



UNIVERSIDAD AUTÓNOMA DEL ESTADO DE
HIDALGO

INSTITUTO DE CIENCIAS BÁSICAS E
INGENIERÍA

ESTIMACIÓN DE ESTADO DE UN
CUADRICÓPTERO

TESIS

QUE PARA OBTENER EL TÍTULO DE:

**Maestro en Ciencias en Automatización
y Control**

PRESENTA:

Ing. Andrés Ramírez García

ASESORES:

Dr. Hugo Romero Trejo

Dr. Omar Jacobo Santos Sánchez



Pachuca de Soto Hgo.,

febrero 2018

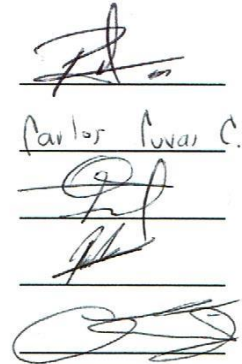


Mineral de la Reforma, Hgo., a 24 de enero de 2018
 Oficio No. MCAC05_2018

Ing. Andrés Ramírez García
 PRESENTE

Por medio de la presente y en mi calidad de coordinador de la Maestría en Ciencias en Automatización y Control, del Área Académica de Computación y Electrónica (AACyE) de la Universidad Autónoma del Estado de Hidalgo (UAEH), me es grato informarle que el Jurado asignado para la revisión de su trabajo de tesis titulado: **“Estimación de estado de un cuadricóptero”**, dirigido por el Dr. Hugo Romero Trejo y el Dr. Omar Jacobo Santos Sánchez, que para obtener el grado de Maestro en Ciencias en Automatización y Control fue presentado por usted, ha tenido a bien en reunión de sinodales, autorizarlo para impresión. A continuación se integran las firmas de conformidad de los integrantes del Jurado:

Dr. Jesús Patricio Ordaz Oliver	(Presidente)	UAEH
Dr. Carlos Cuvas Castillo	(Secretario)	UAEH
Dr. Hugo Romero Trejo	(Vocal 1)	UAEH
Dr. Pablo Antonio López Pérez	(Vocal 2)	UAEH
Dr. Omar Jacobo Santos Sánchez	(Vocal 3)	UAEH



Atentamente
 “Amor, Orden y Progreso”



Dr. Jesús Patricio Ordaz Oliver
 Coordinador de la Maestría en Ciencias en Automatización y Control
 Universidad Autónoma del Estado de Hidalgo

c.c.p. Dr. Dr. Oscar Rodolfo Suárez Castillo, Director del Instituto de Ciencias Básicas e Ingeniería
 c.c.p. Dr. Hugo Romero Trejo, Jefe del Área Académica de Computación y Electrónica
 c.c.p. Expediente/ apl



Ciudad del Conocimiento
 Carretera Pachuca - Tulancingo km. 4.5
 Colonia Carboneras
 Mineral de la Reforma, Hidalgo, México, C.P. 42184
 Tel. +52 771 7172000 exts. 2250 y 2251
jesus_ordaz@uaeh.edu.mx

www.uaeh.edu.mx

"Lo que sabemos es una gota de agua; lo que ignoramos es el océano"

Isaac Newton

Agradecimientos

Existen muchos retos en la vida y muchas metas por alcanzar, pero no hay retos que no puedan ser superados, ni metas inalcanzables. Gracias a mi familia, es que puedo pensar de tal forma, gracias a la educación que me brindaron mis padres soy la persona que soy y no habría elegido un camino diferente. Aún tengo muchos defectos que superar, muchas metas que alcanzar y retos que librar, pero puedo seguir avanzando debido al apoyo incondicional de mi familia. Con este trabajo alcanzo una nueva meta y por ello, les doy gracias.

A mis directores de tesis, agradezco que me compartieron sus conocimientos y su apoyo para llevar a cabo este proyecto. Asimismo agradezco a mis sinodales por su colaboración y sus aportaciones.

A mis compañeros y amigos agradezco su apoyo y que compartieran sus conocimientos así como sus palabras de aliento para salir adelante.

Agradezco al Consejo Nacional de Ciencia Y Tecnología (CONACYT) por el apoyo económico a través de la beca de maestría otorgada al CVU 701645.

Resumen

Con el acelerado avance de la tecnología, los dispositivos encargados de medir o sensor los fenómenos físicos, han reducido considerablemente su tamaño, al igual que su costo, sin embargo, en muchos de los casos aún tienen la desventaja de una baja precisión y alta susceptibilidad al ruido. En el presente trabajo se muestra el diseño de un algoritmo estimador de estado, capaz de reducir los errores de posicionamiento espacial de una plataforma robótica aérea, ocasionados por la limitada precisión de los sensores utilizados, ya que son de bajo costo y de uso civil. Puntualizando, los dispositivos sensores considerados, son el sistema de posicionamiento global (GPS) y el barómetro, los cuales nos proporcionan el posicionamiento traslacional y la altitud de un cuadricóptero respectivamente. Al ser las mediciones entregadas por los sensores, previamente mencionados, limitadas en su precisión y por consecuencia poco confiables para ser alimentadas directamente a un algoritmo de control para la estabilización en posición del robot, se propone que mediante una estrategia que combina un modelo autorregresivo (ARX) con el algoritmo de mínimos cuadrados recursivo y en conjunto con un control proporcional derivativo (PD) se logre una correcta estabilización de la plataforma robótica. Para la implementación de este algoritmo de estimación de estado y control, se proyecta su programación en el autopiloto de arquitectura abierta PixHawk, que permite un fácil desarrollo y flexibilidad.

Abstract

With the exponential increase of technology, the devices to sense the physical phenomena have reduced their size considerably likewise their cost. Nevertheless it has disadvantages like low precision and high sensitivity to noise. Due to the low precision of the global positioning system (GPS) and the barometer, which together provide the spatial position of a quadrotor. In the present paper the design of an algorithm capable of reducing the altitude positioning error is shown. The algorithm proposed consists of the combination of an autoregressive (ARX) model with recursive least squares (RLS) to estimate the state of the quadrotor system. To hover the quadrotor system, a proportional and derivative (PD) controller is used, subdivided in x , y and z displacement. The algorithm was implemented on the Pixhawk autopilot.

Índice general

Agradecimientos	IV
Resumen	V
Abstract	VI
Índice general	VII
Índice de figuras	IX
Índice de tablas	X
1. Introducción	1
1.1. Antecedentes	4
1.2. Justificación	6
1.3. Objetivos	6
1.4. Planteamiento del problema	7
1.5. Hipótesis	8
1.6. Método	8
1.7. Marco Teórico	9
1.7.1. Modelo dinámico	10
1.7.2. Estimación de estado	14
1.8. Propuesta de solución	22
1.9. Conclusiones	23
2. Plataforma experimental	24
3. Resultados experimentales	30
3.1. Simulación	30
3.1.1. Simulación del controlador PD	30
3.1.2. Simulación del algoritmo RLS	35
3.2. Programación	40
3.3. Implementación	45

4. Conclusiones y trabajos futuros	49
A.	50
Bibliografía	64

Índice de figuras

1.1.	Tipos de multirrotores [21].	3
1.2.	Marco de referencia inercial [3].	13
1.3.	Función de transferencia discreta del sistema incluyendo todas las entradas y perturbaciones al sistema.	16
1.4.	Visualización de la estimación recursiva como un proceso iterativo [12].	19
2.1.	Parrot AR.Drone 2.0 con protección para interior.	25
2.2.	Plataforma experimental cuadricóptero.	27
2.3.	Diagrama de conexión [25].	29
3.1.	Diagrama a bloques del control del cuadricóptero.	31
3.2.	Posiciones x, y, z del cuadricóptero.	33
3.3.	Gráfica del control para x, y, z del cuadricóptero.	34
3.4.	Posiciones x, y, z del cuadricóptero con diferentes ganancias.	34
3.5.	Gráfica del control para x, y, z del cuadricóptero con diferentes ganancias.	35
3.6.	Comparación entre las lecturas reales de ϕ y las estimadas, obtenidos a partir de RLS, con condiciones iniciales diferentes de cero.	38
3.7.	Comparación entre las lecturas reales de ϕ y las estimadas, obtenidos a partir de RLS, con condiciones iniciales cero.	38
3.8.	Comparación entre las lecturas reales de z y las estimadas, obtenidos a partir de RLS.	39
3.9.	<i>Software</i> Eclipse.	40
3.10.	<i>Software</i> Mission Planner.	41
3.11.	Comparación entre RLS, lecturas del barómetro y el promedio ponderado.	45
3.12.	Comparación entre la estimación y la deriva del barómetro.	46
3.13.	Señal del barómetro comparada con la estimación y la referencia de control.	47
3.14.	Acercamiento de la comparación entre la estimación y la señal del barómetro.	48

Índice de tablas

3.1. Comparativa entre las diferentes medias del error.	47
---	----

Capítulo 1

Introducción

A lo largo de la historia, el hombre ha buscado la forma de transportarse mediante vehículos que facilitan su traslado; a través de los años, estos vehículos han evolucionado hasta el punto de poder “manejarse solos”, es decir, ya no necesitan una persona para poder navegar y se denominan vehículos no tripulados o autónomos. Los vehículos aéreos no tripulados (UAVs) presentan una gran cantidad de dinámicas, razón por la cual han llamado la atención en diversas áreas académicas.

Un sistema UAV está compuesto generalmente de dos partes, la estación en tierra y la parte aérea. La estación en tierra asegura la preparación de la misión y la comunicación con la parte aérea y con los sistemas que controlan y coordinan el UAV [1]. Estos últimos sistemas pueden presentarse en una configuración todo integrado, como son los autopilotos, de los cuales, un ejemplo es el PixHawk de 3D Robotics y que es una plataforma de arquitectura abierta, donde se puede programar una estrategia de control, que en combinación con el conjunto de sensores, doten de un nivel de autonomía a la plataforma robótica aérea. Es así que para el control de los sistemas dinámicos, la estrategia más aplicada es el controlador Proporcional-Integral-Derivativo (PID); el cual requiere de la sintonización de sus ganancias, que si es hecha adecuadamente, se obtendrá un desempeño apropiado del UAV en la realización de un vuelo estacionario o “hover” (como es denominado en inglés).

Debido a la versatilidad de este tipo de plataformas aéreas, los UAV’s pueden ser utilizados como dispositivos de reconocimiento, observación, adquisición o recolección

de datos; susceptible de llevar diferentes sensores, que le servirán para efectuar diferentes tareas durante un vuelo y que pueden variar en función de sus capacidades [2]. La mayoría de los Sistemas de Navegación Inercial (INS por sus siglas en inglés) de estos vehículos, emplean sensores con principios y tecnologías basadas en sistemas Micro-Electro-Mecánicos (MEMS por sus siglas en inglés), que son ligeros, de bajo costo y con bajo consumo de energía [18]. Sin embargo, los giroscopios MEMS son menos precisos, tienen grandes desviaciones (acumulación de error). Adicionalmente, son más ruidosos y mucho más sensibles a los fenómenos ambientales en comparación a otros como pueden ser los laser y de fibra óptica. Además, de los INS basados en MEMS no se genera una medición de la orientación que pueda tomarse directamente, es por eso que para mejorar el rendimiento y la robustez de estos sistemas, se requiere la aplicación de un filtrado no lineal y de la fusión de datos para lograr una mejor estimación de las variables angulares en el UAVs [18].

Con el filtrado no lineal y la fusión de datos es posible obtener información y mediciones confiables de sensores y sistemas inciertos y/o ruidosos. Otro dispositivo ampliamente utilizado en la navegación de vehículos es el Sistema de Posicionamiento Global (GPS por sus siglas en inglés), que al igual que los MEMS tienen una precisión limitada y además entregan mediciones que oscilan, en el mejor de los casos, en un rango de 2.5 a 3m. En razón de lo anterior, se han empleado diversas técnicas para estimar la posición “real” del UAV, ya sea con sensores más precisos (pero más costosos), o con técnicas alternativas y/o complementarias como lo es el flujo óptico, el cual emplea una cámara, o también con algoritmos matemáticos como el Filtro de Kalman Extendido (EFK), Mínimos Cuadrados Recursivos (RLS), el método del observador no lineal invariante [19] o modelos autorregresivos (ARX) [9, 20]. Adicionalmente, estas técnicas de estimación de variables encuentran una aplicación muy pertinente cuando se realizan vuelos en exterior, donde los sistemas robóticos móviles son muy susceptibles de experimentar perturbaciones propias del ambiente, y de las cuales sus efectos son minimizados a través de la aplicación de estas técnicas o la combinación de algunas de ellas.

Actualmente, existen una gran variedad de UAV's tanto para aplicaciones militares, civiles y de esparcimiento. Dentro de ellos podemos encontrar plataformas de

diversos tamaños, ya sea de ala fija o rotatorias y que han sido desarrolladas con base a las aplicaciones en las cuales se utilizaran. Dentro de las de ala rotatoria se encuentran muchas configuraciones que varían en el número de propelas utilizadas y su disposición geométrica. Todas ellas pueden ser impulsadas a través de sistemas de combustión o eléctricos, siendo los primeros aquellos sistemas que son requeridos para largos tiempos de vuelo o en donde se requiere una mayor capacidad de carga útil. En cuanto al número de rotores, los UAV que poseen 4, 6 u 8 y que son de talla reducida, actualmente son los más populares (Figura 1.1), los cuales, en su gran mayoría, involucran una mecánica simple que permite hacer reparaciones rápidas y de bajo costo. En este trabajo se utiliza un mini helicóptero de 4 rotores identificado en la literatura como cuadrirrotor, el cual es un sistema dinámico subactuado que posee 6 grados de libertad (DOF por sus siglas en inglés) dado que evoluciona en el espacio y que solamente tiene 4 entradas de control independientes [2].

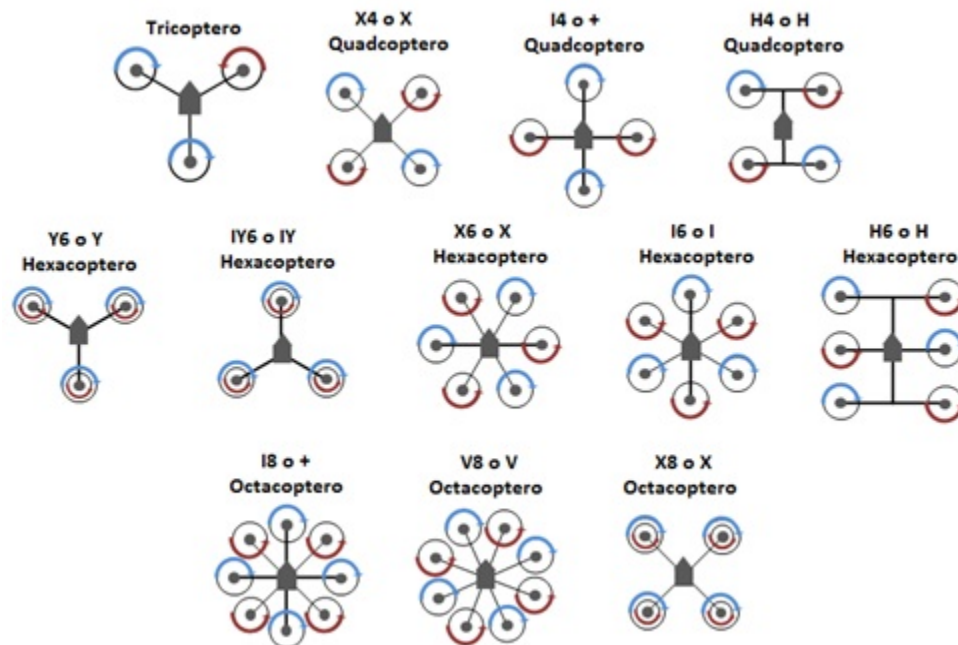


Figura 1.1: Tipos de multirrotores [21].

1.1. Antecedentes

Uno de los principales retos a vencer, para estabilizar un cuadricóptero en exterior, son las condiciones climáticas ya que en muchas de las tareas en las que se utiliza, debe mantenerse en vuelo estacionario o *hover*. Para ello, es esencial conocer el estado completo del cuadricóptero, contando con mediciones o estimaciones precisas del mismo, además de diseñar un control robusto capaz de lidiar con cualquier perturbación o dinámicas no modeladas. Es decir, se requiere contar con un conjunto de sensores confiables o estrategias de filtrado y/o fusión de datos adecuadas que permitan al control realizar su tarea de manera efectiva y por lo tanto el robot aéreo tenga el desempeño deseado.

En lo que a técnicas de control se refiere, existen sistemas de control que se combinan con sensores especializados, por ejemplo, en [3] se presenta una configuración que tiene ocho rotores, cuatro están dedicados a la estabilización de la orientación del helicóptero, y los otros cuatro se utilizan para conducir los desplazamientos laterales. También se ha introducido una precompensación en los ángulos de alabeo (roll) y cabeceo (pitch) la cual está directamente relacionada con la velocidad de los motores laterales. Respecto a la estimación de estado del cuadricóptero, propone utilizar las mediciones de la técnica de flujo óptico para ambientes interiores y en lo que respecta a vuelos en exterior utilizan mediciones del GPS, logrando un vuelo estacionario que es robusto respecto a las perturbaciones del viento. Finalmente, concluyen que bajo ciertas condiciones, la técnica de flujo óptico proporciona mediciones más precisas que las que el GPS. Igualmente en [4] se aborda el problema de la estabilización y el posicionamiento de un cuadricóptero utilizando un sistema de visión, compuesto por una cámara, para controlar su movimiento, combinado con las mediciones de una IMU para estimar la orientación y la posición del cuadricóptero. Sin embargo, cabe mencionar que la aplicación de algunas de estas técnicas se realizan bajo condiciones controladas en ambientes cerrados, además, cuando se utilizan sensores extras, produce un cambio de peso en el sistema y por consiguiente reduce el tiempo de vuelo, asimismo, el sensor de flujo óptico puede tener ciertas restricciones para su uso en exterior.

Los sensores han evolucionado a tal grado que su tamaño se ha reducido considerablemente, sin embargo, siguen teniendo una baja precisión y resolución, características que pueden afectar severamente el desempeño de un sistema tan complicado de controlar como son los cuadricópteros, por tal motivo, el uso de sensores, en combinación con algoritmos matemáticos es una opción a considerar. Por ejemplo en [5] emplean algoritmos para estimar simultáneamente la altitud del vehículo y la elevación, mediante una técnica llamada Localización y Mapeo Simultáneos (SLAM por sus siglas en inglés) en conjunto con una IMU, para aumentar la precisión de la estimación, sin embargo, esta técnica solo se puede utilizar en ambientes interiores debido a la naturaleza del sensor láser. Si reducimos la cantidad de sensores a bordo del UAV, con la intención de reducir su peso, es posible calcular las salidas que no conocemos, utilizando un algoritmo que estime las variables de estado desconocidas, por ejemplo, en [7] se proponen dos métodos para la estimación de la velocidad de traslación de un UAV, basados sólo en los sensores de a bordo. En este caso, entre los sensores se encuentra una cámara, la cual, en conjunto con las mediciones de una IMU, proporcionan los datos que alimentan los estimadores. Un algoritmo propuesto en la literatura, basado en un observador no lineal, fue diseñado usando la técnica de Lyapunov [6], asimismo, en otros documentos reportan el uso de algoritmos donde se utiliza la estrategia del filtro de Kalman (Kalman Filter) en sus diversas variedades [7]. Otro método ampliamente utilizado, en el campo del control de auto-adaptación, es el de mínimos cuadrados. Con la aplicación de todos ellos, se busca mejorar la tarea del control del sistema. En algunos casos se reporta solo resultados de simulación de los diversos algoritmos desarrollados, tal es el caso de [8]. En algunos otros llevan sus trabajos hasta la implementación sobre alguna plataforma experimental como en [9], donde se aborda el problema de la navegación autónoma de un cuadricóptero usando estimadores de estado cuando la señal GPS no está disponible, el cual se realiza mediante un observador que utiliza una aproximación de la doble integral de la aceleración, combinado con el uso del algoritmo recursivo de mínimos cuadrados para obtener mejores estimaciones.

Es importante enfatizar que los MEMS por si solos no son completamente eficientes para determinar la posición y orientación de un UAV, sin embargo, si se combinan

dos o mas de ellos en el mismo sistema, en conjunto con algoritmos de estimación se puede reducir el error inherente a estos dispositivos obteniendo mejores mediciones.

1.2. Justificación

Los cuadricópteros son robots aéreos que presentan muchas ventajas, como ya se mencionó anteriormente, sin embargo, es deseable contar con el estado completo del sistema, ya sea que se tenga la posibilidad de medirlo directamente o a través de estimadores, es decir, conocer en todo momento la posición y orientación del cuadricóptero, lo que facilitara la implementación de una técnica de control que le permita al UAV realizar sus tareas de forma adecuada y eficiente. Es muy común utilizar un GPS para determinar la posición relativa de un cuadricóptero, así como un barómetro para determinar la altura, sin embargo, estos dispositivos tienen un error de precisión en sus mediciones, por lo que es necesario desarrollar un algoritmo que reduzca este error. En un aspecto más profundo, resulta favorable la combinación entre los sensores de abordo del cuadricóptero con algoritmos de estimación. En el presente trabajo se ha optado por utilizar el método de Mínimos Cuadrados Recursivo (RLS por sus siglas en inglés) para la estimación de estado del cuadricóptero, y ya que el algoritmo se programa en el autopiloto que controla el cuadricóptero, presenta ciertas ventajas en su aplicación, por ejemplo, evita incluir más sensores, reduciendo el consumo de energía y el peso del cuadricóptero, logrando así un aumento en el tiempo de vuelo; también, navegar en caso de que la señal del GPS se pierda momentáneamente y de igual forma, reducir el error de posición y orientación inherente a los MEMS. En relación al algoritmo utilizado, si se compara con otros, el RLS es relativamente más sencillo, lo que implica, un menor consumo de la capacidad de procesamiento del autopiloto, al igual que evita usar una base en tierra para el procesamiento de datos y por consiguiente evita los retardos que conlleva la comunicación inalámbrica.

1.3. Objetivos

Los objetivos del presente proyecto, se pueden dividir en:

- **Objetivo general**

Reducir el error de posicionamiento del cuadricóptero en vuelo estacionario, debido a la baja precisión de los sensores de bajo costo utilizados, mediante la estimación de estado utilizando el algoritmo RLS para la estimación de los parámetros del modelo no lineal del sistema logrando así la correcta estabilización en posición del sistema.

- **Objetivos particulares**

- Diseñar un estimador en base al algoritmo RLS, que permita identificar los parámetros del sistema dinámico para su correcta estabilización.
- Estimar el vector de posición del cuadricóptero, por medio del algoritmo RLS para reducir el error asociado con la precisión de los dispositivos GPS comerciales de bajo costo.
- Realizar un estudio de los diferentes controladores de vuelo y autopilotos disponibles en el mercado para identificar el idóneo en cuanto a prestaciones y costo para ser aplicado en la plataforma experimental.
- Programar los algoritmos de estimación y control en el dispositivo a bordo seleccionado para el control de altura del mini helicóptero.

1.4. Planteamiento del problema

Los cuadricópteros son sistemas de múltiples entradas y múltiples salidas (MIMO), además, son subactuados, es decir, tienen más grados de libertad que entradas de control. En este caso particular se tienen seis grados de libertad y solamente cuatro entradas; y es ésta una de las razones que su control se torna difícil, aunada a la presencia de dinámicas de alta velocidad y baja fricción que se tiene en este tipo de sistemas. Adicionalmente a lo anterior, aparecen problemas referentes con la instrumentación de la plataforma, y que están directamente relacionados con los sistemas sensores y su nivel de precisión y repetitibilidad, haciendo complicado el correcto control de la plataforma y más si es para la ejecución de vuelos en exterior en donde

la elección de una buena estrategia de control también es un punto relevante a considerar y que debe de tener como cualidades la capacidad de rechazar o minimizar las perturbaciones externas entrando al sistema robótico y sensorial.

Para que el cuadricóptero navegue o realice un vuelo estacionario, es necesario tomar en cuenta las consideraciones anteriores, sin embargo, es de vital importancia conocer el estado completo del sistema. Para llevar a cabo esta tarea, se presentan múltiples opciones, ya sea utilizar sensores mas precisos o especializados, pero que al tener estas características son mucho más costosos, y que al mismo tiempo pueden aumentar considerablemente el peso a transportar por el helicóptero y que a su vez reducirán el tiempo de vuelo. Es claro que para UAV's de talla pequeña, el uso de sensores de peso considerable no es opción a considerar y por lo tanto todos ellos deben de ser ligeros como una condicionante a cumplir primeramente. La estrategia es usar una cantidad limitada de sensores ligeros y que a pesar de tener una baja precisión al combinarlos con algoritmos matemáticos se pueda tener una estimación muy aceptable de las variables de estado de interés. Para la selección e implementación de estos algoritmos se debe considerar su grado de complejidad y su costo computacional, los cuales deben de ser relativamente moderados de tal manera que permitan su programación y ejecución en el autopiloto elegido.

1.5. Hipótesis

Mediante la aplicación de un observador de estado en lazo abierto es posible realizar la estimación precisa de las variables de posición de un mini helicóptero utilizando las mediciones provistas por el GPS y un barómetro, reduciendo así los errores inherentes de estos sensores y consecuentemente alcanzando la correcta estabilización de la plataforma.

1.6. Método

Para llevar a cabo el objetivo planteado se utilizará una estrategia de control basada en un controlador PD con una sintonización heurística, similar a la que se

realiza en [10]. Así como la implementación de un estimador, basado en el algoritmo RLS, para determinar los parámetros del sistema no lineal. Como primera etapa, se realizará una simulación, en el *software* MATLAB, para entender las dinámicas del cuadricóptero y obtener una primera aproximación de las ganancias del controlador. Asimismo, a partir de datos obtenidos previamente, simular el proceso de estimación, mediante el algoritmo RLS y comprobar el correcto funcionamiento. Finalmente, se utilizará como plataforma, un cuadricóptero con el autopiloto PixHawk de la marca 3DR, donde se programarán los algoritmos mencionados, para estimar la posición del cuadricóptero.

1.7. Marco Teórico

En esta sección se abordaran los temas específicos para resolver el problema planteado anteriormente. En primera instancia debemos conocer el modelo matemático, que describa las dinámicas del cuadricóptero, sin embargo, para modelar el sistema se hacen ciertas consideraciones, o se asume que las condiciones del sistema son ideales. Por estas razones el modelo es una aproximación de nuestro sistema real.

La representación dinámica de un objeto volador es, por supuesto, uno de los objetivos principales que deben resolverse antes que el desarrollo de las estrategias de control. Se mencionan tres enfoques para el modelado de un objeto volador (enfoques newtoniano, lagrangiano y cuaternión). El objeto volador se considera como un objeto sólido que se mueve en un entorno 3D, sometido a fuerzas y pares de torsión, aplicados al cuerpo, en función del tipo de objeto considerado. El modelo dinámico se utiliza para expresar y representar el comportamiento del sistema en el tiempo. Respecto al enfoque de Newton-Euler considera que un cuerpo rígido es un sistema de partículas en el que las distancias entre las partículas no varían. Nos encontramos en la literatura diferentes formas de presentar las dinámicas de cuerpos rígidos en movimiento en el espacio 3D. El enfoque de Newton-Euler y de Euler-Lagrange son los más destacados. El método de Newton-Euler se utiliza en primer lugar para desarrollar la dinámica del cuerpo del objeto y después representar la estructura en el sistema de referencia inercial. Después de estas manipulaciones, expresamos estas dinámicas utilizando el

método de Euler-Lagrange [2].

Si se puede obtener un modelo matemático de la planta, es posible aplicar diversas técnicas de diseño con el fin de determinar los parámetros del controlador que cumpla las especificaciones del transitorio y del estado estacionario del sistema en lazo cerrado. Sin embargo, si la planta es tan complicada que no es fácil obtener su modelo matemático, tampoco es posible un método analítico para el diseño de un controlador PID. En este caso, se debe recurrir a procedimientos experimentales para la sintonía de los controladores PID [11].

Así como determinar el modelo y la estrategia de control que se utilizará, es necesario estimar el estado del cuadricóptero para proveer la posición al algoritmo de control, para esto, también se aborda el desarrollo del algoritmo RLS.

1.7.1. Modelo dinámico

En esta sección se abordará la descripción del modelo matemático del cuadricóptero mediante las ecuaciones de Euler-Lagrange como en [2]. El modelo matemático nos ayudará posteriormente para lograr la estimación de la posición, velocidad y para el diseño del controlador. El modelo matemático de un sistema real se puede definir como el conjunto de ecuaciones diferenciales que denotan el comportamiento de la dinámica propia del sistema [11].

Para obtener el modelo del cuadricóptero se utilizan las ecuaciones de Euler-Lagrange como se describen en [2]. El estado general para un sistema de este tipo, se puede expresar con el siguiente vector:

$$q = (x, y, z, \psi, \theta, \phi) \in \mathbb{R}^6$$

donde $(x, y, z) \in \mathbb{R}^3$ denota la posición tomada del centro de masa del cuadricóptero, relativo al eje de referencia inercial I, $(\psi, \theta, \phi) \in \mathbb{R}^3$, son los ángulos de Euler, ψ es el ángulo de yaw, θ es el ángulo de pitch y ϕ es el ángulo de roll, estos ángulos representan la orientación del vehículo.

El modelo es particionado en dos partes (ξ, η) , por un lado la parte traslacional y

por el otro la parte rotacional.

$$\xi = (x, y, z) \in \mathbb{R}^3, \quad \eta = (\psi, \theta, \phi) \in \mathbb{R}^3$$

Ahora, defina el siguiente Lagrangiano:

$$L(q, \dot{q}) = T_{trans} + T_{rot} - U$$

donde T_{trans} denota la energía cinética traslacional, T_{rot} denota la energía cinética rotacional y U la energía potencial. La ecuación de energía cinética traslacional y rotacional del cuadricoptero es:

$$T_{trans} \triangleq \frac{m}{2} \dot{\xi}^T \dot{\xi} \tag{1.7.1}$$

$$T_{rot} \triangleq \frac{1}{2} \dot{\omega}^T I \dot{\omega} \tag{1.7.2}$$

donde m es la masa del vehículo, ω es la velocidad angular del vehículo, I es la matriz de inercia. El vector de velocidades angulares ω respecto a los ejes de coordenadas del cuerpo se relaciona con las velocidades generalizadas del vector $\dot{\eta}$, en la región donde los ángulos de Euler son válidos, se hace uso de una relación estándar de cinemática [2], como sigue:

$$\dot{\eta} = W_{\eta}^{-1} \omega,$$

en donde:

$$W_{\eta}^{-1} = \begin{bmatrix} -\sin \theta & 0 & 1 \\ \cos \theta \sin \psi & \cos \psi & 0 \\ \cos \theta \cos \psi & -\sin \psi & 0 \end{bmatrix}.$$

Defina $\mathbb{J} = W_{\eta}^T I W_{\eta}$, tal que:

$$T_{rot} \triangleq \dot{\eta}^T \mathbb{J} \dot{\eta}$$

la matriz \mathbb{J} actúa como la matriz de inercia para la energía cinética rotacional del helicóptero, expresada en términos de coordenadas generalizadas η . Finalmente la energía potencial que debe ser considerada es:

$$u = mgz$$

en donde z representa la altura del vehículo y g es la aceleración gravitacional. El modelo dinámico completo del cuadricóptero se obtiene de las ecuaciones de Euler-Lagrange con fuerzas externas generalizadas [2].

$$\frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{q}} - \frac{\partial \mathcal{L}}{\partial q} = F$$

donde el vector de fuerzas externas generalizada F , está formada por $F = (F_\xi, \tau)$, F_ξ es la fuerza translacional aplicada al vehículo debido a la entrada de control principal, por lo tanto es posible expresar lo siguiente: $\hat{F} = [0, 0, u]$ y $u = f_1 + f_2 + f_3 + f_4$, donde:

$$f_i = k_i \omega_i^2 \quad \forall i = 1, \dots, 4, \quad k_i > 0$$

k_i es una constante y ω_i es la velocidad angular del motor i -ésimo. Por lo tanto $F_\xi = R\hat{F}$ donde R es la matriz de rotación que representa la orientación del cuadricóptero relacionada al eje de referencia fijo y está denotada por $R(\psi, \theta, \phi)$: (aquí se usa la siguiente notación c_θ para $\cos \theta$ y s_θ para $\sin \theta$).

$$R = \begin{bmatrix} c_\theta c_\psi & s_\psi s_\theta & -s_\theta \\ c_\psi s_\theta s_\phi - s_\psi c_\theta & s_\psi s_\theta c_\phi + s_\psi c_\theta & c_\theta s_\phi \\ c_\psi s_\theta s_\phi + s_\psi c_\theta & s_\psi s_\theta c_\phi - s_\psi c_\theta & c_\theta c_\phi \end{bmatrix}.$$

Los momentos o pares generalizados encargados de producir los movimientos angulares de roll(ϕ), pitch (θ) y yaw (ψ) respectivamente son definidos como:

$$\tau = \begin{bmatrix} \tau_\psi \\ \tau_\theta \\ \tau_\phi \end{bmatrix} \triangleq \begin{bmatrix} \sum_{i=1}^4 \tau M_i \\ (f_2 - f_4)l \\ (f_3 - f_1)l \end{bmatrix}$$

donde l es la distancia entre los motores y el centro de gravedad del cuadrirrotor y M_i es el momento producido por el motor $i = 1, \dots, 4$, alrededor del centro de gravedad como es mostrado en la Figura 1.2. Debido a que el lagrangiano no contiene términos en la energía cinética combinando $\dot{\xi}$ con $\dot{\eta}$ como se muestra en las ecuaciones (1.7.1-1.7.2), al ser divididas las dinámicas se tiene:

$$m\ddot{\xi} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = F_\xi, \quad (1.7.3)$$

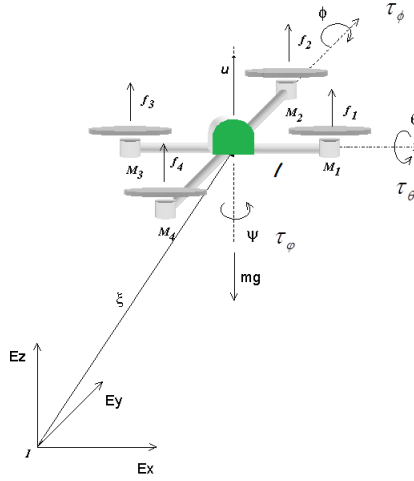


Figura 1.2: Marco de referencia inercial [3].

$$\mathbb{J}\ddot{\eta} + \dot{\mathbb{J}}\dot{\eta} - \frac{1}{2} \frac{\partial}{\partial \eta} (\eta^T \mathbb{J} \dot{\eta}) = \tau.$$

Defina los términos del vector de Coriolis que tienen efectos centrífugos y giroscopios asociados al vector η como:

$$C(\eta, \dot{\eta}) = \dot{\mathbb{J}}\dot{\eta} - \frac{1}{2} \frac{\partial}{\partial \eta} (\dot{\eta}^T \mathbb{J} \dot{\eta}),$$

si se reescribe el sistema (1.7.3) se obtiene

$$m\ddot{\xi} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = F_{\xi}, \quad (1.7.4)$$

$$\mathbb{J}\dot{\eta} + C(\eta, \dot{\eta}) = \tau.$$

Con la finalidad de simplificar el sistema, se propone el siguiente cambio de variables:

$$\tau = C(\eta, \dot{\eta})\dot{\eta} + \mathbb{J}\tilde{\tau},$$

donde $\tilde{\tau} = [\tilde{\tau}_{\psi}, \tilde{\tau}_{\theta}, \tilde{\tau}_{\phi}]$ son las nuevas entradas y por lo tanto,

$$\ddot{\eta} = \tilde{\tau}.$$

Por lo tanto el sistema puede reescribirse como:

$$m\ddot{\xi} + mgz = F_\xi \tag{1.7.5}$$

$$\ddot{\eta} = \tilde{\tau},$$

donde F_ξ está definido como:

$$F_\xi = \begin{bmatrix} -f \sin \theta \\ f \sin \phi \cos \theta \\ f \cos \phi \cos \theta \end{bmatrix} \tag{1.7.6}$$

Al sustituir (1.7.6) en (1.7.5), finalmente se obtiene:

$$\begin{aligned} m\ddot{x} &= -u \sin \theta \\ m\ddot{y} &= u \cos \theta \sin \phi \\ m\ddot{z} &= u \cos \theta \cos \phi - mg \\ \ddot{\phi} &= \tau_\phi \\ \ddot{\theta} &= \tau_\theta \\ \ddot{\psi} &= \tau_\psi \end{aligned} \tag{1.7.7}$$

donde x y y son coordenadas en el plano horizontal, z es la posición vertical [2]. El ángulo alrededor del eje z es yaw ψ , alrededor de eje x es el ángulo de pitch θ y finalmente alrededor del eje y es roll ϕ . Las entradas de control $\tilde{\tau}_\psi, \tilde{\tau}_\theta, \tilde{\tau}_\phi$ son los momentos de yaw, pitch y roll respectivamente.

1.7.2. Estimación de estado

Para realizar el control de un sistema es necesario tener presente todo el estado que describe la dinámica del sistema. En la práctica, estas variables de estado, en muchos de los casos no es posible tener acceso a ellas, ya sea por el costo de los sensores, o por que no existe uno que sea capaz de tomar las mediciones [13]. En este caso es posible diseñar un estimador de estado para obtener de manera indirecta dicha variable de estado. En las siguientes secciones se abordará el método de estimación basado en el

algoritmo RLS, para poder construir las variables de estado, las cuales en este caso son las posiciones del cuadricóptero.

Observadores de estado

En los métodos teóricos para el diseño de sistemas de control, es común suponer que todas las variables de estado están disponibles para su realimentación. Sin embargo, en la práctica no todas las variables de estado están accesibles para poder realimentarse. Entonces, se necesita estimar las variables de estado que no están disponibles. Un dispositivo que estima u observa las variables de estado se llama observador de estado, o, simplemente, un observador. Si el observador de estado capta todas las variables de estado del sistema, sin importar si algunas están disponibles por medición directa, se denomina observador de estado de orden completo [11].

Un observador de estado estima las variables de estado basándose en las mediciones de las variables de salida y de control. Por lo tanto, los observadores de estado pueden diseñarse si y sólo si se satisface la condición de observabilidad.

Observador lineal en lazo Cerrado. El observador en lazo cerrado es utilizado cuando se conoce la evolución de la entrada y salida del sistema. En general para este tipo de observador es necesario conocer la evolución de la salida $y(t)$ y entrada $u(t)$ en un intervalo de tiempo para poder calcular el estado del sistema, ya que se basa en la realización de una copia del sistema original $\dot{\hat{x}} = A\hat{x} + Bu$ y calcular un error de estimación $e(t) = x(t) - \hat{x}(t)$ para llevar el estado $\hat{x}(t) \rightarrow x(t)$. Para poder aplicar dicho esquema de observación y conocer las variables de estado es necesario que el sistema:

$$\begin{aligned}\dot{x} &= Ax + Bu, \\ y &= Cx + Du\end{aligned}$$

sea observable, un sistema es observable si y sólo si la matriz de observabilidad tiene rango n es decir que el rango de dicha matriz y el orden del sistema tenga el

mismo tamaño [13].

$$O = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}.$$

Observador lineal en lazo abierto. Un observador en lazo abierto solo necesita de la entrada de control u para poder realizar la estimación de los estados faltantes. Una de las principales desventajas es que se debe conocer la condición inicial cada vez que se use dicho estimador [13]. De acuerdo a las definiciones anteriores y bajo ciertas condiciones, el algoritmo RLS puede considerarse un observador o estimador en lazo cerrado. En la siguiente sección se aborda más a detalle el desarrollo del algoritmo.

Algoritmo de mínimos cuadrados

Esta técnica se basa principalmente en obtener los parámetros que caracterizan al modelo matemático del sistema, representar su comportamiento dinámico y aproximar el modelo matemático, para obtener la estimación del comportamiento no lineal que presenta un sistema.

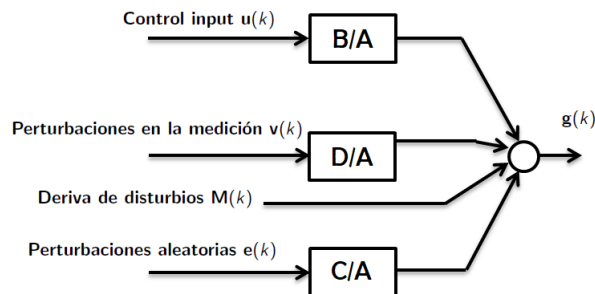


Figura 1.3: Función de transferencia discreta del sistema incluyendo todas las entradas y perturbaciones al sistema.

Considerando la función de transferencia en tiempo discreto, el sistema dinámico puede ser genéricamente representado como en la Figura (1.3), donde $k = hT_S$, siendo T_S el tiempo de muestreo (con $h = 0, 1, 2, \dots$). La secuencia de entrada está definida por

$u(k)$ y la salida por $g(k)$ sujeta a perturbaciones debidas a las señales proporcionadas por los sensores $v(k)$ y por el ruido en el proceso al momento de la puesta en marcha del cuadricóptero [12]. El modelo es escrito de la forma:

$$Ag(k) = Bu(k - 1) + Dv(k) + M(k) + Ce(k), \quad (1.7.8)$$

donde:

$$\begin{aligned} A &= 1 + a_1z^{-1} + \dots + a_{n_a}z^{-n_a} \\ B &= b_0 + b_1z^{-1} + \dots + b_{n_b}z^{-n_b} \\ D &= d_0 + d_1z^{-1} + \dots + d_{n_d}z^{-n_d} \\ M(t) &= m_0 + m_1t + \dots + m_{n_m}t^{-n_m} \\ C &= 1 + c_1z^{-1} + \dots + c_{n_c}z^{-n_c}, \end{aligned}$$

cuando los coeficientes del polinomio no son conocidos, los parámetros del polinomio son determinados por estimación [12]. Para propósitos de realizar la estimación es conveniente escribir la ecuación (1.7.8) de tal forma que enfatice los datos disponibles y los parámetros a estimar:

$$g(k) = \chi^T(k)\beta + e(k), \quad (1.7.9)$$

donde β contiene el vector de parámetros desconocidos del polinomio o vector de parámetros a estimar, y tomando como base la ecuación (1.7.8) queda definido como:

$$\beta^T = [-a_1, \dots, -a_n, b_0, \dots, b_{n_b}, m_0, \dots, m_{n_m}, c_1, \dots, c_{n_c}],$$

y χ^T es llamado el vector de regresión lineal o vector de mediciones, el cual prácticamente consiste en las variables que pueden ser medidas como por ejemplo entradas/salidas:

$$\begin{aligned} \chi^T(k) &= [y(k - 1), \dots, y(k - n_a), u(k - 1), \dots, u(k - n_b - 1), v(k), \dots, v(k - n_d), \\ &\quad 1, t, \dots, t^{n_d}, e(k - 1), e(k - 2), \dots, e(k - n_c)] \end{aligned}$$

Es evidente que este vector contiene los valores de $e(k - 1), e(k - 2), \dots, e(k - n_c)$ el cual, generalmente es desconocido ya que son valores pasados del ruido blanco $e(k)$, una perturbación no observable. Por el momento se asume que $n_c = 0$, en este

caso los coeficientes de C : c_1, c_2, c_3, \dots , son cero y de esta forma las perturbaciones desconocidas no aparecen mas en $\chi(k)$.

También se asume que, de (1.7.9) se tiene una descripción exacta del sistema, por lo tanto el mecanismo de generación del vector de parámetros se determina a partir del vector de mediciones. Para generar el vector de parámetros se supone además un modelo con la estructura correcta dada la siguiente forma:

$$g(k) = \chi^T(k)\hat{\beta} + \hat{e}(k), \quad (1.7.10)$$

donde $\hat{\beta}$, es el vector ajustable de parámetros del sistema el cual se debe hallar y $\hat{e}(k)$ corresponde al error de estimación en el tiempo $k = hT_S$. Por lo tanto el objetivo es seleccionar el vector $\hat{\beta}$, tal que se minimice el error de modelado. De la ecuación (1.7.9) y (1.7.10) se define el error de modelado, el cual corregirá los valores del vector de parámetros:

$$\hat{e}(k) = e(k) + \chi^T(k)(\beta - \hat{\beta}).$$

Se observa que $\hat{e}(k)$ depende del vector de estimación $\hat{\beta}$. Una de las técnicas principales para reducir el error de estimación es llamado recursividad de mínimos cuadrados [12].

Para que el algoritmo de minimos cuadrados pueda ser útil en el control, el sistema de estimación de parámetros debe ser iterativo, permitiendo que el modelo de estimación del sistema se actualice en cada intervalo de muestra, a medida que nuevos datos estén disponibles.

Mínimos Cuadrados Recursivos

Es útil poder visualizar el proceso de estimación en términos de la Figura 1.4, en este sistema, los nuevos datos de entrada/salida están disponibles en cada intervalo. El modelo basado en información pasada (resumido en $\hat{\beta}(k-1)$) se usa para obtener una estimación $\hat{g}(k)$ de la salida actual, esto se compara con la salida observada $g(k)$ para generar un error $\epsilon(k)$, esto a su vez genera una actualización del modelo que corrige $\hat{\beta}(k-1)$ al nuevo valor $\hat{\beta}(k)$. Esta forma recursiva de predicción-corrección permite

un ahorro significativo, en lugar de recalcularse la estimación de mínimos cuadrados en su totalidad, requiriendo el almacenamiento de todos los datos previos, es eficiente y elegante simplemente almacenar el cálculo de la estimación anterior en el momento k , indicado por $\hat{\beta}(k)$, y obtener las nuevas estimaciones $\hat{\beta}(k + 1)$ mediante un paso de actualización que implica solo la nueva observación [12].

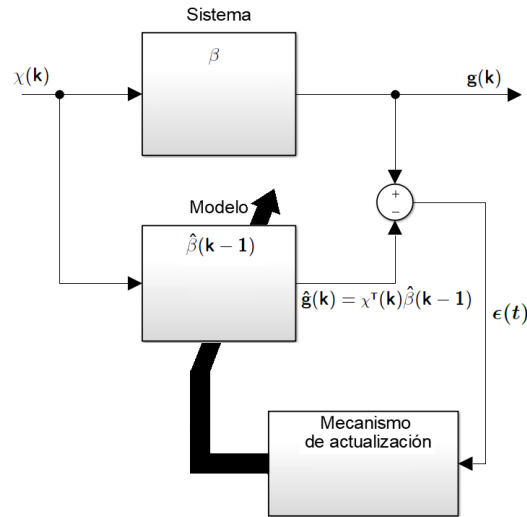


Figura 1.4: Visualización de la estimación recursiva como un proceso iterativo [12].

Usando datos para el estimador del tiempo 1 a k tenemos

$$\hat{\beta}(k) = (\chi^T(k)\chi^T)^{-1}\chi^T(k)g(k)$$

donde escribimos $\chi(k)$ como una función del tiempo para indicar que se basa en datos de los pasos anteriores hasta el tiempo k . Por lo tanto:

$$\chi(k) = \begin{bmatrix} \chi^T(1) \\ \chi^T(2) \\ \vdots \\ \chi^T(k) \end{bmatrix}, \quad \text{igualmente } \mathbf{g}(k) = \begin{bmatrix} g(1) \\ g(2) \\ \vdots \\ g(k) \end{bmatrix}$$

en el momento $k + 1$ obtenemos nuevas medidas del proceso las cuales nos permiten

formar

$$\boldsymbol{\chi}(k+1) = \begin{bmatrix} \chi^T(1) \\ \chi^T(2) \\ \vdots \\ \chi^T(k) \\ \dots \\ \chi^T(k+1) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\chi}(k) \\ \dots \\ \chi^T(k+1) \end{bmatrix} \quad (1.7.11)$$

$$\mathbf{g}(k+1) = \begin{bmatrix} g(1) \\ g(2) \\ \vdots \\ g(k) \\ \dots \\ g(k+1) \end{bmatrix} = \begin{bmatrix} \mathbf{g}(k) \\ \dots \\ g(k+1) \end{bmatrix}. \quad (1.7.12)$$

Entonces, la estimación en el paso $k+1$ esta dada por la expresión

$$\hat{\beta}(k+1) = [\boldsymbol{\chi}^T(k+1)\boldsymbol{\chi}(k+1)]^{-1} \boldsymbol{\chi}^T(k+1)\mathbf{g}(k+1)$$

ahora

$$\begin{aligned} \boldsymbol{\chi}^T(k+1)\boldsymbol{\chi}(k+1) &= [\boldsymbol{\chi}^T(k)\chi(k+1)] \begin{bmatrix} \boldsymbol{\chi}(k) \\ \dots \\ \chi^T(k+1) \end{bmatrix} \\ &= \boldsymbol{\chi}^T(k)\boldsymbol{\chi}(k) + \chi(k+1)\chi^T(k+1). \end{aligned}$$

Por lo tanto, si tenemos $\chi(k+1)$ podemos actualizar fácilmente la anterior matriz de correlaciones $\boldsymbol{\chi}^T(k)\boldsymbol{\chi}(k)$ para obtener la nueva matriz $\boldsymbol{\chi}^T(k+1)\boldsymbol{\chi}(k+1)$, sin embargo, se necesita encontrar una manera de actualizar la inversa de la matriz $\boldsymbol{\chi}^T(k)\boldsymbol{\chi}(k)$ directamente, sin requerir una inversión de matriz en cada paso de tiempo, También, se necesita actualizar el término $\boldsymbol{\chi}^T(k+1)\mathbf{g}(k+1)$ [12]. Ahora, a partir de las ecuaciones (1.7.11) y (1.7.12):

$$\boldsymbol{\chi}^T(k+1)\mathbf{g}(k+1) = [\boldsymbol{\chi}^T(k)\chi(k+1)] \begin{bmatrix} \mathbf{g}(k) \\ \dots \\ g(k+1) \end{bmatrix}$$

$$\boldsymbol{\chi}^T(k+1)\mathbf{g}(k+1) = \boldsymbol{\chi}^T(k)\mathbf{g}(k) + \chi(k+1)g(k+1).$$

Para abreviar las ecuaciones, designamos a $P(k)$ y $B(k)$ como

$$\begin{aligned} P(k) &= [\boldsymbol{\chi}^T(k)\boldsymbol{\chi}(k)]^{-1} \\ B(k) &= \boldsymbol{\chi}^T(k)\mathbf{g}(k), \end{aligned} \tag{1.7.13}$$

entonces, obtenemos

$$\begin{aligned} \hat{\beta}(k+1) &= P(k+1)B(k+1) \\ \hat{\beta}(k) &= P(k)B(k), \end{aligned}$$

igualmente

$$P^{-1}(k+1) = P^{-1}(k) + \chi(k+1)\chi^T(k+1) \tag{1.7.14}$$

y finalmente

$$B(k+1) = B(k) + \chi(k+1)g(k+1). \tag{1.7.15}$$

La ecuación (1.7.15) proporciona una actualización directa de $B(k)$ a $B(k+1)$. El paso crucial es establecer la misma actualización directa de $P(k)$ a $P(k+1)$. La forma estándar de hacerlo es aplicando el lema de inversión de matrices:

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

a (1.7.14). Asignando

$$A = P^{-1}, \quad C = 1, \quad B = \chi(k+1), \quad D = \chi^T(k+1),$$

obteniendo

$$P(k+1) = P(k) \left[I_m - \chi(k+1) (1 + \chi^T(k+1)P(k)\chi(k+1))^{-1} \chi^T(k+1)P(k) \right] \tag{1.7.16}$$

La ecuación (1.7.16) nos da los medios para actualizar $P(k)$ a $P(k+1)$ sin invertir una matriz. De hecho, la única inversión es el término escalar $[1 + \chi^T(k+1)P(k)\chi(k+1)]$. La recursión para $P(k+1)$ se puede combinar con la recursión para $B(k+1)$ (ecuación 1.7.15) de muchas maneras para dar una recursión directa para $\hat{\beta}(k+1)$ de $\hat{\beta}(k)$. La forma más común es definir la variable de error $\epsilon(k+1)$ (como se indica en la Figura 1.4) por:

$$\epsilon(k+1) = g(k+1) - \chi^T(k+1)P(k)\hat{\beta}(k) \tag{1.7.17}$$

y sustituir por $g(k+1)$ en la ecuación (1.41). Así obtenemos:

$$B(k+1) = B(k) + \mathbf{x}(k+1)\mathbf{x}^T(k+1)\hat{\beta}(k) + \mathbf{x}(k+1)\epsilon(k+1).$$

Sustituyendo para $B(k)$, $B(k+1)$ usando las ecuaciones (1.7.13) obtenemos:

$$\hat{\beta}(k+1) = \hat{\beta}(k) + P(k+1)\mathbf{x}(k+1)\epsilon(k+1). \quad (1.7.18)$$

En resumen, el algoritmo RLS completo, para actualizar a $\hat{\beta}(k)$ es como sigue [12]:
en el tiempo $k+1$

I) Usando las nuevas mediciones, formar $\mathbf{x}(k+1)$.

II) Formar $\epsilon(k+1)$ a partir de:

$$\epsilon(k+1) = g(k+1) - \mathbf{x}^T(k+1)\hat{\beta}(k)$$

III) Formar $P(k+1)$ usando

$$P(k+1) = P(k) \left[I_m - \frac{\mathbf{x}(k+1)\mathbf{x}^T(k+1)P(k)}{1 + \mathbf{x}^T(k+1)P(k)\mathbf{x}(k+1)} \right].$$

IV) Actualizar $\hat{\beta}(k)$

$$\hat{\beta}(k+1) = \hat{\beta}(k) + P(k+1)\mathbf{x}(k+1)\epsilon(k+1)$$

v) Esperar por la siguiente muestra y regresar al paso I).

1.8. Propuesta de solución

La técnica de control que se usa es un PD, sintonizado de forma heurística, para mantener el cuadricóptero en *hover*; para estimar su posición se implementó el algoritmo RLS, que permite el uso de un modelo no lineal respecto a los parámetros del sistema, para reducir el error producido por la falta de precisión del GPS y el barómetro. Finalmente, obteniendo una reducción en el error de posición. En los siguientes capítulos se muestra el desarrollo de dicha solución.

1.9. Conclusiones

La revisión de los trabajos afines al tema es de crucial importancia para comprender los alcances e innovaciones que el presente trabajo puntualiza. De igual forma, es necesaria la revisión precisa de todos los aspectos, teóricos y prácticos, que conducirán al desarrollo del trabajo y mantener claros los objetivos que se deben cumplir, es lo que se muestra en este capítulo.

Capítulo 2

Plataforma experimental

En el presente capítulo se exponen detalladamente las características y funciones de los dispositivos mecánicos y electrónicos que componen la plataforma experimental, así como la relación entre estos y su funcionamiento, la cual será facilitada por el CINVESTAV. También, se muestran las características de la plataforma utilizada para el proceso de simulación.

En éste trabajo, inicialmente se utiliza la plataforma Parrot AR.Drone 2.0 (Figura 2.1) para recolectar datos que alimentan el algoritmo RLS en simulación. De acuerdo a la pagina oficial [24] el *drone* se pilotea de forma intuitiva con un *smartphone* o una tableta, tiene una autonomía de vuelo de alrededor de 12 minutos, con un alcance aproximado de 50 metros, y además, incorpora una cámara de 720p video+foto. Con un peso de 380g con protección para interior y 420g con la protección para exterior. De forma general, a continuación se muestran las especificaciones técnicas de la plataforma:

- Grabación de vídeo HD.

Cámara HD 720p 30 FPS con objetivo gran angular diagonal de 92°, perfil de codificación básica: H264; formato fotos: JPEG; conexión: Wi-Fi.

- Asistencia electrónica.

La plataforma incorpora un procesador de 1 GHz 32 bits ARM Cortex A8 con DSP, vídeo a 800 MHz TMS320DMC64x; ejecutando un sistema operativo

Linux 2.6.32, con una memoria RAM DDR2 de 1 GB a 200 MHz. Además, tiene un puerto USB 2.0 de alta velocidad para las extensiones y una antena Wi-Fi b g n. Asimismo, incluye un giroscopio de 3 ejes, con un rango de hasta $2000^\circ/\text{segundo}$ y un acelerómetro, también de 3 ejes, con una precisión de $\pm 50mg$. Se complementa con un magnetómetro, con precisión de 6° , un sensor de presión de $\pm 10Pa$ y sensores ultrasónicos para medir la altitud.

- Motorización.

Dotado de 4 motores sin escobillas de tipo “inrunner”: 14.5 vatios y 28,500 rpm con rodamiento de esferas en miniatura, engranajes Nylatron y rodamiento de esferas autolubricadas de bronce.



Figura 2.1: Parrot AR.Drone 2.0 con protección para interior.

La plataforma Parrot se conectó mediante Wi-Fi a una interfaz desarrollada en el software LabVIEW de National Instruments, es decir, mientras la interfaz se ejecuta en una computadora, ésta se conecta a la señal Wi-Fi del *drone* Parrot, de esta forma, fue posible obtener datos de los sensores a bordo y utilizarlos para la simulación del algoritmo RLS. La interfaz que se utilizó se puede descargar fácilmente desde su pagina de internet [17] y los resultados de la simulación se muestran más adelante.

Posteriormente, para la aplicación del algoritmo en “tiempo real”, se usó como plataforma experimental un cuadricóptero (Figura 2.2), el cual consiste en un chasis

de fibra de vidrio, marca DJI, de 550 mm entre sus extremos, en donde se encuentra instalado el piloto automático de la marca 3D-Robotics llamado Pixhawk, encargado de la acción de control en el cuadricóptero. A este sistema se le pueden añadir periféricos como GPS o cualquier dispositivo con comunicación I2C, un sistema de radio comunicación RC, tubos de pitot, entre otros módulos. De forma general, la plataforma esta compuesta por:

- Estructura de fibra de vidrio en cruz.
- Sensor Compass ublox.
- Sensor GPS Ublox6.
- Autopilot pixhawk.
- Cuatro motores brushless DJI.
- Transmisor y receptor futaba 7C.
- Cuatro controladores electrónicos de velocidad (ESC por sus siglas en inglés).

Autopiloto pixhawk.

Pixhawk es un autopiloto de última generación que ofrece una gran cantidad de beneficios ya que es compatible con una gran cantidad de módulos entre los que destacan: puertos I2C, conectores para integrar modulos de GPS y magnetómetro. Además, un puerto de comunicación serial, puertos analógico-digital, etc. El autopiloto ofrece una gran cantidad de almacenamiento externo a través de la tarjeta integrada MicroSD, ideal para el almacenamiento de los datos [14].

Las características técnicas principales de este autopiloto son las siguientes:

- Microprocesador core cortex 32-bit STM32F427.
- Memoria RAM 168 MHz/256KB, Memoria Flash 2MB.
- Invensense MPU 6000 3-ejes acelerometro y giroscopio.
- Barómetro MS5611.



Figura 2.2: Plataforma experimental cuadricóptero.

- 5 puertos de comunicación serial.
- Comunicación I2C.
- Puerto analógico-digital 3.3 y 6.6 V.

Para el sistema de orientación el autopiloto cuenta con una IMU *MPU 6000 3-axis accelerometer/gyroscope*, con datos de orientación y velocidad angular, los cuales se obtienen al fusionar la información provista por los elementos inerciales como son los acelerómetros y giroscopios.

Motores y controladores de velocidad

Los motores utilizados en esta plataforma son tipo brushless, el motor 920Kv 2212 es un adecuado mecanismo de propulsión para cualquier aplicación multirrotor. Con un peso de 54 g, este motor es capaz de producir hasta medio kilogramo de empuje [15]. Compatible con baterías de 3 o 4 celdas y con hélices 1045 u 8045 respectivamente (3S PROP 1045 4S PROP 8045).

Los motores se conectan a los ESC, los cuales, tienen mayor precisión ya que cuentan con un oscilador de cristal (por lo tanto, la temperatura no afectará el ciclo de trabajo del PWM). Tienen una velocidad de actualización alta sin almacenamiento en búfer de la señal de entrada, lo que resulta en una tasa de respuesta de más de 490Hz, que proporciona una respuesta más rápida del motor y una operación más silenciosa también [15]. El detalle de conexión y configuración se puede encontrar en la pagina web de Ardupilot [26].

Sistema de posicionamiento global (GPS)

El GPS utilizado en la plataforma experimental es de la marca uBlox, y una de las principales ventajas es que integra un “compass” digital tipo HMC5883L. Este GPS es recomendado por la marca 3DRobotics para vuelos en exteriores: la frecuencia de muestreo es de 5 Hz y maneja una precisión de $\pm 2.8m$ [16].

El piloto automático controla todos los periféricos y en base a la lectura de estos, es posible que controle cualquier robot móvil. La plataforma utilizada en el presente trabajo, como ya se había mencionado, lleva a bordo un autopiloto Pixhawk, al cual se adapta un distribuidor de voltaje, donde se conectan los ESC que controlan los motores. También, se conecta al autopiloto, un botón de arme/desarme y una bocina para señales audibles, que en conjunto, sirven para aumentar la seguridad al momento de poner en marcha el cuadricóptero. De igual forma, se conecta un modulo GPS, así como un volmetro-ampermetro que sirve para monitorizar el estado de la batería LiPo de 3 celdas de 4 amperes que alimenta el sistema. En lo que respecta al radio control, es necesario conectar un decodificador por modulación de posición de pulso (PPM por sus siglas en inglés) al receptor del radio control para la comunicación con el autopiloto. En la siguiente figura se aprecia la conexión de todos los dispositivos.

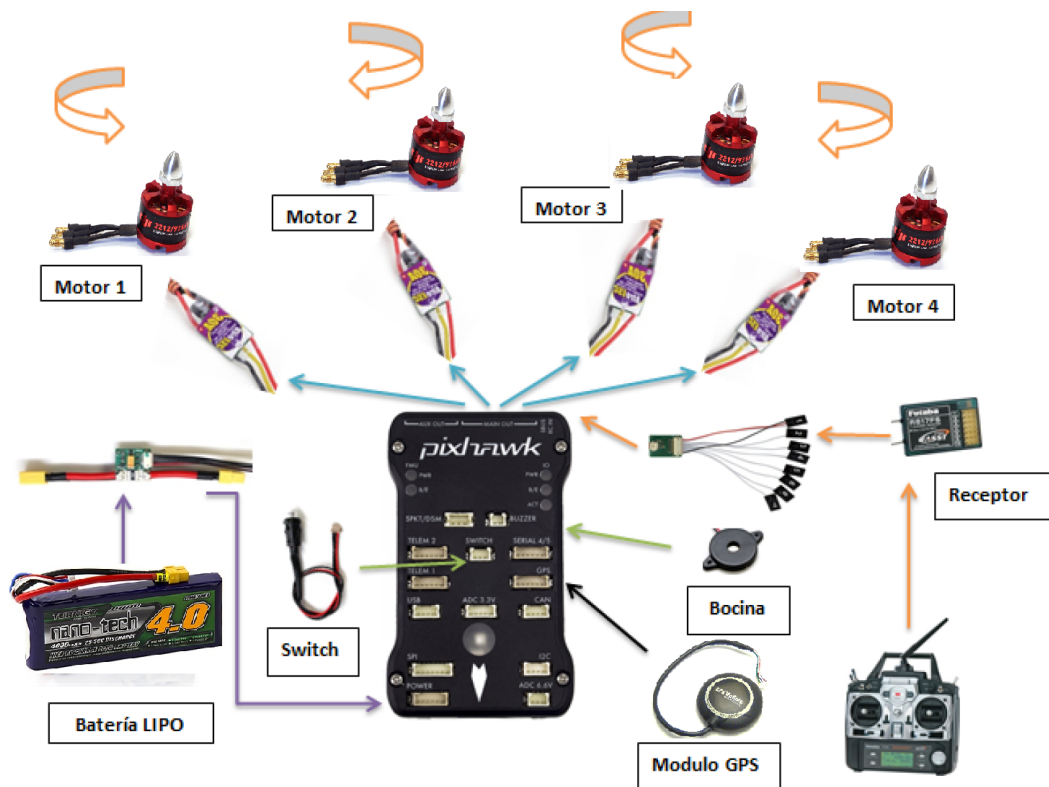


Figura 2.3: Diagrama de conexión [25].

Capítulo 3

Resultados experimentales

A partir del proceso de simulación, es posible llegar a obtener información del comportamiento de un sistema, si está bien modelado. Además, es un paso importante para llevar a cabo una implementación física, con resultados experimentales.

3.1. Simulación

La definición más formal de simulación, puede enunciarse como sigue: es el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el comportamiento del sistema o evaluar nuevas estrategias, dentro de los límites impuestos por un cierto criterio o un conjunto de ellos, para el funcionamiento del sistema (R. E. Shannon).

3.1.1. Simulación del controlador PD

De acuerdo al modelo dinámico del cuadricóptero, se realizó una simulación con la herramienta simulink del *software* MATLAB, para lo cual, la expresión 1.7.7, se dividió en tres subsistemas, $\psi - z$, $\theta - x$ y $\phi - y$ a los cuales se les aplicó un control PD a cada uno, como se muestra en la Figura 3.1. Para llevar a cabo la simulación, se realizaron las configuraciones básicas de la herramienta *simulink* con el método numérico, ode45 y paso variable.

Para controlar la altura z se le aplica una linealización exacta, con el objetivo de cancelar las no linealidades. Considerando la dinámica del modelo respecto a la altura, definido como sigue:

$$m\ddot{z} = u \cos \theta \cos \phi - mg. \quad (3.1.1)$$

Es claro que el subsistema 3.1.1 puede linealizarse de forma exacta, aplicando la siguiente ley de control,

$$u(t) = m(u_1(t) + g)(\cos(\theta) \cos(\phi))^{-1} \quad (3.1.2)$$

donde $\cos(\theta)\cos(\phi) \neq 0$, $\phi \in (-\frac{\pi}{2}, \frac{\pi}{2})$ las cuales son condiciones deseables de operación en un cuadricóptero [3].

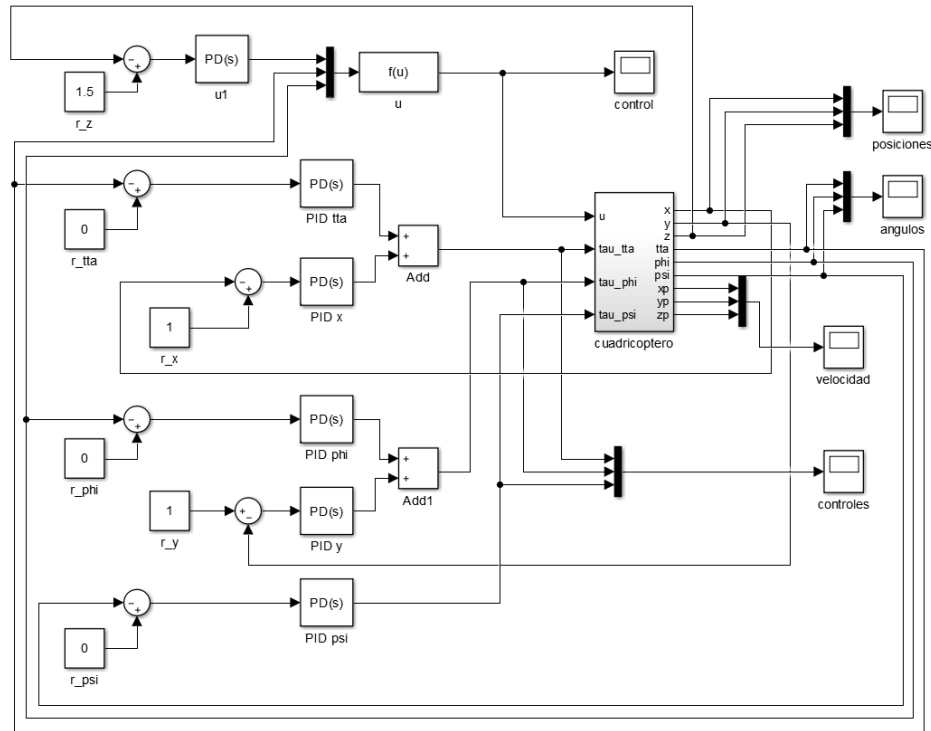


Figura 3.1: Diagrama a bloques del control del cuadricóptero.

Para sintonizar los controles PD se pueden utilizar diversas técnicas, como Ziegler-Nichols o el método de D-particiones, sin embargo, en este trabajo se utilizó una

forma heurística, de una manera similar como en [10], tenemos la sintonización de un PD. Retomando el modelo reducido del cuadricóptero 1.7.7, mostrado nuevamente a continuación,

$$\begin{aligned} m\ddot{x} &= -u \sin \theta \\ m\ddot{y} &= u \cos \theta \sin \phi \\ m\ddot{z} &= u \cos \theta \cos \phi - mg \\ \ddot{\phi} &= \tau_\phi \\ \ddot{\theta} &= \tau_\theta \\ \ddot{\psi} &= \tau_\psi \end{aligned}$$

donde se considera una linealización en z (para ángulos pequeños), primero, si se asume a $m = 1kg$ se obtiene un modelo lineal para el subsistema

$$\ddot{z} = u \cos \theta \cos \phi - g$$

después, sustituyendo a u por 3.1.2, obtenemos

$$\ddot{z} = \frac{u_1 + g}{\cos(\theta) \cos(\phi)} \cos(\theta) \cos(\phi) - g,$$

cancelando términos, la expresión se reduce a

$$\ddot{z} = u_1,$$

donde se propone a u_1 como

$$u_1 = -k_p z - k_d \dot{z}.$$

En la expresión anterior se considera que el punto de equilibrio deseado es cero, por lo tanto se obtiene: $\ddot{z} + k_d \dot{z} + k_p z = 0$, claramente cuando $k_d, k_p > 0$ el sistema es estable. Sin embargo, el objetivo es obtener un controlador con una sobreelongación (M_p) y tiempo de establecimiento (t_s) específicos. Si se considera $M_p = 0.1\%$ y $t_s = 5s$, de acuerdo con el resultado clásico obtenido en [11],

$$M_p = e^{-(\zeta \sqrt{1-\zeta^2})\pi}$$

y

$$t_s = \frac{4}{\zeta \omega_n}$$

(para $\pm 2\%$ criterio) donde ζ y ω_n son el coeficiente de amortiguamiento y la frecuencia natural no amortiguada respectivamente. Dado M_p y t_s , se deduce que

$$\zeta = \frac{|\ln M_p|}{\sqrt{\pi^2 + (\ln M_p)^2}} = 0.82609$$

y

$$\omega_n = \frac{4}{\zeta t_s} = 0.96841,$$

entonces $k_p = \sqrt{\omega_n} = 0.23825$, y $k_d = 2\zeta\omega_n = 1.599$. Si los valores obtenidos no son factibles en la práctica, se deben proponer otros valores para M_p y t_s . De esta manera se sintoniza el controlador en [10].

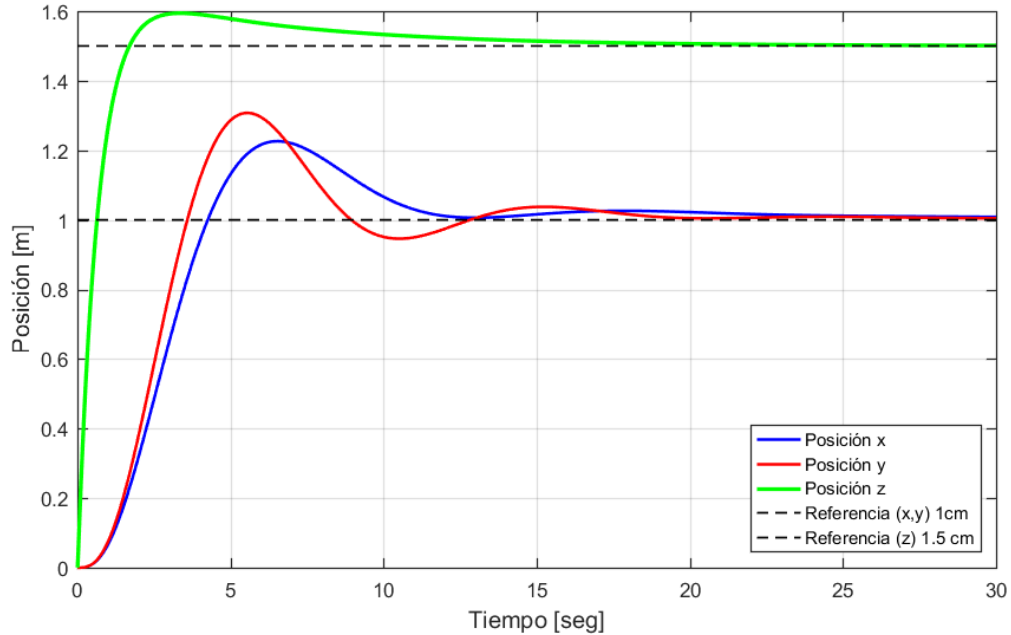


Figura 3.2: Posiciones x, y, z del cuadricóptero.

Para el caso del cuadricóptero, en la Figura 3.2, se puede observar el resultado de la sintonización en (x, y, z) , para la referencia (1.0cm, 1.0cm, 1.5cm) respectivamente; considerando que es una simulación es posible asignar valores arbitrariamente. En la gráfica, se puede observar que la referencia se alcanza máximo a los 20 segundos aproximadamente, por lo que la señal de control es elevada y la energía que demanda es excesiva (Figura 3.3). Si se mejora la sintonización (Figura 3.5), considerando la

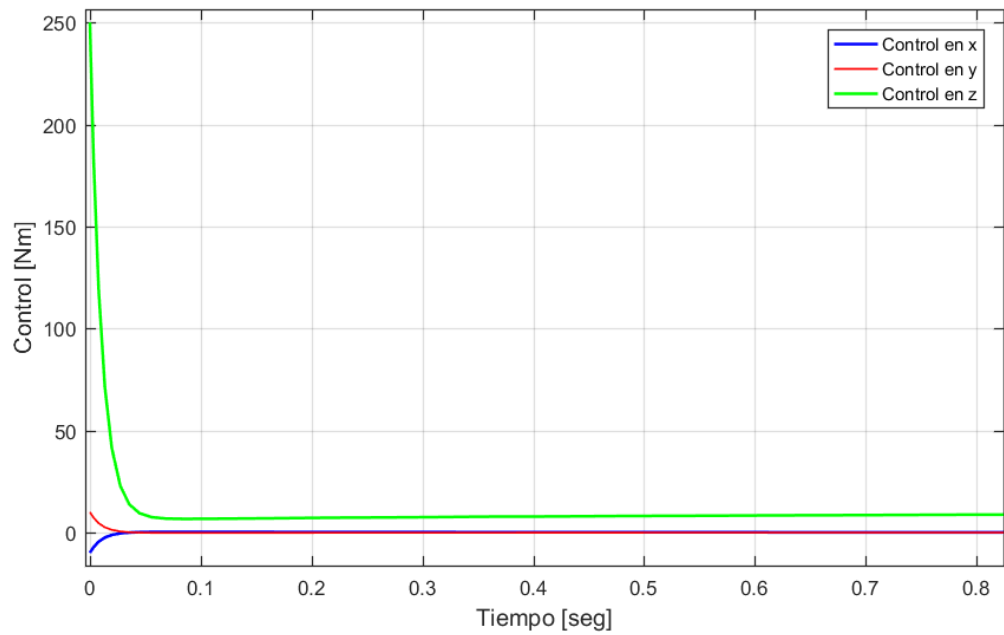


Figura 3.3: Gráfica del control para x, y, z del cuadricóptero.

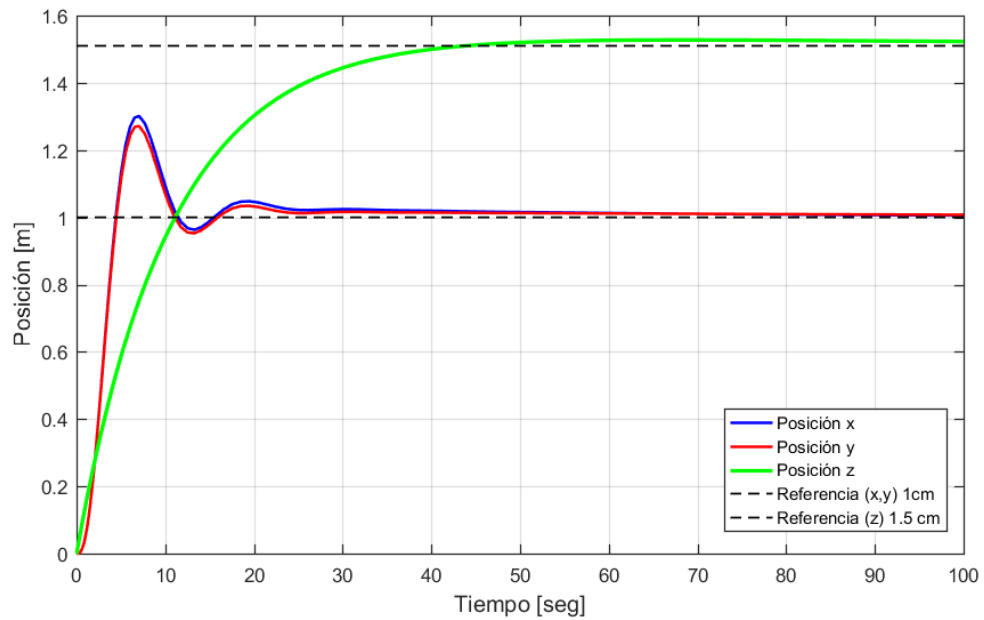


Figura 3.4: Posiciones x, y, z del cuadricóptero con diferentes ganancias.

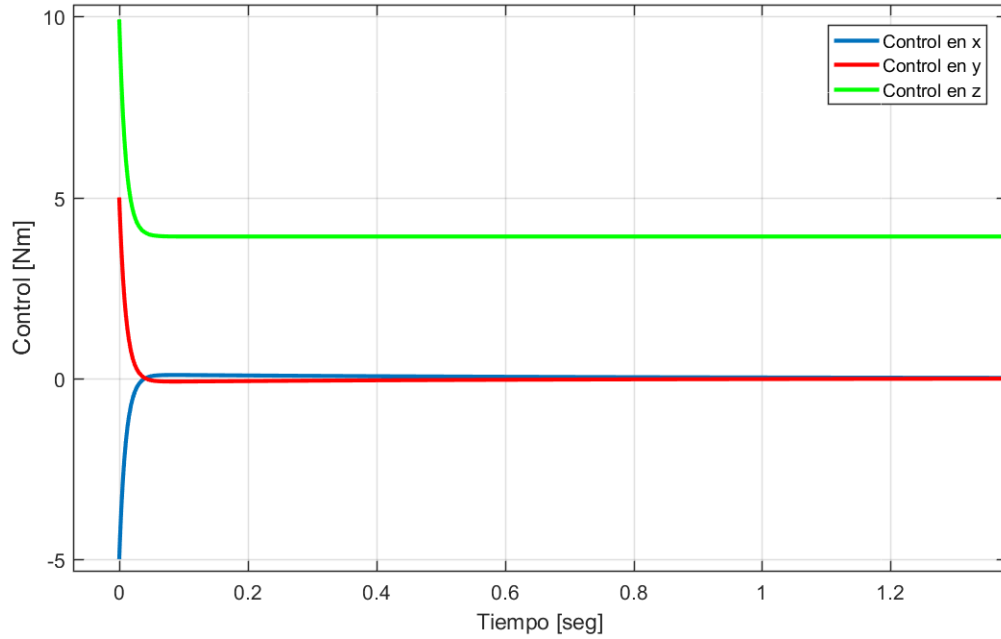


Figura 3.5: Gráfica del control para x, y, z del cuadricóptero con diferentes ganancias.

magnitud del esfuerzo de control, el cuadricóptero tarda más tiempo en llegar a la referencia deseada (Figura 3.4), comparado con la sintonización anterior, es decir, en la primera gráfica, el control que exige es de hasta 180Nm y en la segunda, donde las ganancias del control se modificaron, el esfuerzo de control llega sólo hasta 10Nm. Es importante considerar el esfuerzo de control, ya que, en el sistema real, es posible dañar los actuadores si la señal de control es muy grande.

3.1.2. Simulación del algoritmo RLS

Para realizar la estimación de estado del cuadricóptero es posible utilizar modelos ARX y para determinar los parámetros del modelo se puede usar el algoritmo de RLS. Como un primer paso para la propuesta de un modelo ARX se discretizan los subsistemas del modelo (1.7.7), considerando a $k = 1, 2, \dots, n$ como tiempo discreto, primero para y , sabemos

$$\dot{y} \approx \frac{y(k) - y(k-1)}{T_s},$$

por lo tanto

$$\ddot{y} \approx \frac{\left[\frac{y(k)-y(k-1)}{T_s} - \frac{y(k-1)-y(k-2)}{T_s} \right]}{T_s}$$

$$\ddot{y} \approx \left[\frac{y(k) - 2y(k-1) + y(k-2)}{T_s^2} \right]$$

entonces, si sustituimos \ddot{y} en el subsistema y de 1.7.7, tenemos

$$m \left[\frac{y(k) - 2y(k-1) + y(k-2)}{T_s^2} \right] = u(k) \cos \theta \sin \phi$$

despejando a $y(k)$,

$$y(k) = 2y(k-1) - y(k-2) + T_s^2 \bar{u}_1(k) \quad (3.1.3)$$

donde

$$\begin{aligned} \bar{u}_1(k) &= u(k)v_1(k) \quad y \\ v_1(k) &= \frac{\cos \theta \sin \phi}{m}. \end{aligned}$$

Para x , z y ϕ se realiza el mismo procedimiento y obtenemos

$$x(k) = 2x(k-1) - x(k-2) + T_s^2 \bar{u}_2(k) \quad (3.1.4)$$

$$\begin{aligned} \bar{u}_2(k) &= u(k)v_2(k) \quad y \\ v_2(k) &= \frac{-\sin \theta}{m}, \end{aligned}$$

y finalmente para z

$$z(k) = 2z(k-1) - z(k-2) + \left(\frac{\cos \theta \cos \phi - g}{m} \right) (T_s^2). \quad (3.1.5)$$

En simulación se consideró utilizar a ϕ como salida, así que la ecuación resultante es,

$$\phi(k) = 2\phi(k-1) - \phi(k-2) + u, \quad u = \tau_\phi \quad (3.1.6)$$

y al igual que 3.1.5 se utilizó para la aproximación mediante RLS.

Podemos utilizar las ecuaciones (3.1.3) y (3.1.4) para formar el modelo ARX de y logrando así una primera propuesta,

$$y(k) = a_0 y(k-1) - a_1 y(k-2) + a_2 T_s^2 \bar{u}_1(k)$$

o bien,

$$y(k) = \begin{bmatrix} y(k-1) & y(k-2) & T_s^2 \bar{u}_1(k) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

Donde el primer vector representa el vector de regresión $\boldsymbol{\chi}^T$ y el segundo $\boldsymbol{\beta}$ el vector de parámetros desconocidos.

A partir de este modelo se puede aplicar el algoritmo RLS que se puede resumir como al final de la sección 1.7.4, sin embargo, no podemos conocer un valor futuro, es decir, no podemos conocer a $\boldsymbol{\chi}(k+1)$ ya que k es la muestra actual, entonces reescribimos los pasos de la siguiente forma:

en el tiempo k

- I) Usando las nuevas mediciones de θ y ϕ formar $\boldsymbol{\chi}(k)$.
- II) Usando $\epsilon(k) = y(k) - \boldsymbol{\chi}^T(k)\hat{\boldsymbol{\beta}}(k-1)$ para formar el error $\epsilon(k)$, donde $\hat{\boldsymbol{\beta}}(k-1)$ es el vector de parámetros estimados, sin embargo, debido a que es recursivo, el vector se vuelve escalar.
- III) Formar $P(k)$ usando

$$P(k) = P(k-1) \left[I_m - \frac{\boldsymbol{\chi}(k)\boldsymbol{\chi}^T(k)P(k-1)}{1 + \boldsymbol{\chi}^T(k)P(k-1)\boldsymbol{\chi}(k)} \right].$$

- IV) Actualizar $\hat{\boldsymbol{\beta}}(k-1)$

$$\hat{\boldsymbol{\beta}}(k) = \hat{\boldsymbol{\beta}}(k-1) + P(k)\boldsymbol{\chi}(k)\epsilon(k)$$

- V) Esperar por la siguiente muestra y regresar al paso I).

Para esta parte de la simulación, se utilizó la plataforma Parrot, mediante la interfaz en LabVIEW se obtuvieron datos de las posiciones y velocidades angulares (pitch(θ), roll(ϕ), yaw(ψ)), así como datos de la altura. Las posiciones traslacionales no pueden ser medidas con esta plataforma, ya que no cuenta con GPS. El programa en MATLAB se diseñó para cargar los datos de los ángulos de Euler (θ, ϕ, ψ) obtenidos

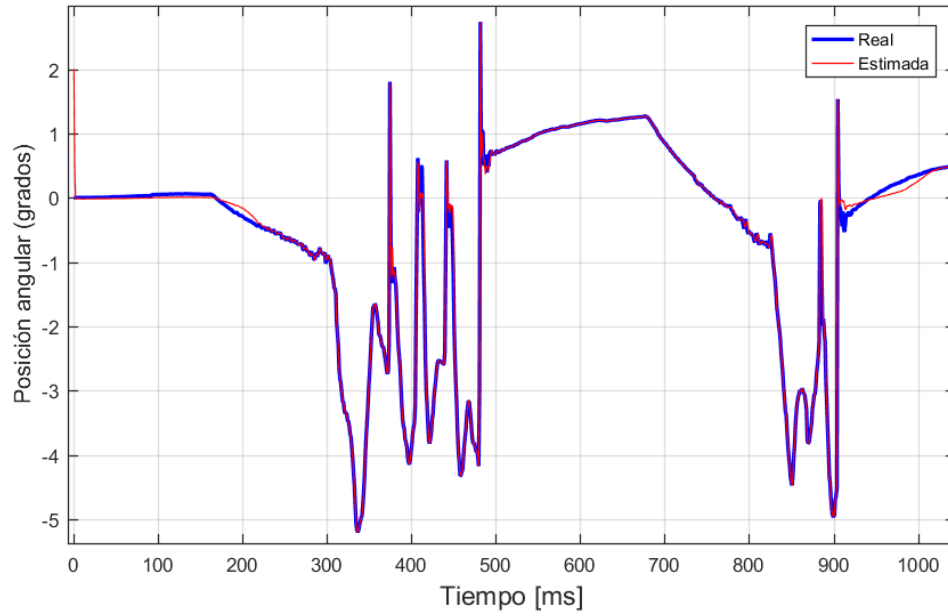


Figura 3.6: Comparación entre las lecturas reales de ϕ y las estimadas, obtenidos a partir de RLS, con condiciones iniciales diferentes de cero.

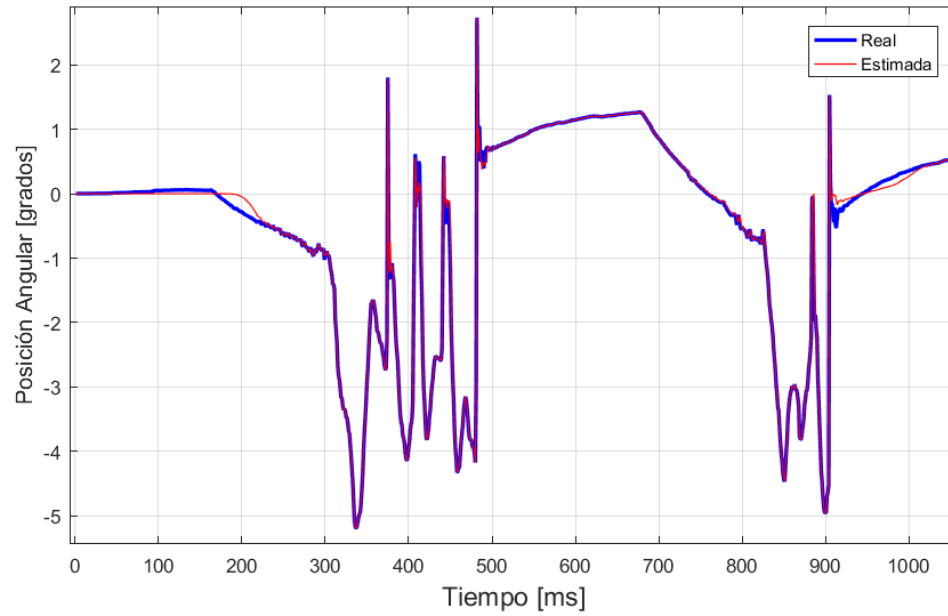


Figura 3.7: Comparación entre las lecturas reales de ϕ y las estimadas, obtenidos a partir de RLS, con condiciones iniciales cero.

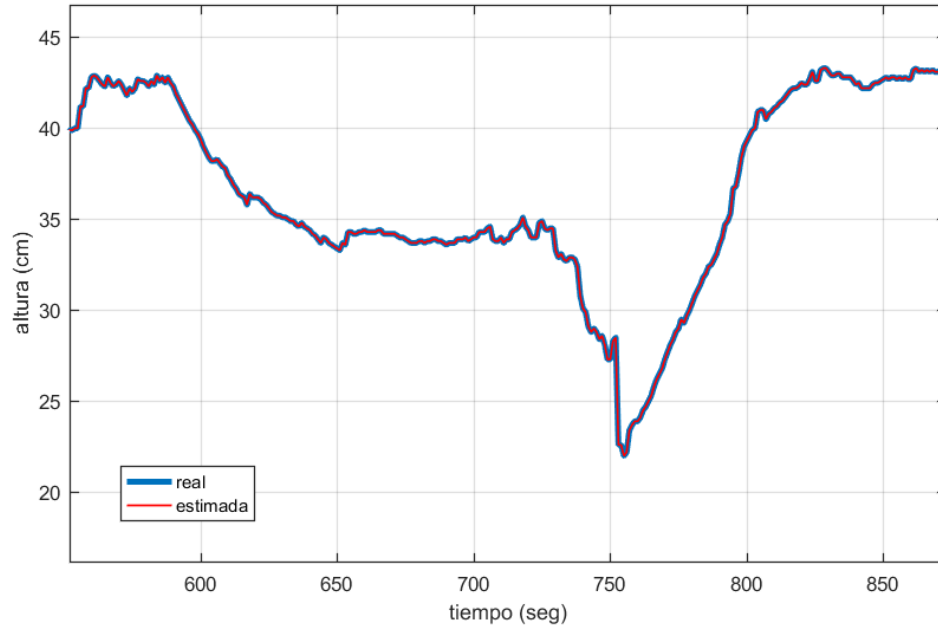


Figura 3.8: Comparación entre las lecturas reales de z y las estimadas, obtenidos a partir de RLS.

anteriormente, y tratarlos con el algoritmo RLS, para verificar la estimación de los parámetros del modelo ARX antes propuesto (ecuaciones 3.1.5 y 3.1.6), y observar el comportamiento de los datos reales respecto a los estimados y de esta forma comprobar el correcto funcionamiento y programación del algoritmo. En la Figura 3.6, observamos la aproximación a ϕ cuando las condiciones iniciales del algoritmo son diferentes de cero, es decir, asignamos $\beta^T = [5, 5, 5]$ para esta gráfica, asimismo en la Figura 3.7, las condiciones iniciales son igual a cero y finalmente en la Figura 3.8 la aproximación a la altura z . En esta gráfica se hizo un acercamiento para poder apreciar las gráficas de los datos recolectados y la estimada. En las tres imágenes es posible observar que el algoritmo proporciona una buena estimación.

3.2. Programación

Para desarrollar el *firmware* utilizado por el autopiloto PixHawk el programa Eclipse (Figura 3.10) fue fundamental, el cual, es una plataforma de *software* compuesto por un conjunto de herramientas de programación de código abierto multi-plataforma, para desarrollar lo que el proyecto llama “Aplicaciones de Cliente Enriquecido”, opuesto a las aplicaciones “Cliente-liviano” basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador eclipse para java (ECJ por sus siglas en inglés) que se entrega como parte del *software* Eclipse (y que son usados también para desarrollar el mismo entorno)[22]. Eclipse fue desarrollado originalmente por IBM como el sucesor de su fa-



Figura 3.9: *Software* Eclipse.

milia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios [22].

Así como Eclipse se utiliza para desarrollar el *firmware* del autopiloto, el *software* Mission Planner (MP) se utiliza para cargar el *firmware* a la plataforma, además de

otras funcionalidades. MP es una estación de control terrestre para aeroplanos, helicópteros, (multirrotores) y vehículos terrestres. Es compatible con Windows solamente. MP puede utilizarse como una utilidad de configuración o como un suplemento de control dinámico para su vehículo autónomo [23]. Entre sus funciones tenemos:

- Cargar el firmware (el *software*) en el piloto automático (APM, PX4 ...) que controla su vehículo.
- Configurar y ajustar su vehículo para un rendimiento óptimo.
- Planificar, guardar y cargar misiones autónomas en el autopiloto.
- Descargar y analizar los registros de misión creados por su autopiloto.
- Con el hardware de telemetría adecuado puede:
 - Controlar el estado de su vehículo mientras está en funcionamiento.
 - Registrar los datos de telemetría que contienen mucha más información que los registros del autopiloto a bordo.
 - Ver y analizar los registros de telemetría [23].



Figura 3.10: *Software* Mission Planner.

Mediante estas dos importantes herramientas se desarrollo el *firmware* y se cargo al autopiloto, ambos son *software* libres y el autopiloto es de arquitectura abierta,

con los que es posible mejorar o desarrollar algoritmos de control propios, para implementarlos físicamente.

En lo que respecta a la creación del firmware para el autopiloto pixhawk primero se estableció la comunicación entre los periféricos conectados al autopiloto, por ejemplo, en primera instancia el barómetro con el cual se comenzó a trabajar para obtener datos de la altura proporcionada por este sensor, a partir de estas lecturas se comenzó a programar el algoritmo RLS de la misma forma que se programó en MATLAB, con algunas ligeras diferencias debido a la naturaleza de los dos diferentes *software*, uno de programación-simulación y el otro de programación para la creación del firmware para el Pixhawk y también considerar que la aplicación es en “tiempo real”.

El cuerpo del programa se divide principalmente en:

- La declaración de librerías.
- Declaración de variables.
- Después encontramos el siguiente código, el cual se encarga de asignar el tiempo de procesos para cada subrutina especificada de acuerdo a la tabla especificada en los comentarios del código:

```
/*
 scheduler table for fast CPUs - all regular tasks apart from the fast_loop(
 should be listed here, along with how often they should be called
 (in 2.5ms units) and the maximum time they are expected to take (in
 microseconds)
1    = 400hz
2    = 200hz
4    = 100hz
8    = 50hz
20   = 20hz
40   = 10hz
133  = 3hz
400  = 1hz
4000 = 0.1hz

*/
```

```
static const AP_Scheduler::Task scheduler_tasks[] PROGMEM = {
    {Rol_telemetria, 8,    400},
    {Leer_radio,    4,    10},
    {alturarol,    40,    100},
    {posicion,     4,    10},
    {read_battery, 40,    20},
    { MemorySD,    20,    100},
    { minimos,     20,    20}
};
```

- Finalmente el cuerpo principal del programa.

```
void setup(){...}
void loop(){...}
static void fast_loop(void){...}
static void Rol_telemetria(){...}
...
static void minimos(){...}
AP_HAL_MAIN();
```

El programa no se ejecuta de forma secuencial en el sentido estricto, ya que, en la “tabla” del programa donde están las subrutinas, se le asigna un tiempo de procesamiento, a cada una, de tal forma que la secuencia, en un pequeño lapso, puede variar. Sin embargo, si se considera un lapso de tiempo suficientemente grande, sería posible encontrar una secuencia.

En lo que respecta a la subrutina denominada “minimos” es el código correspondiente al algoritmo RLS y se puede observar a continuación:

```
static void minimos(void)
{
    ts=.05;
    g=9.81;

    zg=(ts*ts)*c_alt*(((cos(pitch)*cos(roll))/m)-g);
    z2=z1; //z(k-1)
    z1=zi; //z(k-2)
    zi=zg;
    ttg=[ttg1_11;ttg1_12;ttg1_13]
```

```
P1=z1*z1+z2*z2+zg*zg;

P2=P1*(1-(P1*P1)/(1+P1*P1));
ttg2_11=ttg1_11+P2*z1*er;
ttg2_12=ttg1_12+P2*z2*er;
ttg2_13=ttg1_13+P2*zg*er;

ttg1_11=ttg2_11;
ttg1_12=ttg2_12;
ttg1_13=ttg2_13;

ze=z1*ttg1_11+z2*ttg1_12+zg*ttg1_13; //z estimada
}
```

El código es una subrutina del Pixhawk que realiza la estimación de la altura a partir de los ángulos de euler.

El control de altura fue programado a partir del estudio previo, que se realizó tanto en la revisión bibliográfica como en la parte de simulación, teniendo como fin el desarrollo del control de altura con un PD y considerando la linealización exacta expresada en 3.1.2. La programación del control de altura se realizó a partir de las consideraciones correspondientes a la dinámica propia del cuadricoptero, es decir el control de los ángulos de euler y para este primer caso el control de altura obteniendo a m_1 , m_2 , m_3 y m_4 como los torques de salida de cada uno de los motores, c_alt el control de altura, c_gas y las variables con el prefijo radio, corresponden a las señales enviadas por el radio control, asimismo las variables con el prefijo c indican el esfuerzo de control para los ángulos de euler. El código para el control de altura es el siguiente:

```
...c_gas=radio_throttle + c_alt;
m1=c_gas-radio_roll+radio_pitch+radio_yaw+c_roll-c_pitch-c_yaw;
m2=c_gas+radio_roll-radio_pitch+radio_yaw-c_roll+c_pitch-c_yaw;
m3=c_gas+radio_roll+radio_pitch-radio_yaw-c_roll-c_pitch+c_yaw;
m4=c_gas-radio_roll-radio_pitch-radio_yaw+c_roll+c_pitch+c_yaw;
...
```

En los fragmentos anteriores del código podemos observar la programación del control de altura y el algoritmo RLS. En el apéndice A se muestra el código desarrollado para el autopiloto.

3.3. Implementación

Los resultados obtenidos a partir del algoritmo programado, se muestran en la Figura 3.11, donde, la línea azul representa la altura obtenida de las mediciones del barómetro; la línea roja la estimación obtenida a partir del algoritmo RLS y finalmente la línea punteada representa el promedio ponderado, sin embargo, en este caso en particular, el promedio tiene la misma ponderación, es decir, la estimación y las lecturas del barómetro tienen el mismo peso, 50 % cada una. Los datos utilizados para esta gráfica se obtuvieron a partir del almacenamiento de datos del autopiloto PixHawk, sin embargo, en este punto aún no se han realizado pruebas que verifiquen el buen funcionamiento del control, en la gráfica 3.11 solo se muestra el funcionamiento del algoritmo RLS.

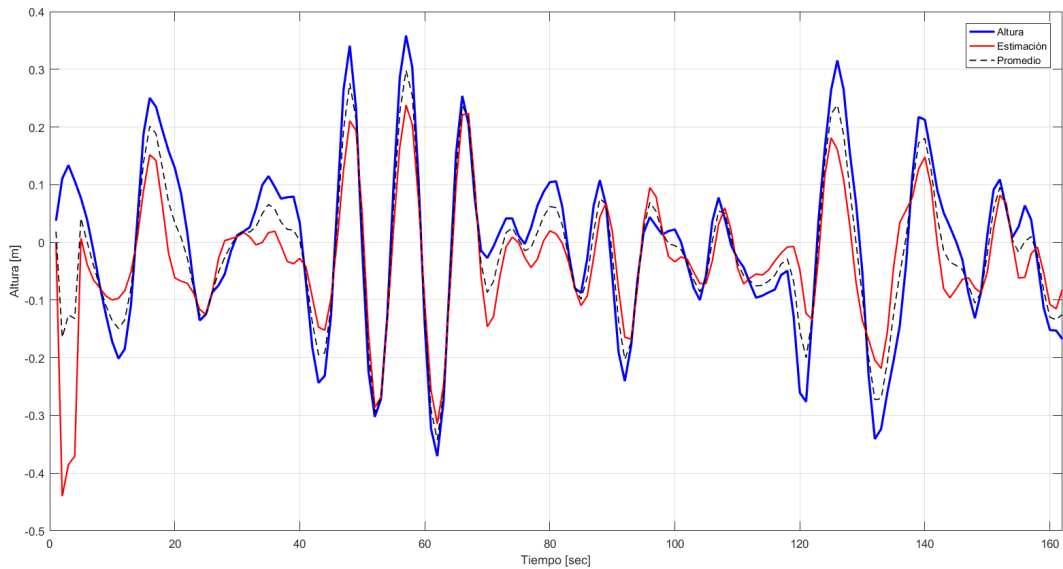


Figura 3.11: Comparación entre RLS, lecturas del barómetro y el promedio ponderado.

Es importante mencionar que en el gráfico, las alturas negativas que se aprecian, se deben al principio de funcionamiento del barómetro. La medida de presión entra en un modelo de atmósfera (la Atmósfera Estándar Internacional) que relaciona la presión con la altitud y de ahí se extrae la altitud de vuelo. Este sistema tiene cierta imprecisión porque la atmósfera nunca se comporta como el modelo, pero es suficientemente bueno y robusto. También, como ya se había mencionado, los MEMS tienen

una deriva, que es más evidente al estar estático por mucho tiempo, como se aprecia en la Figura 3.12.

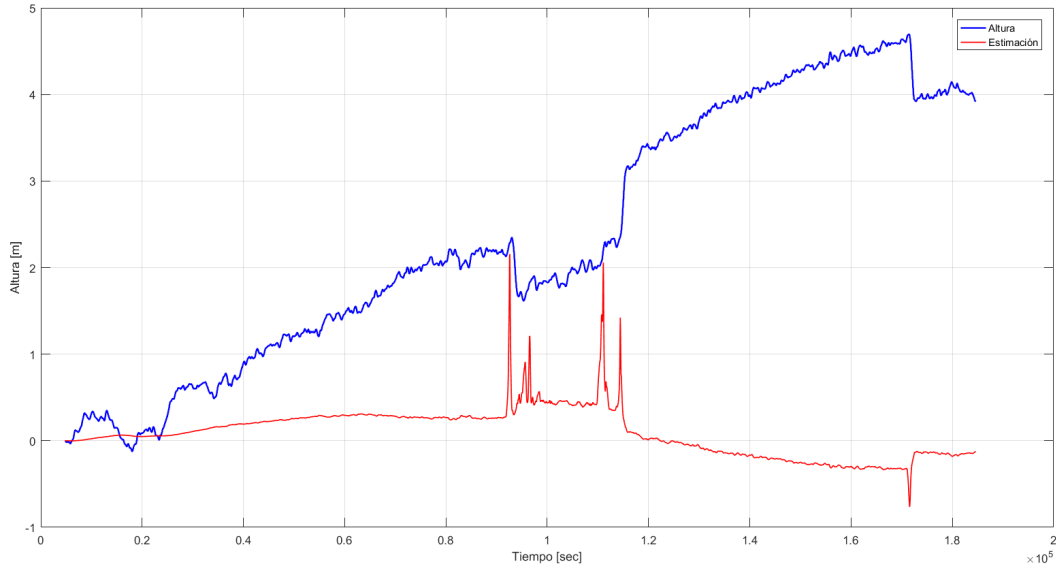


Figura 3.12: Comparación entre la estimación y la deriva del barómetro.

La aplicación física de nuestros algoritmos siempre lleva un proceso de instrumentación y estudio de los componentes de la planta, además de realizar un acoplamiento de señales, por lo que, implica retos importantes en la aplicación de la teoría desarrollada. También en la Figura 3.13 podemos observar un gráfico que muestra la altura del cuadricoptero, la línea roja representa la altura directa del barómetro, la azul es el resultado de la estimación de estado y la línea verde representa la referencia del control. En la Figura 3.14 podemos observar un acercamiento del gráfico de la Figura 3.13 donde se aprecia que los límites máximos y mínimos de la estimación no superan a las lecturas de altitud proporcionadas por el barómetro. Esta conclusión es más evidente en la Tabla 3.1 donde se compara la media del error, obtenida a partir de la estimación, las lecturas del barómetro y un filtro, esto, en función a la referencia establecida. El filtro utilizado para la comparación es una utilidad del autopiloto Pixhawk.

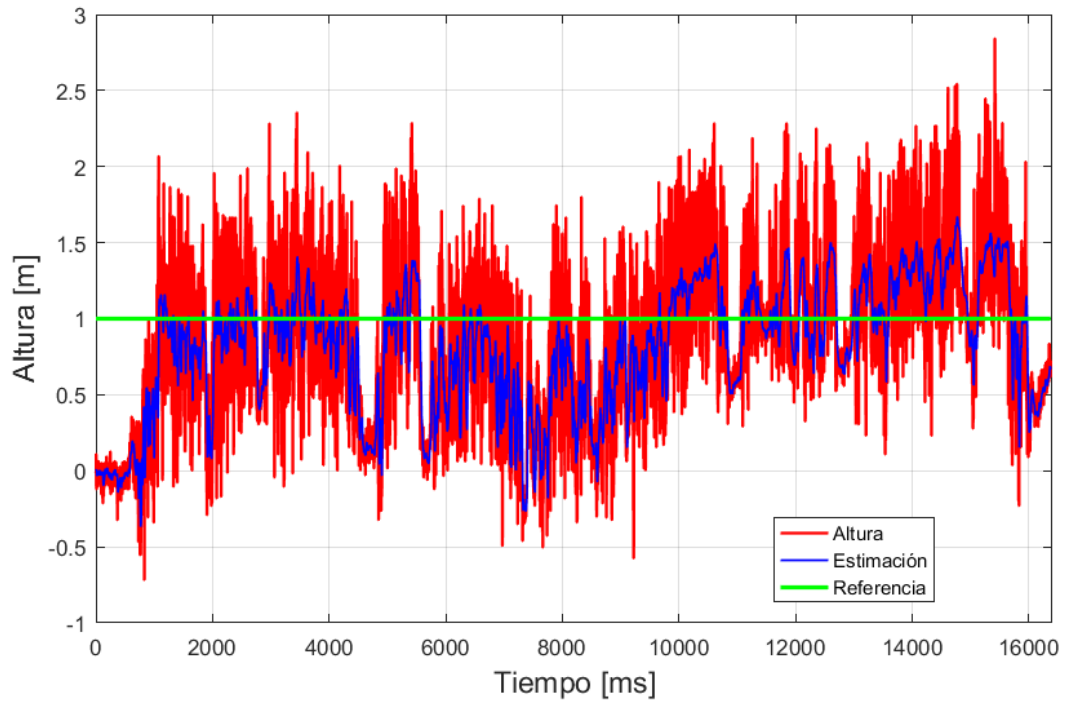


Figura 3.13: Señal del barómetro comparada con la estimación y la referencia de control.

Media del error	Estimación	Barómetro	Filtro
Altura	0.356 m	0.455 m	0.408 m

Tabla 3.1: Comparativa entre las diferentes medias del error.

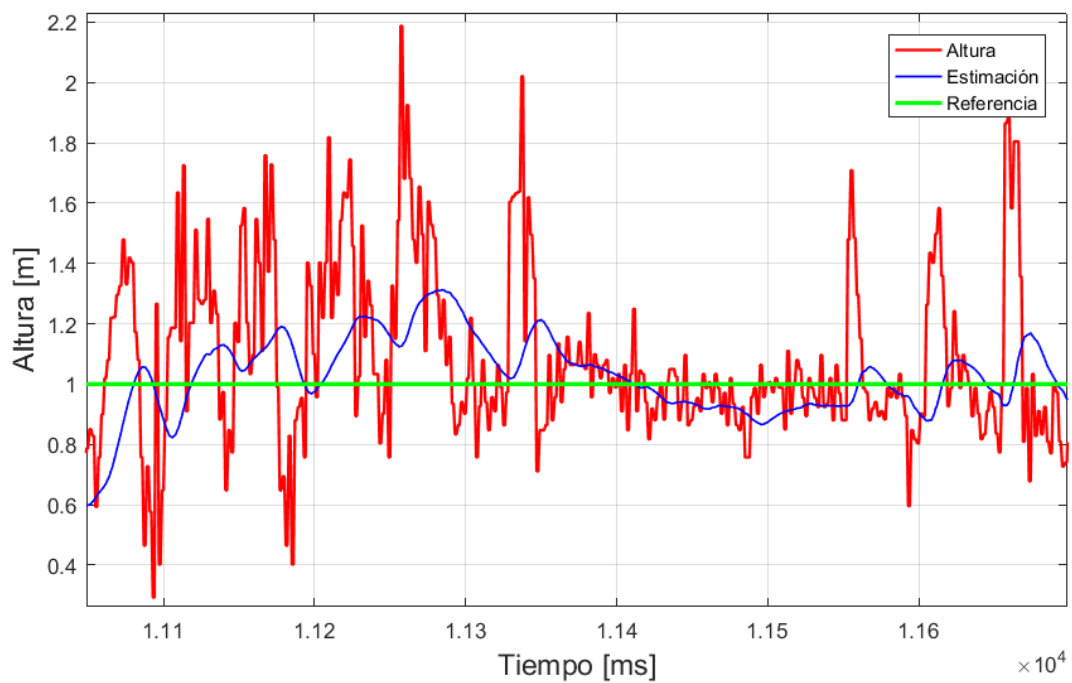


Figura 3.14: Acercamiento de la comparación entre la estimación y la señal del barómetro.

Capítulo 4

Conclusiones y trabajos futuros

Con los resultados obtenidos se demuestra que mediante el algoritmo RLS programado en el autopiloto Pixhawk se logró reducir el error de posicionamiento de altura del cuadricóptero. Sin embargo como trabajo futuro es posible implementar una técnica control diferente, además, considerar dinámicas no modeladas en el polinomio del modelo ARX, para aumentar la eficiencia del algoritmo en general.

Asimismo implementar el algoritmo para la reducción del error de posicionamiento traslacional del cuadricóptero y su control.

Apéndice A

El siguiente código se desarrollo para controlar la altura del cuadricoptero en base a la estimación de estado realizada con el algoritmo RLS.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>

// Common dependencies
#include <AP_Common.h>
#include <AP_Progmem.h>
#include <AP_Menu.h>
#include <AP_Param.h>
#include <StorageManager.h>
// AP_HAL
#include <AP_HAL.h>
#include <AP_HAL_AVR.h>
#include <AP_HAL_SITL.h>
#include <AP_HAL_PX4.h>
#include <AP_HAL_VRBRAIN.h>
#include <AP_HAL_FLYMAPLE.h>
#include <AP_HAL_Linux.h>
#include <AP_HAL_Empty.h>
#include <AP_Math.h>
// Application dependencies
#include <GCS.h>
#include <GCS_MAVLink.h> // MAVLink GCS definitions
#include <AP_SerialManager.h> // Serial manager library
#include <AP_GPS.h> // ArduPilot GPS library
```

Apéndice A:

```
#include <DataFlash.h>           // ArduPilot Mega Flash Memory Library
#include <AP_ADC.h>              // ArduPilot Mega Analog to Digital Converter
                                Library
#include <AP_ADC_AnalogSource.h>
#include <AP_Baro.h>
#include <AP_Compass.h>         // ArduPilot Mega Magnetometer Library
#include <AP_Math.h>            // ArduPilot Mega Vector/Matrix math Library
#include <AP_Curve.h>           // Curve used to linearlise throttle pwm to
                                thrust
#include <AP_InertialSensor.h>  // ArduPilot Mega Inertial Sensor (accel & gyro)
                                Library
#include <AP_AHRS.h>
#include <AP_NavEKF.h>
#include <AP_Mission.h>        // Mission command library
#include <AP_Rally.h>          // Rally point library
#include <AC_PID.h>             // PID library
#include <AC_PI_2D.h>          // PID library (2-axis)
#include <AC_HELI_PID.h>       // Heli specific Rate PID library
#include <AC_P.h>               // P library
#include <AC_AttitudeControl.h> // Attitude control library
#include <AC_AttitudeControl_Heli.h> // Attitude control library for
                                traditional helicopter
#include <AC_PosControl.h>      // Position control library
#include <RC_Channel.h>         // RC Channel Library
#include <AP_Motors.h>          // AP Motors library
#include <AP_RangeFinder.h>     // Range finder library
#include <AP_OpticalFlow.h>     // Optical Flow library
#include <Filter.h>             // Filter library
#include <AP_Buffer.h>          // APM FIFO Buffer
#include <AP_Relay.h>           // APM relay
#include <AP_ServoRelayEvents.h>
#include <AP_Camera.h>          // Photo or video camera
#include <AP_Mount.h>           // Camera/Antenna mount
#include <AP_Airspeed.h>        // needed for AHRS build
#include <AP_Vehicle.h>         // needed for AHRS build
#include <AP_InertialNav.h>     // ArduPilot Mega inertial navigation library
#include <AC_WPNav.h>           // ArduCopter waypoint navigation library
#include <AC_Circle.h>          // circle navigation library
#include <AP_Declination.h>     // ArduPilot Mega Declination Helper Library
#include <AC_Fence.h>           // Arducopter Fence library
```

Apéndice A:

```
#include <SITL.h> // software in the loop support
#include <AP_Scheduler.h> // main loop scheduler
#include <AP_RCMapper.h> // RC input mapping library
#include <AP_Notify.h> // Notify library
#include <AP_BattMonitor.h> // Battery monitor library
#include <AP_BoardConfig.h> // board configuration library
#include <AP_Frsky_Telem.h>
#if SPRAYER == ENABLED
#include <AC_Sprayer.h> // crop sprayer library
#endif
#if EPM_ENABLED == ENABLED
#include <AP_EPM.h> // EPM cargo gripper stuff
#endif
#if PARACHUTE == ENABLED
#include <AP_Parachute.h> // Parachute release library
#endif
#include <AP_LandingGear.h> // Landing Gear library
#include <AP_Terrain.h>
#include <LowPassFilter2p.h>
// AP_HAL to Arduino compatibility layer
#include "compat.h"
// Configuration
#include "defines.h"
#include "config.h"
#include "config_channels.h"

# define MAIN_LOOP_RATE 400
# define MAIN_LOOP_SECONDS 0.0025f
# define MAIN_LOOP_MICROS 2500

const AP_HAL::HAL& hal = AP_HAL_BOARD_DRIVER;
static AP_Scheduler scheduler;
AP_BattMonitor battery;
static AP_SerialManager serial_manager;
static OpticalFlow optflow;
static AP_OpticalFlow_PX4 Optflow(optflow);
static AP_Baro barometer;
static Compass compass;
static AP_GPS gps;
```

Apéndice A:

```
static AP_InertialSensor ins;
static AP_Notify notify; // Notificacion
static float baro_climbrate; // barometer climbrate in cm/s

// SONAR
#if CONFIG_SONAR == ENABLED
static RangeFinder sonar;
static bool sonar_enabled = true; // enable user switch for sonar
#endif
// Inertial Navigation EKF
#if AP_AHRS_NAVEKF_AVAILABLE
AP_AHRS_NavEKF ahrs(ins, barometer, gps, sonar);
#else
AP_AHRS_DCM ahrs(ins, barometer, gps);
#endif

static AP_InertialNav_NavEKF inav(ahrs);

float baro_alt=0, alt_fil,alt_antb=0,vel_b=0, ofil=0;
int radio_roll, radio_pitch, radio_yaw, aux_1, aux_2, aux_3;
uint16_t radio[7];
float roll, pitch, yaw,radio_throttle;
float c_roll, c_pitch, c_yaw, c_alt, c_x=0, c_y=0;
float gyro_x, gyro_y, gyro_z;
static Vector3f pos;
static Vector3f vels;
static Vector3f gyro;
float c_gas=0,kp_roll, kd_roll, kp_pitch, kd_pitch,
  kp_yaw, kd_yaw, kp_z=20, kd_z=-70, ki_z;
int kp_x=0, kd_x=0, kp_y=0, kd_y=0, aux_r=0,latt,lngg;
static AP_BoardLED board_led;

float m1,m2,m3,m4,ze,t1,ts,volt,sate;
float tsa,zer,ze1,zei,kpz,kdz,g=9.81,m=1.1; /////ganacias altura
float z2,z1,zi,zg,P1,er,ttg1_11,ttg1_12,ttg1_13,
ttg2_11,ttg2_12,ttg2_13,P2,promp;
float vel_z,alt_z,posz,posx,posy,velx,vely,velz,pos_z,xer,yer,
fix,pz,csz,csx,csy;
```

Apéndice A:

```
// Integration time (in seconds) for the gyros (DCM algorithm)
// Updated with the fast loop
static float G_Dt = 0.02;
//Filtro barometro
static LowPassFilter2pfloat fil_posz(10,0.8);//1.1, 0.5 excelente, 30 .29
static LowPassFilter2pfloat fil_velz(10,0.3);//0.4 excelente, 30 .08
static LowPassFilter2pfloat fil_he(10,0.97);//30 .
//static LowPassFilter2pfloat fil_velz (30,0.29);//30 .29
//static LowPassFilter2pfloat low_pass_filter(800, 30);

//-----
// Time in microseconds of main control loop
static uint32_t fast_loopTimer;

// Dataflash storage in SD
#if CONFIG_HAL_BOARD == HAL_BOARD_PX4
static DataFlash_File DataFlash("/fs/microsd/APM/LOGS");
#endif
static uint16_t log_num;
/*
Format characters in the format string for binary log messages
  b  : int8_t
  B  : uint8_t
  h  : int16_t
  H  : uint16_t
  i  : int32_t
  I  : uint32_t
  f  : float
  n  : char[4]
  N  : char[16]
  Z  : char[64]
  c  : int16_t * 100
  C  : uint16_t * 100
  e  : int32_t * 100
  E  : uint32_t * 100
  L  : int32_t latitude/longitude
  M  : uint8_t flight mode
*/
```


Apéndice A:

```
#define LOG_TEST_MSG 1

//Paquete a enviar dependiente del tipo de variables de la tabla de abajo
struct PACKED log_Test {
    LOG_PACKET_HEADER;
    float f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15; // ff

};

static const struct LogStructure log_structure[] PROGMEM = {
    { LOG_TEST_MSG, sizeof(log_Test),
      "1", "ffffffffffffffff",
        "f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15" }
};

//////////////////////////////////// NOTIFY //////////////////////////////////////
#if CONFIG_HAL_BOARD == HAL_BOARD_PX4
static ToshibaLED_PX4 toshiba_led;
#else
static ToshibaLED_I2C toshiba_led;
#endif
#define LED_DIM 0x11

///Tareas////////////////////////////////////

/*
scheduler table for fast CPUs - all regular tasks apart from the fast_loop()
should be listed here, along with how often they should be called
(in 2.5ms units) and the maximum time they are expected to take (in
microseconds)
1    = 400hz
2    = 200hz
4    = 100hz
8    = 50hz
20   = 20hz
40   = 10hz
133  = 3hz
400  = 1hz
4000 = 0.1hz
```

Apéndice A:

```
*/

static const AP_Scheduler::Task scheduler_tasks[] PROGMEM = {
// #if OPTFLOW == ENABLED
//   { update_optical_flow, 8, 40 }, // 20
// #endif
  // { Rol_telemetria, 8, 400 }, // 400 // 8, 100 Bien
  { Leer_radio, 4, 10 },
  { alturarol, 40, 100 },
  { posicion, 4, 10 },
  // { trayectoria, 400, 20 },
  { read_battery, 40, 20 },
  // { optimal_gains, 40, 20 }
  { MemorySD, 20, 100 },
  { minimos, 20, 20 } // tenia 20
};

void setup()
{
  barometer.init();
  barometer.calibrate();
  ins.init(AP_InertialSensor::COLD_START, AP_InertialSensor::RATE_400HZ);
  ahrs.init();
  serial_manager.init_console();
  serial_manager.init();
  compass.init();
  compass.read();

  gps.init(NULL, serial_manager);
  ahrs.set_compass(&compass);
  hal._uartC->begin(57600);
  // hal._uartD->begin(9600); // Aded rol090316
  battery.set_monitoring(0, AP_BattMonitor::BattMonitor_TYPE_ANALOG
    _VOLTAGE_AND_CURRENT);
  battery.init();
  // initialise the main loop scheduler
  scheduler.init(&scheduler_tasks[0],
    sizeof(scheduler_tasks)/sizeof(scheduler_tasks[0]));
}
```

Apéndice A:

```
hal.rcout->set_freq(15,490); // Inicializacion de los motores
    a frecuencia de 490
hal.rcout->enable_ch(0); // como son cuatro motores
son 15 bits de q son 1111=15
hal.rcout->enable_ch(1); // Habilitamos los motores del 0 al 3
hal.rcout->enable_ch(2);
hal.rcout->enable_ch(3);

// hal.scheduler->delay(2000); //Rol modified 4 dic 2015
fast_loopTimer = hal.scheduler->micros();//Rol modified 4 dic 2015
//para guardar en sd
DataFlash.Init(log_structure,
    sizeof(log_structure)/sizeof(log_structure[0]));
if (DataFlash.NeedErase()) {
    DataFlash.EraseAll();
}
// We start to write some info (sequentially) starting from page 1
// This is similar to what we will do...
log_num = DataFlash.StartNewLog();
}

void loop()
{
    ins.wait_for_sample();//Rol
    uint32_t timer = micros(); //Rol
    // used by PI Loops
    G_Dt = (float)(timer - fast_loopTimer) / 1000000.0f;
    fast_loopTimer = timer;
    fast_loop(); //Llamado al loop rápido
    scheduler.tick(); //Rol
    uint32_t time_available = (timer + MAIN_LOOP_MICROS) - micros();//Rol
    scheduler.run(time_available);//Rol // verificar ejemplo
}

static void fast_loop(void)
{
    //gps.update();
    ahrs.update();
    compass.read();
}
```

Apéndice A:

```
t1=hal.scheduler->millis();

kd_x = 10; //38;//kd; //18
kd_y = 10; //38;//kd;
kp_x = 5; //29;//kp;
kp_y = 5; //29;//kp;
kp_yaw =110; //115;//115//90;//80
kd_yaw =80; //80;//80//100;
kp_roll =30; //36 bien//21,38
kd_roll =91; ///84 bien//91
kp_pitch =30; //bien//21,42
kd_pitch =91; ///85 //91
kpz=5;//18;
    kdz=10;//43;

roll = ahrs.roll;
pitch = ahrs.pitch;
yaw = ahrs.yaw;

gyro = ins.get_gyro();
pos = inav.get_position();
vels = inav.get_velocity();

gyrox = gyro.x; //velocidades en los angulos ahrs
gyroy = gyro.y;
gyroz = gyro.z;

posx = pos.x/1000; //posiciones x,y,z ahrs
posy = pos.y/1000;
posz = pos.z/1000;

velx = vels.x; //velocidades ahrs
vely = vels.y;
velz = vels.z;

vel_z = inav.get_velocity_z();
pos_z = inav.get_altitude();

promp=(alt_fil*.3)+ze*.7;
tsa=0.0025;
```

Apéndice A:

```
//ERROR DE ALTURA
zer=1-alt_fil;
ze1=zei;
zei=zer;

//control_altura
c_alt=((kpz*zer+kdz*((zer-ze1)/tsa))+g)*m/(cos(pitch)*cos(roll));

// C_orientacion angulos
c_roll = kp_roll * roll + kd_roll * gyro_x;
c_pitch = kp_pitch * pitch + kd_pitch * gyro_y;
c_yaw = kp_yaw * yaw + kd_yaw * gyro_z;

//Control orientacion x y
//c_x = kp_x * posX + kd_x * vel_x;
//c_y = kp_y * posY + kd_y * vel_y;

if(aux_2 >1200){
    c_gas=radio_throttle + c_alt;

m1=c_gas-radio_roll+radio_pitch+radio_yaw+c_roll-c_pitch-c_yaw;
m2=c_gas+radio_roll-radio_pitch+radio_yaw-c_roll+c_pitch-c_yaw;
m3=c_gas+radio_roll+radio_pitch-radio_yaw-c_roll-c_pitch+c_yaw;
m4=c_gas-radio_roll-radio_pitch-radio_yaw+c_roll+c_pitch+c_yaw;
}else{
    c_gas=radio_throttle;

m1=c_gas-radio_roll+radio_pitch+radio_yaw+c_roll-c_pitch-c_yaw;
m2=c_gas+radio_roll-radio_pitch+radio_yaw-c_roll+c_pitch-c_yaw;
m3=c_gas+radio_roll+radio_pitch-radio_yaw-c_roll-c_pitch+c_yaw;
m4=c_gas-radio_roll-radio_pitch-radio_yaw+c_roll+c_pitch+c_yaw;
}

if(radio_throttle>1150){
```

Apéndice A:

```
        hal.rcout->write(0,m1);
        hal.rcout->write(1,m2);
        hal.rcout->write(2,m3);
        hal.rcout->write(3,m4);

    }else{
        hal.rcout->write(0,1100);
        hal.rcout->write(1,1100);
        hal.rcout->write(2,1100);
        hal.rcout->write(3,1100);
    }
}

static void alturarol()
{
    barometer.update();
    baro_alt = barometer.get_altitude(); //Return meters
    baro_climbrate = barometer.get_climb_rate() * 100.0f;
    ////////////////////////////////////// Filtro
    alt_fil=fil_posz.apply(baro_alt);
}

static uint32_t micros()
{
    return hal.scheduler->micros();
}

static void Leer_radio(){

    for (uint8_t i = 0; i <= 7; i++)
    {radio[i] = hal.rcin->read(i);}

    radio_roll = (radio[0]-1500)/3;//(radio[0]-1500)/3;
    radio_pitch = (radio[1]-1510)/3;//(radio[1]-1492)/3;
    if (radio[2]>1470){
        radio_throttle = 1470+(radio[2]-1470)*0.32;
    }else{
        radio_throttle = (radio[2]);
    }
}
```

Apéndice A:

```
    radio_yaw = (radio[3]-1510)/2;
    aux_1 = radio[4];
    aux_2 = radio[5];
    aux_3 =(radio[6]-1099)*0.15; //Select gains
}

static void MemorySD(void)
{ //Paramteros a enviar lista

    struct log_Test pkt = {
        LOG_PACKET_HEADER_INIT(LOG_TEST_MSG),
        /*1*/  f1:posx,
        /*2*/  f2:posy,
        /*3*/  f3:posz,
        /*4*/  f4:c_x,
        /*5*/  f5:c_y,
        /*6*/  f6:radio_throttle,
        /*7*/  f7:zer,
        /*8*/  f8:baro_alt,
        /*9*/  f9:alt_fil,
        /*10*/ f10:c_alt,
        /*11*/ f11:ofil,
        /*12*/ f12:sate,
        /*13*/ f13:volt,
        /*14*/ f14:ze,
        /*15*/ f15:t1,
    };

    DataFlash.WriteBlock(&pkt, sizeof(pkt));
}

static void minimos(void)
{
    ts=.05;
    //m=.4;
    g=9.81;

    zg=(ts*ts)*c_alt*(((cos(pitch)*cos(roll))/m)-g);
    z2=z1;
}
```

Apéndice A:

```
    z1=zi;
    zi=zg;
        /////          chit=[z1,z2,zg];
        /////          chi=[z1;z2;zg];
        /////          ttg=[ttg1_11;ttg1_12;ttg1_13]

    P1=z1*z1+z2*z2+zg*zg;

    er=alt_fil-(z1*ttg1_11+z2*ttg1_12+zg*ttg1_13);    //RLS

    P2=P1*(1-(P1*P1)/(1+P1*P1));
    ttg2_11=ttg1_11+P2*z1*er;
    ttg2_12=ttg1_12+P2*z2*er;
    ttg2_13=ttg1_13+P2*zg*er;

    ttg1_11=ttg2_11;
    ttg1_12=ttg2_12;
    ttg1_13=ttg2_13;

    ze=z1*ttg1_11+z2*ttg1_12+zg*ttg1_13;
}

static void read_battery(){
    battery.read();
    volt=battery.voltage();
    if (volt <= 11 && volt > 5){
        toshiba_led.init();
        toshiba_led.update();
        toshiba_led.set_rgb(0,LED_DIM,0);
    }
}

static void posicion(){
    static uint32_t last_msg_ms;
    static uint32_t last_update;
    float dt;

    gps.update();
    if (last_msg_ms != gps.last_message_time_ms())
    {
```


Apéndice A:

```
    last_msg_ms = gps.last_message_time_ms();
    const Location &loc =gps.location();
    sate = gps.status();

    latt=loc.lat;
    lngg=loc.lng;
    //toshiba_led.update();
    //toshiba_led.set_rgb(0,0,LED_DIM);
}
uint32_t currrtime = hal.scheduler->millis();
dt = (float)(currrtime - last_update) / 1000.0f;
last_update = currrtime;
inav.update(dt);
if( sate >=3 && posx != 0){
    const Location &loc = gps.location();
    ahrs.set_home(loc);
    compass.set_initial_location(loc.lat, loc.lng);
    toshiba_led.init();
    toshiba_led.update();
    toshiba_led.set_rgb(0,0,LED_DIM);
}
inav.get_longitude());
}

AP_HAL_MAIN();
```

Bibliografía

- [1] P. Castillo, P. García, R. Lozano, P. Albertos, “Modelado y estabilización de un helicóptero con cuatro rotores,” *Revista Iberoamericana de Automatica e Informatica Industrial*, vol. 4, pp. 41-57, 2007.
- [2] R. Lozano, *Unmanned aerial vehicles: Embedded control*, 2013, John Wiley & Sons.
- [3] H. Romero, S. Salazar, R. Lozano, “Real Time stabilization of an Eight-Rotor UAV Using Optical Flow”. *IEEE Transactions on Robotics*, vol. 25, no. 2009.
- [4] H. Romero, S. Salazar, R. Lozano, “Visual servoing applied to real-time stabilization of a multi-rotor UAV” , *Cambridge University Press*, vol. 30, pp. 1203-1212. 2012.
- [5] S. Grzonka, G. Grisetti, and W Burgard, “A Fully Autonomous Indoor Quadrotor”, *IEEE Transactions on Robotics*, vol. 28, NO. 1, FEBRUARY 2012.
- [6] P. Castillo, A. Dzul, R. Lozano. (julio 2004). Real-Time Stabilization and Tracking of a Four-Rotor Mini Rotorcraft. *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*, 12, 510-516.
- [7] M. Rafik, and L. Vincenzo, “Image momentos-based velocity estimation of UAVs in GPS denied environments”, *Safety, Security, and Rescue Robotics (SSRR), 2014 IEEE International Symposium*, 2014.
- [8] X.Liu, T. Huang, X. Tang, and H. Xin, “Design of self-adaptive PID controller based on least square method.” *IEEE Computer Society*, Oct. 2009, pp. 527-529.

- [9] A. Montiel, O. Santos, S. Salazar, H. Romero, I. González and R. Lozano, "State Estimation for a Quadrotor Aircraft in GPS Denied Environment", Accepted in *International Conference on Unmanned Aircraft Systems (ICUAS)*
- [10] O. Santos, H. Romero, S. Salazar, O. García, R. Lozano, "Optimized Discrete Control Law for Quadrotor Stabilization: Experimental Results", *Springer Science+Business Media Dordrecht*, 2016
- [11] Katsuhiko Ogata. (2010). *Ingeniería de control moderna*. Madrid: Pearson.
- [12] Wellstead P, Zarrop M. *Self-tuning Systems Control and Signal Processing*. Hoboken, New Jersey: John Wiley & Sons Ltd.; 1991.
- [13] Chen, C *Linear System Theory and Design*, 2013, Oxford University
- [14] <https://store.3drobotics.com/products/3dr-pixhawk>, Octubre de 2016
- [15] <http://www.hobbyking.com/hobbyking/store/38455/TurnigyMultistar/4220>, Octubre de 2016
- [16] <https://store.3drobotics.com/products/3dr-gps-ublox-with-compass>, Octubre de 2016
- [17] <http://sine.ni.com/nips/cds/view/p/lang/es/nid/211837>, octubre 2016
- [18] B. Kada, K. Munawar, M.S. Shaikh, M.A. Hussaini, U.M. Al-Saggaf. (2016). "UAV attitude estimation using nonlinear filtering and low-cost MEMS sensors". *IFAC-PapersOnLine* , 49-21.
- [19] P. Martin, E. Salaün . (2008). "An Invariant Observer for Earth-Velocity-Aided Attitude Heading Reference Systems". *IFAC-The International Federation of Automatic Control* , 9857-9864.
- [20] W. Chen, X. Li, X. Song, B. Li, X. Song and Q. Xu. (2015). "A novel fusion methodology to bridge GPS outages for land vehicle positioning". *Meas. Sci. Technol.* 26, 1-12.

BIBLIOGRAFÍA

- [21] The UAV Guide Wiki. (2014). Multicopter. junio 2017, Sitio web: <http://wiki.theuavguide.com/wiki/Multicopter>
- [22] ECLIPSE. Tools and IDEs. Junio 2017, de ECLIPSE Sitio web: <http://www.eclipse.org/>
- [23] ArduPilot Dev Team. "Mission Planner". Junio 2017, Sitio web: <http://ardupilot.org/planner/docs/mission-planner-overview.html>
- [24] Parrot AR.Drone 2.0 Elite Edition Share my bird's eye view in HD. Diciembre 2017, de Parrot Sitio web: <https://www.parrot.com/us/drones/parrot-ardrone-20-elite-edition#parrot-ardrone-20-elite-edition>
- [25] A. Montiel. (Enero 2016). "Navegación de un cuadricóptero en exteriores sin asistencia de GPS". CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL, 65.
- [26] Ardupilot. (Octubre, 2010). Connect ESCs and Motors. Diciembre 2017, de Ardupilot Sitio web: <http://ardupilot.org/copter/docs/connect-escs-and-motors.html>