altura y K_P es la constante de proporcional diseñada en el capítulo 11.

Control derivador

Este caso es idéntico al control derivativo de Yaw, ya que se deriva el error en lugar de la señal a seguir. Por lo tanto, se implementa el cálculo del error utilizando la expresión 15.7.

Para controlar la altura se actúa sobre la señal *Throttle* de la *CC3D*, pero la salida del controlador implementado tiene unidades de fuerza. Por lo tanto, se convierte la señal de salida del controlador mediante la siguiente ecuación (en el capítulo 11 se encuetra la deducción de dicha ecuación):

$$PWM_{Throttle} = -9{,}7Th^2 + 214{,}1Th + 926{,}7 \tag{15.6}$$

Donde $PWM_{Throttle}$ es el comando enviado a la *CC3D* y Th la salida del controlador. Los coeficientes de la ecuación 15.6 se obtuvieron experimentalmente durante la caracterización de la *CC3D* y de los motores (ver capítulos 7 y 4 respectivamente).

15.3.3. Control de velocidad

Para el control de velocidad se implementó un control proporcional dado por siguiente expresión:

$$u = K_P(v_d - \hat{v}) \tag{15.7}$$

Donde v_d es la velocidad deseada, dada por el usuario al inicio del programa, $\hat{v}$ es la medida de velocidad dada por el GPS y y K_P es la constante de proporcional diseñada en el capítulo 12.

De igual manera que en los controladores mencionados anteriormente, fue necesario adaptar la señal de contorl antes de enviarla a la *CC3D*. En este

caso la salida del controlador es el ángulo de *Pitch*, que se convierte en una señal PWM compatible con la *CC3D* de la siguiente manera:

$$PWM_{\psi} = 8{,}7804\psi + 1500 \tag{15.8}$$

Donde PWM_{ψ} es el comando enviado a la *CC3D* y ψ la salida del controlador. Los datos necesarios para deducir la conversión dada por la ecuación 15.8 fueron relevados experimentalmente durante la caracterización de la *CC3D* (ver capítulo 7).

15.4. Módulo generador de trayectorias

Esta librería se encarga de generar una trayectoria total[8] en base a los waypoints ingresados por el usuario. Para lograr esto se recorre una lista con los waypoints a seguir (ordenados temporalmente) y se realiza, entre pares consecutivos de waypoints, el algoritmo de generación presentado en el capítulo 8. Una vez generada la trayectoria total, esta es almacenada en una lista ordenada para ser utilizada por el algoritmo de seguimiento.

15.5. Módulo seguidor de trayectorias

Se implementó una función que recibe una lista con las trayectorias a seguir y decide que algoritmo de seguimiento utilizar, según la sub-trayectoria[9] sea circular o rectilínea. Esta función se encarga también de controlar la finalización de las sub-trayectorias y el consecuente pasaje a la siguientes. También determina si se completó la trayectoria total.

Para calcular el ángulo de Yaw deseado, el algoritmo de seguimiento utiliza la función *atan2()* y esta presenta discontinuidades en $\pm 2\pi$. Para evitar esto se utilizó la función *fix()* como se puede ver en la ecuación 15.9.

$$\theta_{d_k} = \begin{cases} \theta_{d_k} & \left|\theta_{d_k} - \theta_{d_{k-1}}\right| < \pi \\ \theta_{d_k} - 2\pi fix\left(\frac{\theta_{d_k} - \theta_{d_{k-1}} - sign\left(\theta_{d_k} - \theta_{d_{k-1}}\right)}{2\pi}\right) & \left|\theta_{d_k} - \theta_{d_{k-1}}\right| \geq \pi \end{cases} \tag{15.9}$$

Donde θ_{d_k} es el ángulo de Yaw deseado en el instante k.

[8] Dado que se llamó *trayectoria* al camino a seguir entre dos waypoints, se denomina *trayectoria total* a aquella que une todos la totalidad de los waypoints.

[9] Se le denomina sub-trayectoria al elemento unidad de una trayectoria, este puede ser una recta o un arco de circunferencia.

15.6. Comunicación con la *CC3D*

15.6.1. *UAVTalk*

UAVTalk es un protocolo binario de código abierto diseñado para la comunicación de UAVs, principalmente para el envío de telemetría a estaciones terrestres. Es un protocolo de alto nivel independiente del medio físico que se utilice, ya que provee el transporte de estructuras de datos encapsuladas en objetos[10].

El protocolo implementa mensajes de *handshake* para entablar una comunicación estable y de *acknowledge* para asegurar la correcta recepción de objetos.

Se desarrolla en la *Beagleboard* un decodificador del protocolo *UAVTalk* para recibir los mensajes que contienen la actitud estimada por la *CC3D*. Esta es la principal fuente de información para realimentar los controladores. No se implementa la decodificación de ningún otro mensaje, pero se configura la *CC3D* para que envíe los datos de actitud de forma periódica, independientemente de recibir o no respuesta a sus mensajes.

15.6.2. *Futaba S-BUS*

El protocolo propietario *S-BUS* fue creado por la compania Japonesa *Futaba*, este permite la comunicación de un receptor RF con hasta 18 servos mediante un único cable. Esto es posible ya que la información de cada servo es multiplexada y enviada mediante un puerto serie, en lugar de la forma tradicional en la que se utiliza una señal PWM para cada servo. Se optó por utilizar esta opción frente al PWM debido a que la *Beagleboard* dispone únicamente de tres salidas PWM controladas por hardware.

Cabe destacar que el trabajo de ingeniería inversa para determinar las características del protocolo no fue realizado en el transcurso del presente trabajo, sino que fue tomado de otro proyecto[11] de código abierto.

Los mensajes S-BUS poseen 25 bytes de longitud y deben ser enviados con una periodicidad de 14ms. Están formados de la siguiente manera:

```
Sync(8bits) Canal_1(11bits) ... Canal_16(11bits) End(8bits)
```

El protocolo S-BUS deriva del RS232, sus principales características son:

- Utiliza una señal UART invertida.
- 10000 baudios.

[10] `https://wiki.openpilot.org/display/WIKI/UAVObjects`

[11] `http://developer.mbed.org/users/Digixx/notebook/futaba-s-bus-controlled-by-mbed/`

- 8 bits de datos.
- Paridad par y dos bits de parada.

Para que la *Beagleboard* pueda cominucarse con la *CC3D* mediante este protocolo, es necesario adaptar la señal como se menciona en la sección 14.3.

15.7. Comunicacion con el GPS

Como se mencionó anteriormente, se utilizó el programa *GPSD* para recibir y decodificar la información del GPS. Este programa se encuentra disponible para descargar desde el repositorio de *Ångström*. Para ejecutarlo, es necesario especificarle la ruta al puerto serie donde estaría conectado el GPS y un puerto TCP a donde publicar los datos leídos. Para poner en funcionamiento el programa hay que ejecutar el comando:

```
gpsd /dev/ttyUSB0 -S 1234 -N
```

Desde el programa principal se accede a la información entregada por *GPSD* mediante la libreria *gpslib*, también disponible para descargar del repositorio de *Ångström*.

15.8. Comunicación con la IMU

La IMU ya posee varios sensores integrados en la placa, como es el caso del barómetro. Además, se conectaron los dos sensores de ultrasonido a dos de sus ADC. La comunicación entre la *Begleboard* y la IMU se hace a través de un FTDI conectado a uno de los puertos USB de la primera.

La IMU agrupa los resultados de las medidas en un mensaje y las envía de forma binaria a través de un puerto serie. Cada trama de datos consiste de un byte de inicio, 30 bytes correspondietes a las medidas de los sensores y un byte de finalización. Para parsear el mensaje se utiliza información del tamaño que ocupa cada medida en memoria.

Capítulo 16

Pruebas sobre el sistema implementado

Una vez diseñadas e implementadas las distintas partes que componen al cuadricóptero, se realizan pruebas que combinan el funcionamiento real de ciertos elementos y simulaciones de otros. Restringiendo su libertad de movimiento, es posible corroborar la correcta interacción entre los distintos módulos diseñados sin poner en riesgo al sistema físico.

16.1. Descripción de la simulación

Para restringir los grados de libertad de movimiento del cuadricóptero se utiliza el *Canetígero*, el cual permite realizar giros únicamente en *Pitch* y *Yaw*. Por lo tanto, quedan restringidos los giros en *Roll* y los desplazamientos en x, y y z.

Para la ejecución del simulador, se carga un archivo `way_points_in.txt` en la *Beagleboard* que contiene las especificaciones de la misión deseada. Ésto es una lista de *waypoints* que incluye las posiciones (x, y, z) y sus respectivas orientaciones θ expresadas en grados para facilitar su uso al operador.

En función de dichas especificaciones, la *Beagleboard* genera una trayectoria a recorrer para cumplir con el objetivo, tal como se ejecutaría en un vuelo real.

Finalizada la generación de la trayectoria e inicializadas la distintas partes del sistema, comienza la simulación de vuelo. La misma consiste en realizar una inclinación según el *Pitch* simulando el avance del vehículo y giros según el *Yaw* simulando la dirección de vuelo.

El ángulo de *Pitch* deseado se fija en $-6{,}6°$, correspondiente a la inclinación necesaria para alcanzar una velocidad de $3m/s$ en régimen. Por otro lado, el ángulo de *Yaw* deseado varía según las exigencias del algoritmo de seguimiento.

Para la ejecución del algoritmo de seguimiento es necesario obtener una medida del GPS. Dado que esto no es posible en esta prueba, se utiliza la

simulación descrita en el capítulo 9 para engañar al sistema en cuanto a su ubicación espacial.

Por último, se establece una comunicación vía *WiFi* entre la *Beagleboard* y un PC que ejecuta un programa en *MatLab*. Dicho programa recibe, en primera instancia, la trayectoria generada por el cuadricóptero y luego su posición instantánea a medida que avanza el estado de la simulación. Graficando tanto la trayectoria ideal como la posición instantánea, se obtiene una medida del comportamiento del seguimiento de trayectorias en tiempo real. También se puede apreciar el efecto del control de seguimiento en la respuesta física del cuadricóptero en el ángulo *Yaw*.

Módulos probados

- Generación de trayectorias
- Seguimiento de trayectorias
- Comunicación:
  - *Beagleboard* - PC
  - *Beagleboard* - *CC3D*
  - *CC3D* - Sistema de propulsión
- Respuesta de la *CC3D* en *Pitch* y *Yaw*
- Control de *Yaw*

Módulos simulados

- GPS

Módulos no presentes

- Comunicación:
  - *Beagleboard* - GPS
  - *Beagleboard* - *IMU*
- Respuesta de la *CC3D* en *Roll* y *Thottle*
- Sensores de altitud
- Detección y evasión de obstáculos
- Control de altitud
- Control de velocidad

16.2. Resultados

A modo de ejemplo se presentan los resultados de una prueba en particular, aunque sin duda existen infinitas posibilidades dependiendo de los *waypoints* ingresados.

16.2.1. Objetivo de vuelo

El archivo `way_points_in.txt` utilizado contiene los siguientes *waypoints*:

0	0	0	0
23	-30	0	50
20	10	0	200
5	0	0	180
5	30	0	-90

Las filas contienen los *waypoints* en orden de precedencia, siendo la primer fila el *waypoint* de partida. Las columnas contienen, ordenados de izquierda a derecha, los valores de x, y, z y la orientación respectivamente.

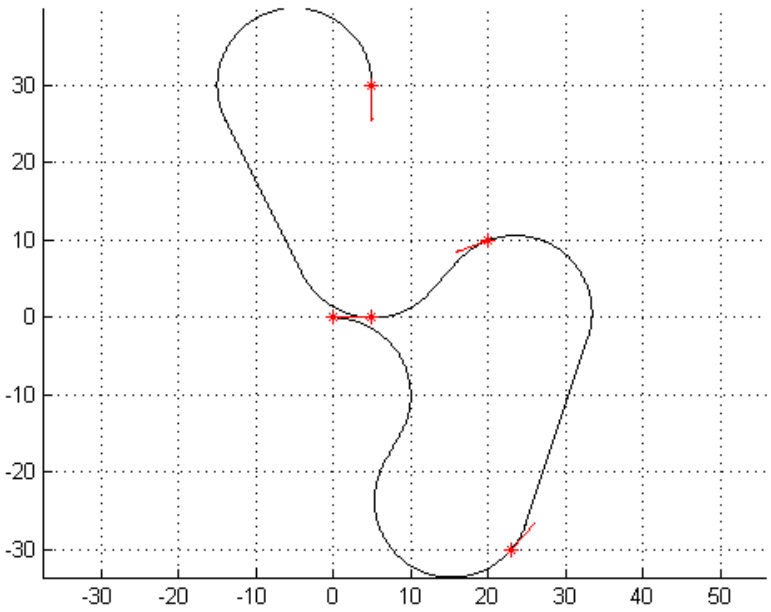


Figura 16.1: *Waypoints* y trayectoria generada.

16.2.2. Generación de la trayectoria

La figura 16.1 muestra los *waypoints* dados con sus respectivas orientaciones y la trayectoria generada. Puede verse el correcto funcionamiento del algoritmo ejecutado en *Beagleboard*.

16.2.3. Seguimiento de la trayectoria

Durante la ejecución del seguimiento se puede observar, además de su funcionamiento, la respuesta del sistema en términos de posición angular, omitiendo el ángulo de *Roll* debido a la restricción dada por el *Canetígero*.

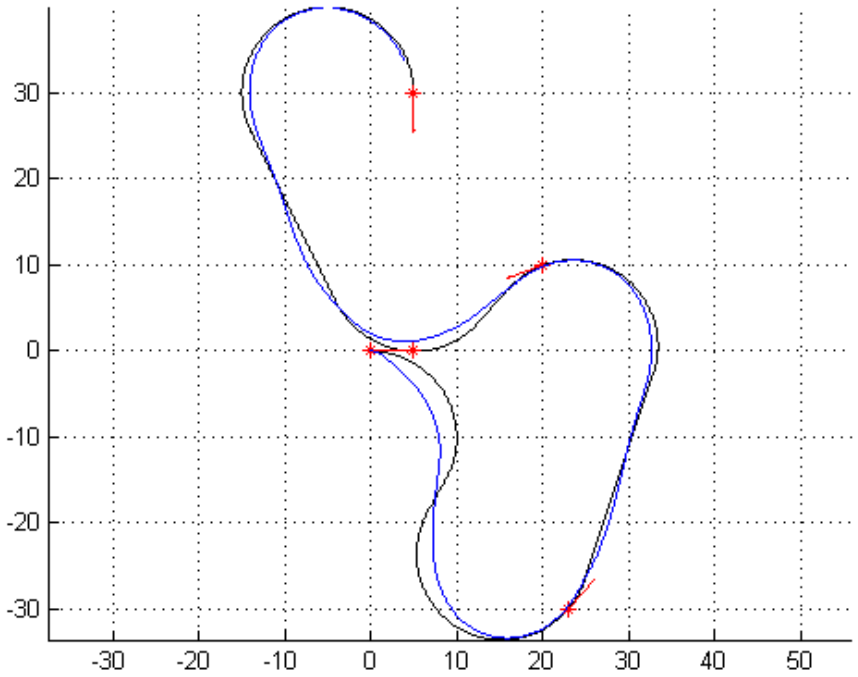


Figura 16.2: *Waypoints*, trayectoria generada y seguimiento.

La figura 16.2 muestra el gráfico obtenido luego de finalizada la trayectoria. En ella puede verse un seguimiento razonablemente bueno, donde la primer parte es la que presenta mayor desviación frente a la trayectoria ideal. Esto se debe a la condición inicial de la velocidad, la cual es nula, combinado con el hecho de comenzar describiendo una curva circular.

16.3. Conclusiones

Lo expuesto en este capítulo es la prueba más cercana al funcionamiento real que se dispone hasta el momento. Sin considerar el control de altura y de velocidad, se comprueba la actuación satisfactoria del sistema que brinda autonomía de vuelo al vehículo. La actuación del algoritmo de seguimiento supera con creces los objetivos planteados.

Capítulo 17

Conclusiones

En el siguiente capítulo se exponen conclusiones referentes al proyecto en general, analizando los objetivos logrados, planteando mejoras y trabajo futuro.

Conclusiones generales

Se logró diseñar e implementar un sistema que cumple en buena medida con los objetivos fijados. Si bien no se llegaron a realizar la totalidad de las pruebas sobre el sistema real, se puedo concluir la efectividad del trabajo mediante simulaciones.

En primer lugar se destaca la acertada elección de componentes que permitieron el armado de un prototipo funcional. Existe al día de hoy una extensa variedad de piezas disponibles para armar un cuadricóptero, y no todas las combinaciones pueden resultar en un producto final adecuado. Esto se debe sobre todo a la dificultad para encontrar la combinación óptima entre las distintas partes, ya sea por estimar correctamente las capacidades del producto final, o por encontrar la compatibilidad entre productos de distintos fabricantes.

Se invirtió una considerable cantidad de tiempo en la familiarización con las piezas de electrónica que se utilizaron, sobre todo en el manejo de la *Beagleboard* y en la implementación de la comunicación entre todas las partes que componen el sistema. El resultado obtenido en este aspecto fue satisfactorio, se logró una correcta interconexión de todos los componentes y una adecuada distribución de energía.

Se considera que la elección del controlador de actitud *CC3D* adquirido permitió centrar el foco del proyecto en solucionar la autonomía de vuelo. Lamentablemente, esta decisión se tomó muy adentrado el proyecto, momento en el que se replantearon varios de los caminos que se habían tomado. Esto, junto con otros factores, produjeron un considerable atraso en los tiempo

planteados inicialmente, lo que derivó en la imposibilidad de concluir el proyecto como se esperaba. Sin embargo, se considera provechosa la elección de adquirir la *CC3D*, dado que resultó de fácil integración y cumplió ampliamente con las expectativas.

En la fase de caracterización se obtuvo información de gran importancia para el posterior desarrollo de los distintos algoritmos y controladores. Por un lado se realizó una caracterización de los instrumentos de medida empleados, generando información sobre su funcionamiento y evaluando la calidad de las estimaciones. La caracterización del sistema físico controlado en actitud por la *CC3D* resultó central para el diseño de los controladores y la sintonización del algoritmo de seguimiento de trayectorias.

Se implementa un algoritmo de generación de trayectorias que resulta la base de la autonomía de vuelo del cuadricóptero. El resultado, si bien limita los posibles de movimientos del cuadricóptero a rectas y circunferencias, garantiza la generación de trayectorias que cumplen con creces los objetivos planteados. Estos fueron:

- Pasar por los waypoints objetivos con la dirección indicada.
- El seguimiento de las trayectorias autogeneradas debe ser realizable por el cuadricóptero.
- La trayectoria debe ser eficiente en lo que respecta a minimizar la distancia recorrida entre los objetivos.

Se implementa con éxito un algoritmo de seguimiento de trayectorias que basa su funcionamiento en el manejo del ángulo *Yaw*, actuando de esta manera sobre la componente horizontal del empuje de los motores. Se realizaron pruebas exhaustivas de dicho algoritmo en un sistema parcialmente simulado, obteniendo resultados alentadores. Se logró seguir todas las trayectorias probadas con un error menor al planteado en los objetivos.

Para completar la autonomía de vuelo se diseñaron e implementaron controladores de seguimiento de altura, de velocidad y un algoritmo de detección y evasión de obstáculos. Debido a falta de tiempo, estos últimos aspectos fueron simulados pero no pudieron ser probados.

Modificaciones en el planteamiento original

En un principio, se tomó como punto de partida un prototipo desarrollada por el proyecto anterior (uQuad!). Se pretendía utilizar su plataforma física y software implementado para solucionar el control de actitud. Sin embargo, luego de varias idas y vueltas, se optó por abandonar dicho proyecto debido a diversos motivos. Por un lado, la plataforma física (estructura, motores y energía) se encontraba obsoleta. El avance de la oferta en los últimos

años permitía conseguir un equipamiento superior sin aumentar el presupuesto. Por otro lado, dado que el funcionamiento del prototipo heredado no se encontraba suficientemente depurado, dejarlo en condiciones para poder desarrollar nuestro proyecto a partir del mismo requería excesivo tiempo y trabajo. Los argumentos planteados anteriormente motivaron la adquisición del controlador de actitud *CC3D*.

Pese a las modificaciones de la idea original a seguir, los objetivos principales no se vieron alterados en su esencia. Se mantuvo el compromiso de diseñar un sistema de autonomía de vuelo para un vehículo con arquitectura de cuadricóptero.

Trabajo futuro

Resta probar el funcionamiento de los controladores de velocidad y de altitud en el sistema real. Para esto es necesario implementar la integración de las medidas de los sensores (ultrasonido y GPS).

Una vez superado el punto anterior, se debe continuar con la fase de pruebas en condiciones normales de vuelo. Es probable que las constantes de los controladores y algoritmos, calibradas para las condiciones específicas del laboratorio, deban ser modificadas en el marco de un vuelo real.

Los *waypoints* están determinados según las coordenadas cartesianas. Queda pendiente implementar una interfaz para el usuario que permita ingresar los objetivos en coordenadas geográficas, ésto facilita el uso del prototipo en aplicaciones reales donde los objetivos son dados en éste tipo de coordenadas.

Se plantea la posibilidad de aumentar la capacidad sensorial en pos la mejorar la estimación de la posición. Para esto se analizaron diferentes opciones, como la utilización de una cámara (ver [20] y [5]) o la utilización de correcciones para las medidas del GPS (ver [12] y [19]).

En un futuro proyecto que siga la misma linea de investigación, queda abierta la posibilidad mejorar la autonomía del cuadricóptero implementado, explorando nuevas soluciones para la generación de trayectorias, seguimiento de trayectorias y evasión de obstáculos.

Apéndice A

Manual de Usuario

En este manual contiene los pasos necesarios para compilar el kernel de *Linux* ((con distribución *Ángström*) que corre en la *Beagleboard*. Luego se explica cómo obtener, compilar y ejecutar el software implementado.

A.1. Compilación del Kernel

Los siguientes pasos describen como compilar un kernel de *Linux* con distribución *Ángström* para su uso en la *Beagleboard-Xm*. Para la compilación se utilizó un PC con *Ubuntu 14.04*. La compilación hace uso de las herramientas *OpenEmbedded*[1] y *bitbake*, en conjunto con los scripts que automazian el proceso provistos por los desarrolladores de *Ángström*. El procedimiento presentado a continuación fue tomado principalmente de [14].

Antes de comenzar, se deben instalar los siguientes paquetes:

```
sudo apt-get install sed wget cvs subversion git-core \
coreutils unzip texi2html texinfo docbook-utils gawk \
python-pysqlite2 python-ply python-progressbar diffstat \
help2man make gcc build-essential g++ desktop-file-utils \
chrpath dosfstools kpartx
```

A.1.1. Paso 1 – Preparar el entorno

Lo primero que se debe hacer es reconfigurar el dash, para esto ejecutar el siguiente comando y seleccionar la opción `No`:

```
sudo dpkg-reconfigure dash+
```

[1] http://www.openembedded.org/wiki/Documentation

Para obtener los scripts que automatizan la descarga de los fuentes necesarios y la compilación (setup scripts):

```
mkdir Angstrom && cd Angstrom+
git clone git://github.com/Angstrom-distribution/setup-scripts.git
```

Para lograr que *Open Embedded* no trabaje con la versión más reciente de *Ångström* (Kernel 3.0.17+), luego de hacer el primer clone de los `setup-scripts` se debe pasar a una versión anterior del repositorio:

```
cd setup-scripts
git branch -a
git checkout angstrom-v2013.06-yocto1.4
```

Se puede configurar *bitbake* para que use determinada cantidad de *threads*. En este caso se configura para funcionar con cuatro. Para esto editar el archivo `./conf/local.conf`:

```
gedit conf/local.conf+
```

Y modificar las siguientes líneas:

```
PARALLEL_MAKE     = "-j4"
BB_NUMBER_THREADS = "4"
```

Se puede evitar que se borren los archivos temporales al finalizar la compilacion. Esto puede ser útil para ver el estado de los fuentes antes de la creación de la imagen del kernel. Para lograr esto se debe comentar (agregar '`#`' al inicio) la siguiente linea del archivo `./conf/local.conf`:

```
# INHERIT += "rm_work"
```

A.1.2. Paso 2 – Descarga de fuentes y primera compilación

Si se encuentra detras de un proxy, vea el arcivo `oebb.sh`, este permite la configuración del mismo.

Para comenzar la compilación:

```
MACHINE=beagleboard ./oebb.sh config beagleboard/
MACHINE=beagleboard ./oebb.sh update
. ./environment-angstrom-v2013.06
```

Para verificar que funcionó la última línea se puede escribir `bit` en la línea de comandos y presionar tabulador. Deberían aparecer varios comandos relacionados con bitbake que antes no estaban dado que no existen en el path de *Linux*.

Correr bitbake hasta la etapa de configuración:

```
MACHINE=beagleboard bitbake virtual/kernel -c configure
```

A.1.3. Paso 3 – Segunda compilación

Compilar imagen (esto compila tambien el rootfs):

```
MACHINE=beagleboard bitbake systemd-image
```
[2]

A.1.4. Paso 4 – OPCIONAL – configuración del kernel

El siguiente comando habilita un menú (CLI) de configuración del kernel. Se puede configurar por ejemplo habilitar soporte a ciertos *drivers* o la frecuencia del bus I2C.

```
MACHINE=beagleboard bitbake virtual/kernel -c menuconfig
```

A.1.5. Paso 5 – Compilación final

Compilar el kernel nuevamente (en teoría esto es necesario únicamente si se cambió la configuración del kernel):

```
MACHINE=beagleboard  bitbake virtual/kernel -c compile -f
```

A.1.6. Paso 6 – OPCIONAL – Recompilar

Recompilar kernel y bootloader (muchas veces luego de hacer todos los pasos anteriores, la beagle no bootea, lo siguiente puede ayudar):

```
MACHINE=beagleboard bitbake virtual/kernel -c clean
MACHINE=beagleboard bitbake virtual/bootloader -c clean

MACHINE=beagleboard bitbake virtual/kernel
MACHINE=beagleboard bitbake virtual/bootloader
```

A.1.7. Preparar tarjeta SD para instalar *Ångström*

Ejecutar el scrpit `mkcard2.sh` para darle formato a la tarjeta SD (sustituir mmcblk0 por el nombre que le asigne el sistema operativo a su tarjeta SD):

```
sudo ./mkcard2.sh /dev/mmcblk0
```

Este script (disponible en el repositorio *git* en la carpeta *scripts*) crea dos particiones en la tarjeta SD y las nombra *boot* y *Angstrom*. Para comprobar que los *file systems* fueron montados correctamente y se encuentran vacíos, ejecutar el comando `df -h`.

Posicionarse en el directorio donde se encuentran los archivos compilados de Angstrom:

[2] Este comando compila tambien el bootloader y el kernel

```
cd ~/setup-scripts/deploy/eglibc/images/beagleboard/
```

Copiar y renombrar los archivos de booteo en la primera partición (boot):

```
cp MLO-beagleboard-2011.09-r7 /media/federico/boot/MLO
cp u-boot-beagleboard-2011.09-r7.img\
/media/federico/boot/u-boot.img
cp uImage--3.2.28-r125b.2-beagleboard-20141007214437.bin \
/media/federico/boot/uImage
```

Descomprimir el root filesystem desde el .tar en la segunda partición (Angstrom):

```
sudo tar -xJv -C /media/federico/Angstrom/ -f \
Angstrom-console-image-eglibc-ipk-v2013.12-beagleboard.rootfs.tar.xz
```

Por último, desmontar las particiones de la tarjeta SD:

```
umount /media/federico/boot
umount /media/federico/Angstrom
```

A.2. Comunicación con la *beagleboard*

Para comunicarse con la *Beagleboard* se puede utilizar el puerto *RS232* de la misma o conectarse a través de una red TCP/IP. Por practicidad, durante el vuelo se realiza la conexión a través de *Wi-Fi*, utilizando el protocolo `ssh`.

Para esto hacer (asumiendo que la IP es `10.42.43.2`);

```
ssh root@10.42.43.2
```

El usuario *root* no requiere contraseña.

A.3. Compilación y ejecución del código

En esta sección se explica como obtener y ejecutar el código en la *Beagleboard*. Se incluyen también los pasos previos de configuración.

Preparar entorno de desarrollo

Preparar la *Beagleboard* para poder compilar el software:

```
opkg update
opkg upgrade
reboot
opkg update
opkg install task-native-sdk
```

Si se desea compilar modulos de kernel se deben obtener los fuentes y encabezados necesarios:

```
opkg install kernel-dev
opkg install kernel-headers
opkg install cmake
```

GIT

El software se encuentra en un repositorio en *github*, este se accede a través de la heramienta *git*.

Instalación de *git*:

```
opkg update
opkg install git
opkg install ca-certificates
```

Configurar *git* (se pone como ejemplo la configuración realizada en el presente proyecto):

```
git config --global user.name "Beagle_uQuad2"
git config --global user.email "federicofsv@gmail.com"
git config --global http.proxy http://proxy.fing.edu.uy:3128/
git config --global https.proxy https://proxy.fing.edu.uy:3128/
git config --global push.default simple
git config --global http.sslVerify true
```

Agregar la siguientes líneas en el archivo `~/.gitconfig`:

```
[http]
        sslVerify = true
        sslCAinfo = /etc/ssl/certs/ca-certificates.crt
```

Compilación y ejecución

Obtener el código y posicionarse en la carpeta `src`:

```
git clone https://github.com/ffavaro/uQuad2.git
cd uQuad2/src
```

El código utiliza *cmake*[3] para preparar la compilación. Se creó un *script* para automatizar el proceso, que se encarga de vaciar el contenido de la carpeta `uQuad2/src/build`, ejecutar *cmake* y copiar los archivos necesarios al directorio donde se encuentra el ejecutable del programa principal.

El *script* se encuentra en `uQuad2/src/` y se ejecuta de la siguiente manera:

```
./build_cmake.sh
```

Se implemetó otro *script* que se encarga de compilar el código y ejecutarlo. Este también se encarga de pasarle al programa principal los parámetros recibidos. Por ejemplo, en la simulación de vuelo hay que ejecutar (el *script* se encuentra en `uQuad2/src/`):

```
./start.sh <log_name> <throttle_inicial>
```

[3] http://www.cmake.org/

Referencias

[1] Rodrigo Alonso, Guzmán Hernández, and Pablo Iturralde. Automatización del vuelo de un avión. *Proyecto de Fin de Carrera de Ingeniería Eléctrica*, Facultad de Ingeniería, Universidad de la República, April 2009.

[2] Andrés Suttner Andrés Chappe, Rodrigo Sosa. Diseño, construcción y validación de un vehículo autónomo no tripulado (uav). *Proyecto de Fin de Carrera de Ingeniería Eléctrica*, Facultad de Ingeniería, Universidad de la República, 2007.

[3] Ward Brown. Brushless dc motor control made easy. Technical report, Microchip Technology Inc., 2002.

[4] García Carrillo, Dazul López, R Lozano, and C Pégard. *Quad Rotorcraft Control, Vision-Based Hovering and Navigation.* Springer ISBN 9781447143987.

[5] Davide Scaramuzza Christian Forster, Matia Pizzoli. Svo: Fast semi-direct monocular visual odometry. *IEEE International Conference on-Robotics and Automation (ICRA)*, 2014.

[6] Aeroquad Community. Aeroquad. `http://www.aeroquad.com/forum.php`.

[7] Ardupilot Community. Ardupilot. `http://copter.ardupilot.com/`.

[8] OpenPilot Community. Openpilot. `https://wiki.openpilot.org`.

[9] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[10] Ignacio Alonso Fernández-Coppel. La proyección utm. `http://www.cartesia.org/data/apuntes/cartografia/cartografia-utm.pdf`, febrero 2001.

[11] Bob Gross. Maxsonar operating on a multi-copter. http://www.maxbotix.com/articles/067.htm, 2013.

[12] Yu Gu Jason N. Gross and Matthew B. Rhudy. Robust uav relative navegation with dgps, ins, and peer-to-peer radio ranging. *IEEE Transactions on automation science and engineering*, 2014.

[13] Yucong Lin and Srikanth Saripalli. Path planning using 3d dubins curve for unmanned aerial vehicles. *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014.

[14] Ångstrom Project. Building Ångström. http://wp.angstrom-distribution.org/building-angstrom/.

[15] Bart De Moor Oscar Mauricio Agudelo. Computergestuurde regeltechniek exercise session. case study: Quadcopter. http://homes.esat.kuleuven.be/~maapc/static/files/CACSD/exercises/Session%203/quadcopter_exercise.pdf, 2014.

[16] Santiago Paternain, Rodrigo Rosa, and Matías Tailanián. Implementación de un UAV con arquitectura de cuadricóptero. *Proyecto de Fin de Carrera de Ingeniería Eléctrica*, Facultad de Ingeniería, Universidad de la República, Agosto 2012.

[17] J.B. Sousa P.B. Sujit, S. Saripalli. An evaluation of uav path following algorithm. *2013 European Control Conference (ECC)*, 2013.

[18] Andrei M. Shkel and Vladimir Lumelsky. Classification of the dubins set. *Robotics and Autonomous Systems 34*, 34:179–202, 2001.

[19] Tomoji Takasu and Akio Yasuda. Development of the low-cos rtk-gps receiver with an open source program package rtklib. *International Symposium on GPS/GNSS, International Convention Center Jeju, Korea, November 4-6, 2009.*

[20] Sander van Gameren and Patrick Wijnings. Design of a vision-enabled quadrocopter.

[21] LONG Teng YU Chenglong KOU Jiaxun WANG Zhu, LIU Li. Enhanced sparse a* search for uav path planning using dubins path estimation. *Proceedings of the 33rd Chinese Control Conference*, 2014.

[22] Xiong-Feng Zhu Jun-Tao Zhang Xian-Zhong Gao, Zhong-Xi Hou and Xiao-Qian Chen. The shortest path planning for manoeuvres of uav. *Acta Polytechnica Hungarica, Vol. 10, No. 1, 2013*, 2013.

Índice de tablas

Índice de figuras