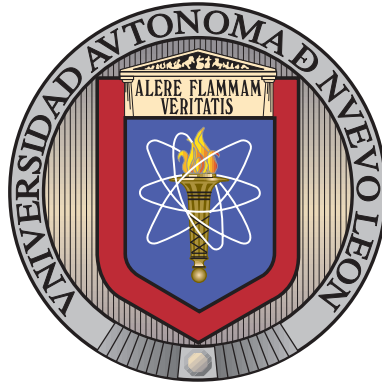


UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

SUBDIRECCIÓN DE ESTUDIOS DE POSGRADO



ESTABILIZACIÓN DE VUELO EN UAVS CON
SINTONIZACIÓN INTELIGENTE DE
CONTROLADORES CONVENCIONALES

POR

VÍCTOR DE JESÚS MEDRANO ZARAZÚA

COMO REQUISITO PARA OBTENER EL GRADO DE

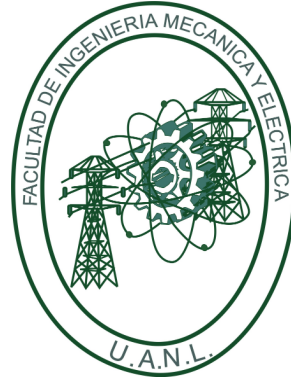
MAESTRO EN CIENCIAS DE LA INGENIERÍA ELÉCTRICA

ABRIL 2017

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

SUBDIRECCIÓN DE ESTUDIOS DE POSGRADO



ESTABILIZACIÓN DE VUELO EN UAVS CON
SINTONIZACIÓN INTELIGENTE DE
CONTROLADORES CONVENCIONALES

POR

VÍCTOR DE JESÚS MEDRANO ZARAZÚA

COMO REQUISITO PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA INGENIERÍA ELÉCTRICA

ABRIL 2017

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica
Subdirección de Estudios de Posgrado

Los miembros del Comité de Tesis recomendamos que la tesis «Estabilización de vuelo en UAVs con sintonización inteligente de controladores convencionales», realizada por el alumno Víctor de Jesús Medrano Zarazúa, con número de matrícula 1442635, sea aceptada para su defensa como opción al grado de Maestro en Ciencias de la Ingeniería Eléctrica.

El Comité de Tesis



Dr. Luis Torres-Treviño

Asesor



Dr. Juan Angel Rodríguez Liñán

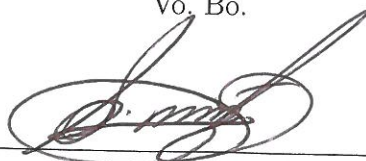
Revisor



Dr. Oscar Salvador Salas Peña

Revisor

Vo. Bo.



Dr. Simón Martínez Martínez

Subdirector de Estudios de Posgrado

San Nicolás de los Garza, Nuevo León, abril 2017

A mi familia, amigos y aquellas personas que se esfuerzan por transformar y moldear el mundo para bien de la sociedad. Especialmente a mi madre, que ha estado ahí en todo momento.

ÍNDICE GENERAL

Agradecimientos	XII
Nomenclaturas	XIV
Resumen	XVI
1. Introducción	1
1.1. Motivación	2
1.2. Antecedentes	4
1.3. Justificación	7
1.4. Hipótesis	9
1.5. Objetivos	9
1.6. Metodología	9
1.7. Contribuciones	10
1.8. Contenido de la tesis	10
2. Marco Teórico	12

2.1. Modelo del UAV	12
2.2. Arquitectura del UAV	17
2.2.1. Esquema de control	17
2.3. Sistema Difuso	18
2.4. Base de reglas del sistema difuso	20
3. Resultados en simulación	21
3.1. Simulación de UAV	21
3.2. Animación	25
4. Resultados experimentales	26
4.1. Reporte técnico del prototipo	26
4.1.1. Diseño	26
4.1.2. Hardware	28
4.1.3. Software	34
4.1.4. Funcionamiento	34
4.2. Identificación de parámetros	35
4.3. Obteniendo ganancias del controlador para el prototipo construido . .	39
4.4. Resultados	43
5. Conclusiones	46
A. Código de Simulación de UAV (Scilab)	51

B. Reglas del Sistema Difuso	85
B.1. Reglas para el ajuste de K_p	85
B.2. Reglas para el ajuste de K_i	87
B.3. Reglas para el ajuste de K_d	89

ÍNDICE DE FIGURAS

1.1. Diversos tipos de UAVs.	2
1.2. Controlador MultiWii SE V2.5	2
1.3. Estructura mecánica de un cuadricóptero	3
1.4. Aplicaciones de los cuadricópteros	3
1.5. Diagrama de controlador PID	5
1.6. Control PID + Lógica Difusa (izquierda) [1]. Control PID + Método Empírico (derecha) [19]	6
1.7. Control PID + Lógica Difusa + Analizador de Señal	7
1.8. Ciclo de tendencia en tecnologías emergentes (Fuente: Gartner - Julio 2016)	8
2.1. Tramas inercial (izq.) y de cuerpo (der.) del cuadricóptero.	12
2.2. Pares en ángulos de Euler i.e. Roll/Pitch (Izquierda) y Yaw (Derecha)	15
2.3. Esquema de control para lograr la estabilización de vuelo estacionario	18
2.4. Sistema difuso general [24]	18
2.5. Cuantificación de valores de entrada	19

2.6. Magnitud del incremento/decremento en el ajuste de ganancia	19
3.1. Posición angular: Ziegler Nichols, Sintonización Manual, Sistema Difuso	23
3.2. Velocidad angular: Ziegler Nichols, Sintonización Manual, Sistema Di- fuso	23
3.3. Posición lineal: Ziegler Nichols, Sintonización Manual, Sistema Difuso	24
3.4. Velocidad lineal: Ziegler Nichols, Sintonización Manual, Sistema Difuso	24
3.5. Animación de la simulación	25
4.1. Vista explosionada de la estructura del cuadricóptero	27
4.2. Batería de Tecnología Li-Po	28
4.3. Cargador/Balaceador de Tecnología Li-Po 50W/6A con capacidad de LiHV Accucell-6	29
4.4. Sistema R/C digital proporcional	29
4.5. Controlador de vuelo estándar CRIUS MWC MultiWii SE V2.6	31
4.6. Motores MT1806 2280KV	32
4.7. Controlador de velocidad para motor	32
4.8. Propulsores utilizados	33
4.9. Microcontrolador Arduino Pro Mini y Lector SD	33
4.10. MultiWiiConf	34
4.11. Modelo CAD del cuadricóptero	36
4.12. Impresión 3D de la estructura del cuadricóptero y sus componentes en el lugar correspondiente	36

4.13. Experimento 1. Par generado en los ángulos de ϕ y θ	37
4.14. Experimento 2. Par generado en el ángulo ψ	38
4.15. Control y grabación de orientación en el cuadricóptero	43
4.16. Medición del ángulo ϕ	44
4.17. Medición del ángulo θ	44
4.18. Prueba de vuelo	45

ÍNDICE DE TABLAS

2.1. Reglas para el comportamiento de ΔK_p	20
3.1. Parámetros de la simulación	22
4.1. Características de la batería Turnigy ®	28
4.2. Características del controlador de vuelo	30
4.3. Características de los motores	31
4.4. Datos del experimento 1	37
4.5. Datos del experimento 2	38
4.6. Parámetros del prototipo construido	39
4.7. Evolución de las ganancias e índices de desempeño para el ángulo ϕ .	40
4.8. Evolución de las ganancias e índices de desempeño para el ángulo θ .	41
4.9. Evolución de las ganancias e índices de desempeño para el ángulo ψ .	42

AGRADECIMIENTOS

Quisiera agradecer a todas las personas involucradas directa e indirectamente durante el desarrollo de esta tesis. Primeramente a quienes estuvieron colaborando de forma más cercana, como lo son mi asesor, el Dr. Luis Martín Torres-Treviño y el M.C. Mario Aguilera Ruiz, ya que sin su ayuda esto no habría sido posible. También quisiera agradecer al Dr. Juan Ángel Rodríguez Liñán por los aportes hechos a esta tesis y a todo el equipo de investigadores del área de Mecatrónica por darme esta oportunidad, haciendo mención especial de la Dra. Griselda Quiróz Compeán.

Agradezco a la Facultad de Ingeniería Mecánica y Eléctrica (FIME), la Universidad Autónoma de Nuevo León, al Consejo Nacional de Ciencia y Tecnología (CONACYT) por otorgarme la beca número 662142 y los medios para realizar esta investigación. Deseo agradecer a las personas que me acompañaron durante estos dos años y de quienes aprendí bastante: Mis padres (Lucy Zarazúa y Víctor Medrano), mis dos hermanas (Carolina Medrano y Lucy Medrano), Juan P. Saucedo, mis familiares, Vidal Trejo, Carlos Arvizu, Aldebarán Alonso, Jorge A. Villanueva, Antonio Zalapa, Miguel Tovar, Emilse Vázquez, Ana Leal, César Ledezma, Javier Ríos, Munster Rodríguez, Job Benítez, Luis González, Roberto de la Rosa, Sam Li, Tony Ribo, Emiliano Ramos, Ever Solís, Brandon Ramírez, Carolina Quintanilla, Martín García, Hugo Ruiz, Víctor Rangel, César Árzola, Jazmín Martínez, Toñita Gutiérrez, Humberto Huerta y seguramente muchos más que me faltó mencionar (por lo cual me disculpo, pues no ha sido intencional), con los quienes pase gratos momentos en este sinuoso pero fructuoso camino. Finalmente agradecer a aquellas personas que no conozco pero cuya obra habla e inspira a la humanidad, y que

ayudaron a que los momentos difíciles fueran más llevaderos.

NOMENCLATURAS

OS	Sobretiro de la respuesta en el tiempo
SSE	Error de estado estable de la respuesta en el tiempo
ST	Tiempo de asentamiento de la respuesta en el tiempo
RT	Tiempo de levantamiento de la respuesta en el tiempo
K_p	Ganancia proporcional del controlador PID
K_i	Ganancia integral del controlador PID
K_d	Ganancia derivativa del controlador PID
ΔK_p	Incremento en ganancia proporcional
ΔK_i	Incremento en ganancia integral
ΔK_d	Incremento en ganancia derivativa
\mathbf{R}_B^I	Matriz de rotación (de trama de cuerpo a trama inercial)
ξ	Vector de posición del cuadricóptero
x	Posición en el eje x del cuadricóptero
y	Posición en el eje y del cuadricóptero
z	Posición en el eje z del cuadricóptero
η	Vector de orientación del cuadricóptero
ϕ	Ángulo de rotación del cuadricóptero alrededor del eje x (Roll)
θ	Ángulo de rotación del cuadricóptero alrededor del eje y (Pitch)
ψ	Ángulo de rotación del cuadricóptero alrededor del eje z (Yaw)
ϕ_d	Ángulo roll deseado
θ_d	Ángulo pitch deseado
ψ_d	Ángulo yaw deseado

\mathbf{V}_B	Vector de velocidad lineal del cuadricóptero
$V_{x,B}$	Velocidad en el eje x del cuadricóptero
$V_{y,B}$	Velocidad en el eje y del cuadricóptero
$V_{z,B}$	Velocidad en el eje z del cuadricóptero
$\boldsymbol{\nu}$	Vector de velocidad angular del cuadricóptero
p	Velocidad angular del cuadricóptero alrededor del eje x_B
q	Velocidad angular del cuadricóptero alrededor del eje y_B
r	Velocidad angular del cuadricóptero alrededor del eje z_B
ω_i	Velocidad angular del motor i
f_i	Fuerza en la dirección del eje del motor i
τ_{M_i}	Par de torsión alrededor del eje del motor i
I_{M_i}	Momento de inercia del motor i
\mathbf{T}_B	Vector de la fuerza de empuje generada por los cuatro motores
T	Fuerza de empuje en la dirección del eje z_B
k	Factor de empuje
b	Factor de arrastre
k_d	Factor de fuerza de arrastre
\mathbf{F}_D	Vector de fuerza de arrastre
$\boldsymbol{\tau}_B$	Vector del par generado en cada ángulo de Euler
τ_{ϕ_B}	Par generado en el ángulo roll
τ_{θ_B}	Par generado en el ángulo pitch
τ_{ψ_B}	Par generado en el ángulo yaw
m	Masa del cuadricóptero
L	Distancia entre el rotor y el centro de masa del cuadricóptero
g	Fuerza de gravedad
t	Tiempo
\mathbf{I}	Matriz diagonal de inercias
I_{xx}	Momento de inercia alrededor del eje x_B
I_{yy}	Momento de inercia alrededor del eje y_B
I_{zz}	Momento de inercia alrededor del eje z_B

RESUMEN

Víctor de Jesús Medrano Zarazúa.

Candidato para obtener el grado de Maestro en Ciencias de la Ingeniería Eléctrica

Título del estudio: ESTABILIZACIÓN DE VUELO EN UAVS CON SINTONIZACIÓN INTELIGENTE DE CONTROLADORES CONVENCIONALES.

OBJETIVOS Y MÉTODO DE ESTUDIO: Los vehículos aéreos no tripulados han tenido un gran avance y popularidad durante los últimos años. La comunidad científica busca desarrollar soluciones innovadoras en este campo. En este trabajo se propone el desarrollo de un estabilizador de vuelo basado en la sintonización inteligente de un controlador convencional. La sintonización se logra evaluando la respuesta en el tiempo del sistema.

CONTRIBUCIONES Y CONCLUSIONES: Se logra el comportamiento estable de un vehículo aéreo por medio de un método alternativo y de bajo costo. Los resultados muestran que el método utilizado para sintonizar las ganancias de un controlador, tiene una mejor respuesta a un impulso en la velocidad angular que otros. El desarrollo del proyecto se llevó a cabo con herramientas de software libre para compartir la información de la misma manera.

CAPÍTULO 1

INTRODUCCIÓN

Un vehículo aéreo no tripulado o UAV (Unmanned Aerial Vehicle) es descrito como una aeronave motorizada capaz de realizar su misión sin un operador humano a bordo [16]. Éste puede ser usado para una serie de tareas donde el acceso a través del aire es la mejor solución, donde es innecesario o peligroso el uso de un vehículo tripulado o cuando un alto nivel de precisión y seguimiento de pasos es requerido [15]. Desde su llegada han extendido el potencial humano, permitiendo la ejecución de tareas peligrosas o difíciles de manera segura y eficiente, lo cual se traduce en ahorrar tiempo, dinero y lo más importante, salvar vidas [6].

Los UAVs han tenido avances sin precedentes durante las dos últimas décadas debido principalmente a sus aplicaciones civiles y militares (ver Figura 1.1) [2, 5]. Su tecnología ha crecido de manera radical recientemente, incorporando materiales compuestos ligeros, electrónica embebida avanzada (ver Figura 1.2) y algoritmos computacionales eficientes [11].



Figura 1.1: Diversos tipos de UAVs.

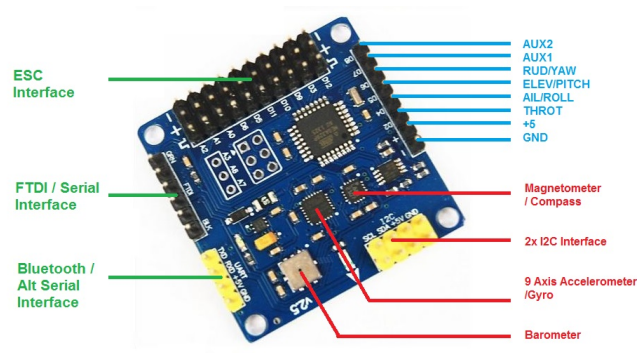


Figura 1.2: Controlador MultiWii SE V2.5

1.1 MOTIVACIÓN

En años recientes los cuadricópteros se han vuelto muy populares respecto a otros UAVs debido a su reducido tamaño, bajo costo, simplicidad de la mecánica, buena maniobrabilidad, capacidad de supervivencia y aumento de la carga útil [10].

El uso de hélices giratorias permite al cuadricóptero producir las fuerzas aerodinámicas de empuje necesarias para despegar y aterrizar verticalmente, permanecer estacionario y volar a bajas altitudes [5]. Los cuatro motores del cuadricóptero están localizados en la parte frontal, trasera, izquierda y derecha del marco transversal (ver Figura 1.3) [15].

Los cuadricópteros están ganando más importancia gracias a su fácil implementación en aplicaciones tales como inspección de construcciones después de un



Figura 1.3: Estructura mecánica de un cuadricóptero

desastre, misiones de rescate, transportación, agricultura/cultivo a distancia y captura de imágenes aéreas [12]. Algunas de estas aplicaciones pueden observarse en la Figura 1.4.

La estabilidad de vuelo estacionario (hover stability) de un cuadricóptero es una de las tareas importantes a realizar para aplicaciones dedicadas a tomar imágenes a bordo (inspección de construcciones, monitoreo de vida silvestre, vigilancia, etcétera) ya que requieren de imágenes claras para análisis posteriores. Ser estable durante el vuelo estacionario también previene al cuadricóptero de colisionar debido a fuertes vientos o a su propio peso [19].



Figura 1.4: Aplicaciones de los cuadricópteros

1.2 ANTECEDENTES

Desde su introducción en la década de 1950 [9], los vehículos aéreos no tripulados, también conocidos como UAVs (por sus siglas del inglés Unmanned Aerial Vehicles), han sido utilizados por la milicia para misiones consideradas sucias, peligrosas o poco intensas [16]. El primer UAV fue el Q-2 hecho por Ryan Aeronautical, volado en la década de 1950 para reconocimiento militar [26]. Muchos UAVs utilizados en la milicia pesan cientos o incluso miles de libras y pueden alcanzar una altitud de hasta 6000 pies. Además, la milicia también usa UAVs pequeños o micro como el Dragon Eye, FPASS, Pointer o Raven [23]. Los UAVs con aplicaciones militares son costosos en su desarrollo y mantenimiento, lo cual los hace inapropiados para aplicaciones civiles [9].

Desde la década de 1990, el surgimiento de las baterías de alta densidad de potencia (de iones de litio y polímero de litio), los equipos miniaturizados, y los dispositivos de redes inalámbricas permitieron que los UAVs pequeños fueran asequibles para investigadores y aficionados [9]. Este avance propició que el uso de UAVs haya crecido rápidamente durante los últimos años [8] y que el interés de la comunidad científica por el diseño de UAVs sea cada vez mayor con el objetivo de desarrollar vehículos más baratos y capaces. Este mismo interés por los UAVs para aplicaciones civiles y militares hizo necesario el desarrollo de la teoría de control de vuelo y algoritmos cada vez más eficientes y rápidos [2].

Con la madurez actual de la tecnología y el alto nivel de demanda, los UAVs están encontrando su camino dentro de las aplicaciones civiles. Con esta tendencia a la alza, los académicos e investigadores están ahora interesados en oportunidades de investigación para desarrollar soluciones únicas e innovadoras en el campo de los UAVs [18].

Existen diversas clasificaciones y tipos de UAVs [18], entre los cuales destacan los cuadricópteros, que están ganando más importancia debido a diversas ventajas

ya mencionadas (ver sección 1.1); razones por las que ha recibido más atención que otros micro UAVs [21].

Según Wang et. al, un vehículo cuadrimotor es un sistema no lineal sub-actuado de fuerte acoplamiento y múltiples variables. En el presente, la tecnología PID (ver Figura 1.5) se aplica ampliamente en el control de un vehículo cuadrimotor por sus características: estructura simple, tecnología madura, y fácil realización de la ingeniería en la práctica. Los controladores PID convencionales tienen las ventajas de tener estructura simple y parámetros ajustables, pero es excesivamente dependiente en el modelo preciso de un sistema cuadrimotor complejo con grandes perturbaciones, alta no linealidad e incertidumbre, por lo que los efectos de control esperados no son realizados de manera general [28].

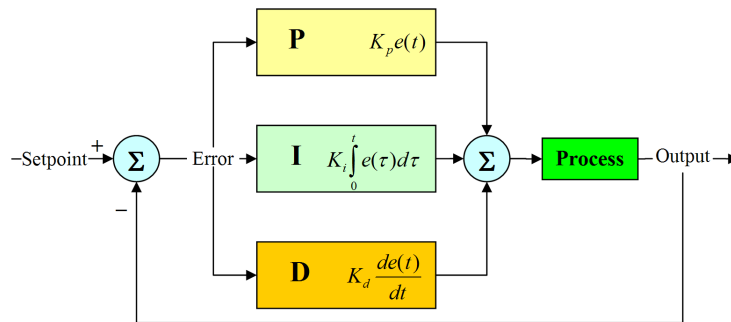


Figura 1.5: Diagrama de controlador PID

La sintonización o ajuste de parámetros es importante para un mejor desempeño de los controladores PID. Los controladores PID pueden ser sintonizados en una variedad de formas, incluyendo métodos convencionales como: sintonización manual, sintonización Ziegler-Nichols, sintonización Cohen-Coon y Z-N respuesta escalón. Los métodos anteriores tienen sus propias limitaciones en comparación con las técnicas de soft computing como GA, PSO y EP (por sus siglas del inglés *Genetic Algorithms*, *Particle Swarm Optimization* y *Evolutionary Programming* respectivamente). Según Gowrisankar y Kumar [14], las técnicas de *soft computing* han demostrado su excelencia en dar mejores resultados para características de estado estable e índices de desempeño, pues son capaces de lidiar con problemas no lineales, objetivos múlti-

ples y propiedades dinámicas de los componentes que aparecen frecuentemente en situaciones del mundo real. La lógica difusa en combinación con métodos de soft computing se ha utilizado en aplicaciones para diversos campos [7], pues usando mecanismos de inferencia verbales lidia de manera eficiente con los problemas de incertidumbre [22].

El controlador puede ser sintonizado sin cambiar el punto de operación, pero debe ser re-sintonizado si éste cambia o el proceso se altera con el tiempo. La lógica difusa ha tenido un buen efecto de control para procesos con características no lineales. La combinación de una ley de control PID (lineal) y una estrategia de sintonización como lógica difusa (no lineal) puede llegar a ser una ley de control altamente no lineal e incrementar significativamente la robustez del sistema de control [17]. Los esquemas de la Figura 1.6 han sido utilizados por [1] y [19] para la sintonización del controlador PID de un cuadricóptero.

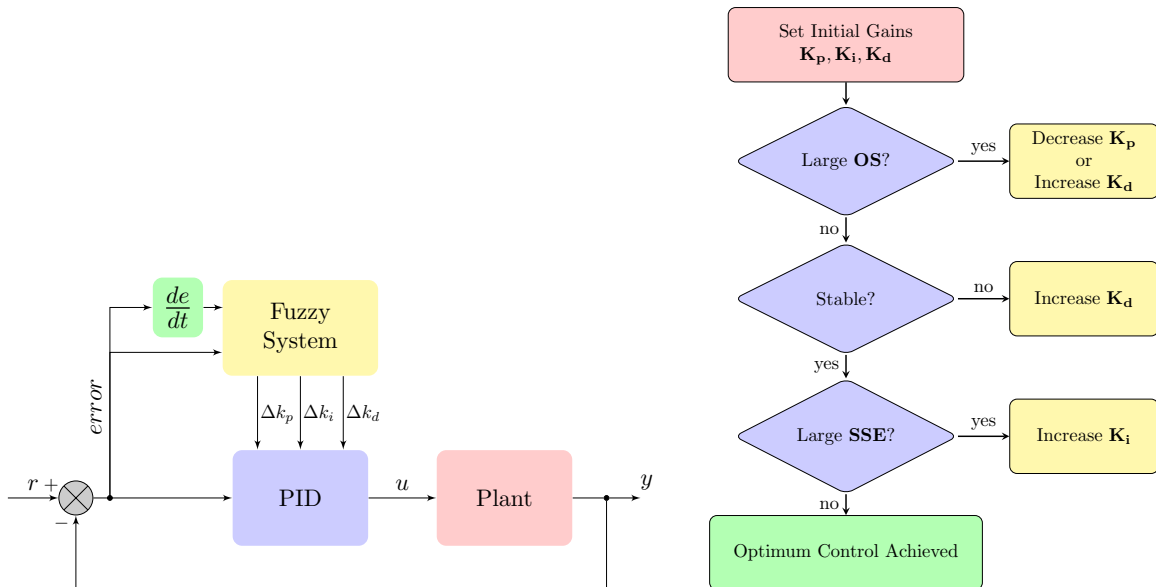


Figura 1.6: Control PID + Lógica Difusa (izquierda) [1]. Control PID + Método Empírico (derecha) [19]

El esquema de la Figura 1.7 fue propuesto por [4] en la sintonización inteligente de un control PID para un motor de CD, obteniendo los índices de desempeño de la señal con los datos de la variable a controlar. Este esquema también fue utilizado

por [3] para controlar la temperatura en celdas de Peltier. Esto resulta de interés para controlar un UAV por medio de este mismo esquema.

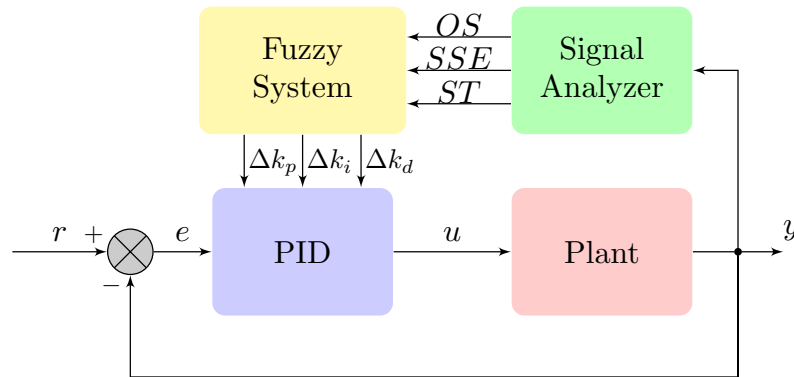


Figura 1.7: Control PID + Lógica Difusa + Analizador de Señal

1.3 JUSTIFICACIÓN

- Participar en la investigación de UAVs debido a sus actuales y potenciales aplicaciones de manera individual, así como en enjambres. Según Gartner, una de las tres tendencias en las tecnologías emergentes actuales es la construcción de máquinas perceptivas inteligentes, dentro de las cuales están clasificados los UAVs. Los UAVs aún tienen grandes expectativas en cuanto a sus aplicaciones (ver Figura 1.8)
- Ahorrar recursos humanos y económicos en las distintas aplicaciones donde UAVs son utilizados. Con UAVs a su disposición, las compañías pueden ahorrar dinero recopilando información con menos mantenimiento y sin riesgo a la seguridad humana. Por ejemplo, en enero de 2012, la oficina del sheriff del condado de Mesa (en el estado de Colorado, EE. UU.) fue pionera en el uso de la aplicación de la ley de UAVs. La agencia ya ha utilizado la tecnología para proveer fotografías aéreas de accidentes automovilísticos y apoyar al departamento de bomberos a combatir incendios detectando puntos de emergencia; además de ver a los UAVs como una herramienta de búsqueda de ayuda y esfuerzos

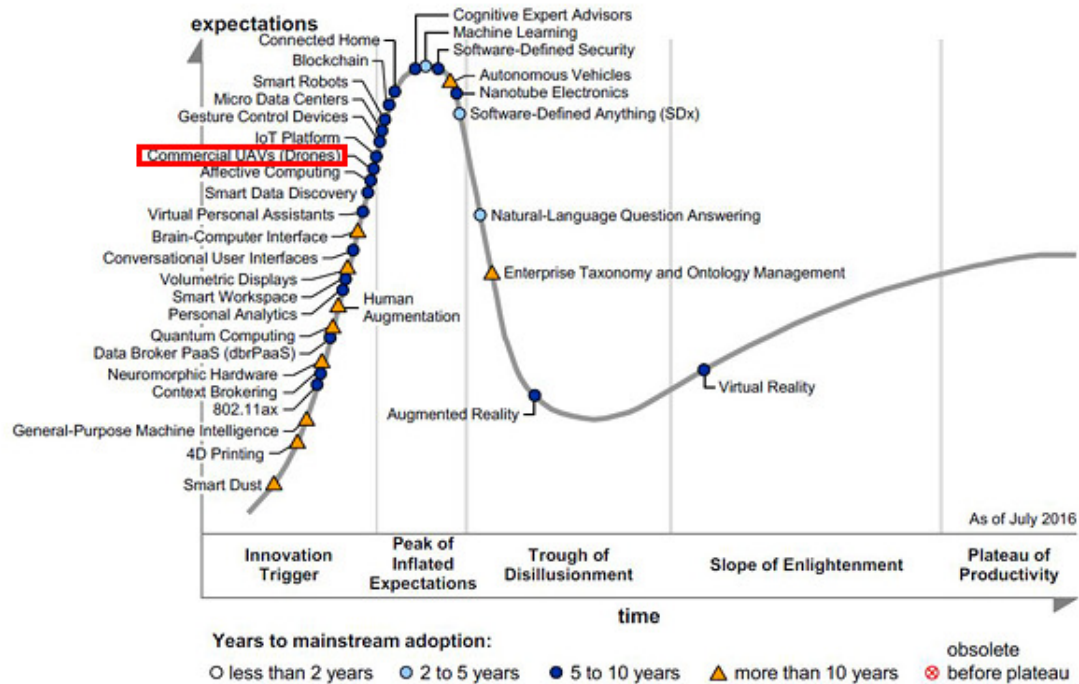


Figura 1.8: Ciclo de tendencia en tecnologías emergentes (Fuente: Gartner - Julio 2016)

de rescate. El uso de los UAVs en estas aplicaciones es menos costoso, ya que el costo total según la AUVSI (Association for Unmanned Vehicle Systems International) es de USD\$3.36 por hora, comparado al precio de USD\$250-\$600 por hora para un vehículo tripulado [6].

- Compartir libremente con la comunidad científica y demás interesados, los procedimientos y resultados de la investigación.

1.4 HIPÓTESIS

Se consigue un mejor desempeño (en comparación con métodos convencionales de sintonización) en un estabilizador de vuelo de un vehículo cuadrirrotor, si es supervisado por medio de un sistema inteligente que lo sintonice autónomamente a través de los índices de desempeño del controlador.

1.5 OBJETIVOS

- Objetivo general:

Desarrollar un estabilizador de vuelo basado en sintonización inteligente de un controlador convencional y apoyado por la respuesta en el tiempo del sistema.

- Objetivos particulares:

- Seleccionar modelo matemático del UAV.
- Sintonizar control de estabilización de vuelo estacionario del UAV por medio de un sistema difuso.
- Simular modelo, control y sintonizador de manera integrada.
- Implementar en un ambiente o banco de pruebas.

1.6 METODOLOGÍA

A lo largo de este trabajo se realizaron las siguientes tareas:

1. Revisión bibliográfica.
2. Definir estructura del UAV.

3. Seleccionar modelo dinámico del UAV.
4. Diseñar sintonizador inteligente y control.
5. Simulaciones de vuelo para el UAV.
6. Construir prototipo del UAV.
7. Identificar parámetros y obtener ganancias.
8. Pruebas reales de vuelo.

1.7 CONTRIBUCIONES

En el desarrollo de esta tesis se llegaron a las siguientes contribuciones:

- Proponer un método alternativo para la estabilización de un UAV.
- Desarrollar un esquema con software libre, capaz de evaluar los parámetros del controlador por medio de la respuesta en el tiempo del UAV y de hacer un reajuste.
- Conseguir un buen desempeño en la estabilidad de un UAV con herramientas de bajo costo.

1.8 CONTENIDO DE LA TESIS

El resto de este documento está organizado de la siguiente forma:

El capítulo 2 muestra el modelo matemático del UAV utilizado y explica como es posible lograr su control mediante el accionamiento de los propulsores del vehículo a la velocidad angular calculada. También se presenta el esquema propuesto, cuya

parte principal es el sistema difuso. Al final del capítulo se explica como funcionan las reglas que rigen al sistema difuso.

El capítulo 3 muestra los resultados de la simulación (con parámetros propuestos por [13]) y la realización de una animación 3D para presentar de una manera más ilustrativa el comportamiento del UAV durante su estabilización.

El capítulo 4 explica como se desarrolló el prototipo físico y la obtención de sus parámetros para hallar las ganancias que permitan un buen desempeño. Posteriormente se muestran los resultados experimentales (se llevaron a cabo pruebas de vuelo) después de programar el controlador con las ganancias propuestas por la simulación.

Por último, se muestran las conclusiones, así como el posible trabajo a futuro.

CAPÍTULO 2

MARCO TEÓRICO

El presente capítulo muestra el modelo matemático del UAV y una breve explicación de las variables cruciales para lograr su control. Posteriormente, se presenta el esquema propuesto y cómo funciona el sistema difuso. Por último, se expone una pequeña muestra de las reglas que rigen al sistema difuso para lograr un mejor ajuste de las ganancias de un controlador PID.

2.1 MODELO DEL UAV

Considere la Figura 2.1 para entender el modelo cinemático y dinámico del cuadricóptero.

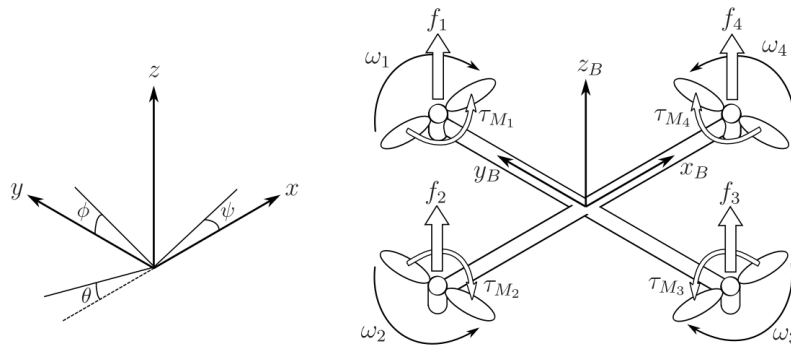


Figura 2.1: Tramas inercial (izq.) y de cuerpo (der.) del cuadricóptero.

La posición absoluta lineal del cuadricóptero es definida en los ejes x, y, z de la trama inercial con $\boldsymbol{\xi}$. La orientación, i.e. la posición angular, es definida en la trama inercial con tres ángulos de Euler mediante $\boldsymbol{\eta}$. El ángulo pitch θ determina la rotación del cuadricóptero alrededor del eje y . El ángulo roll ϕ determina la rotación alrededor del eje x y el ángulo yaw ψ alrededor del eje z (ver Ecuación 2.1).

$$\boldsymbol{\xi} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \boldsymbol{\eta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \quad (2.1)$$

La matriz de rotación \mathbf{R} que relaciona la orientación de las tramas inercial y de cuerpo se muestra en la Ecuación 2.2. Esta matriz se deriva de la convención XYZ de los ángulos de Euler.

$$\mathbf{R}_{\mathbf{B}}^{\mathbf{I}} = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - c\phi s\psi & c\psi s\theta c\phi + s\psi s\phi \\ s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \quad (2.2)$$

En la trama de cuerpo, las velocidades lineales son determinadas por $\mathbf{V}_{\mathbf{B}}$ y las velocidades angulares por $\boldsymbol{\nu}$ (ver Ecuación 2.3).

$$\mathbf{V}_{\mathbf{B}} = \begin{bmatrix} V_{x,B} \\ V_{y,B} \\ V_{z,B} \end{bmatrix}, \quad \boldsymbol{\nu} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (2.3)$$

La matriz Jacobiana \mathbf{J} que relaciona las velocidades angulares de las tramas de cuerpo e inercial se muestra en las Ecuaciones 2.4 y 2.5. Vempati et. al. [27] explican a detalle y de manera clara como se obtiene esta matriz Jacobiana a través del modelo del cuadricóptero.

$$\mathbf{J} = \begin{bmatrix} 1 & 0 & -s\theta \\ 0 & c\phi & c\theta s\phi \\ 0 & -s\phi & c\theta c\phi \end{bmatrix} \quad (2.4)$$

$$\boldsymbol{\nu} = \mathbf{J}\dot{\boldsymbol{\eta}}, \quad \dot{\boldsymbol{\eta}} = \mathbf{J}^{-1}\boldsymbol{\nu}, \quad (2.5)$$

Supongamos que el cuadricóptero tiene estructura simétrica con los cuatro brazos alineados en los ejes x_B y y_B en la trama de cuerpo. De este modo, la matriz de inercia es la matriz diagonal \mathbf{I} , en la cual $I_{xx} = I_{yy}$. Los tres elementos de la matriz \mathbf{I} representan los momentos de inercia alrededor de los ejes de la trama de cuerpo (ver Ecuación 2.6).

$$\mathbf{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \quad (2.6)$$

La velocidad angular del motor i , denotado por ω_i , crea una fuerza f_i en la dirección del eje del motor. La velocidad y aceleración angular del motor también crean un par de torsión τ_{M_i} alrededor del eje del motor

$$f_i = k\omega_i^2, \quad \tau_{M_i} = b\omega_i^2 + I_{M_i}\dot{\omega}_i \quad (2.7)$$

en donde k es la constante de empuje, b es la constante de arrastre y el momento de inercia del motor i es I_{M_i} . Usualmente el efecto de $\dot{\omega}_i$ es considerado pequeño y por lo tanto omitido.

Las fuerzas combinadas de los motores crean una fuerza de empuje \mathbf{T}_B . El par de torsión resultante $\boldsymbol{\tau}_B$ consiste de los pares de torsión τ_{ϕ_B} , τ_{θ_B} y τ_{ψ_B} en la dirección de los ángulos de la trama de cuerpo correspondientes (ver Ecuaciones 2.8 y 2.9).

$$\mathbf{T}_B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^4 f_i \end{bmatrix} = k \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^4 \omega_i^2 \end{bmatrix} \quad (2.8)$$

$$\boldsymbol{\tau}_B = \begin{bmatrix} \tau_{\phi_B} \\ \tau_{\theta_B} \\ \tau_{\psi_B} \end{bmatrix} = \begin{bmatrix} Lk(\omega_1^2 - \omega_3^2) \\ Lk(\omega_2^2 - \omega_4^2) \\ \sum_{i=1}^4 \tau_{M_i} \end{bmatrix} = \begin{bmatrix} Lk(\omega_1^2 - \omega_3^2) \\ Lk(\omega_2^2 - \omega_4^2) \\ b(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \end{bmatrix} \quad (2.9)$$

Donde T es la fuerza de empuje en la dirección del eje z_B de la trama de cuerpo y L es la distancia entre el motor y el centro de masa del cuadricóptero. Por lo tanto, el movimiento alrededor del eje x_B es conseguido disminuyendo la velocidad del motor 3 e incrementando la velocidad del motor 1 (o viceversa). Similarmente, el movimiento alrededor del eje y_B es conseguido disminuyendo la velocidad del motor 4 e incrementando la velocidad del motor 2 (o viceversa). El movimiento alrededor del eje z_B es conseguido incrementando las velocidades angulares de dos motores que se encuentren sobre el mismo eje (e.g: Motor 1 y 3) y disminuyendo la velocidad angular de los otros dos (e.g: Motor 2 y 4). La Figura 2.2 ilustra cada uno de estos movimientos.



Figura 2.2: Pares en ángulos de Euler i.e. Roll/Pitch (Izquierda) y Yaw (Derecha)

Además de la fuerza de empuje, se puede modelar la fuerza de arrastre o la fricción (debido a la resistencia del aire) como una fuerza proporcional a la velocidad lineal en cada dirección. La Ecuación 2.10 es una visión muy simplificada de la fricción en fluidos.

$$\mathbf{F}_D = \begin{bmatrix} -k_d \dot{x} \\ -k_d \dot{y} \\ -k_d \dot{z} \end{bmatrix} \quad (2.10)$$

Donde k_d es el factor de la fuerza de arrastre. Si se desea precisión adicional, la constante k_d puede ser separada en tres constantes de fricción diferentes, una para cada dirección de movimiento.

Suponiendo que el cuadricóptero es un cuerpo rígido, la segunda ley de Newton puede ser usada para describir la dinámica traslacional del cuadricóptero en la trama inercial, esto es

$$m\ddot{\boldsymbol{\xi}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \mathbf{R}_B^I \mathbf{T}_B + \mathbf{F}_D \quad (2.11)$$

donde m es la masa del cuadricóptero y g es la fuerza de gravedad.

De la ecuación $\boldsymbol{\tau} = I\dot{\boldsymbol{\nu}} + \boldsymbol{\nu} \times (I\boldsymbol{\nu})$, la dinámica rotacional es expresada por

$$\dot{\boldsymbol{\nu}} = \begin{bmatrix} \tau_{\phi_B} I_{xx}^{-1} \\ \tau_{\theta_B} I_{yy}^{-1} \\ \tau_{\psi_B} I_{zz}^{-1} \end{bmatrix} - \begin{bmatrix} \frac{I_{yy} - I_{zz}}{I_{xx}} qr \\ \frac{I_{zz} - I_{xx}}{I_{yy}} pr \\ \frac{I_{xx} - I_{yy}}{I_{zz}} pq \end{bmatrix} \quad (2.12)$$

Se utiliza una ley de control PID para lograr la estabilización de vuelo estacionario. Los pares de torsión requeridos para este tipo de estabilización se obtienen mediante las Ecuaciones 2.13-2.15.

$$\tau_{\phi_B} = (K_{\phi,D}(\dot{\phi}_d - \dot{\phi}) + K_{\phi,P}(\phi_d - \phi) + K_{\phi,I} \int (\phi_d - \phi)) I_{xx} \quad (2.13)$$

$$\tau_{\theta_B} = (K_{\theta,D}(\dot{\theta}_d - \dot{\theta}) + K_{\theta,P}(\theta_d - \theta) + K_{\theta,I} \int (\theta_d - \theta)) I_{yy} \quad (2.14)$$

$$\tau_{\psi_B} = (K_{\psi,D}(\dot{\psi}_d - \dot{\psi}) + K_{\psi,P}(\psi_d - \psi) + K_{\psi,I} \int (\psi_d - \psi)) I_{zz} \quad (2.15)$$

Donde ϕ_d , θ_d y ψ_d son los ángulos de Euler deseados, mientras que $K_{\alpha,\beta}$ es la ganancia destinada a cada ángulo de Euler α (ϕ , θ o ψ) del tipo β (P : Proporcional, I : Integral o D : Derivativa).

Posteriormente se calculan las velocidades angulares necesarias en los motores para alcanzar los pares de torsión requeridos mediante las ecuaciones 2.16-2.19.

$$\omega_1^2 = \frac{T}{4k} - \frac{\tau_{\theta_B}}{2kL} - \frac{\tau_{\psi_B}}{4b} \quad (2.16)$$

$$\omega_2^2 = \frac{T}{4k} - \frac{\tau_{\phi_B}}{2kL} + \frac{\tau_{\psi_B}}{4b} \quad (2.17)$$

$$\omega_3^2 = \frac{T}{4k} + \frac{\tau_{\theta_B}}{2kL} - \frac{\tau_{\psi_B}}{4b} \quad (2.18)$$

$$\omega_4^2 = \frac{T}{4k} + \frac{\tau_{\phi_B}}{2kL} + \frac{\tau_{\psi_B}}{4b} \quad (2.19)$$

2.2 ARQUITECTURA DEL UAV

2.2.1 ESQUEMA DE CONTROL

Se utiliza el esquema propuesto por [3] y [4] (ver figura 2.3) para la estabilización de vuelo estacionario de un cuadricóptero. En este esquema, la respuesta en el tiempo del cuadricóptero es evaluada (a través del OS, SSE y ST) para después hacer una sintonización o reajuste en las ganancias de los controladores PIDs de las Ecuaciones 2.13-2.15.

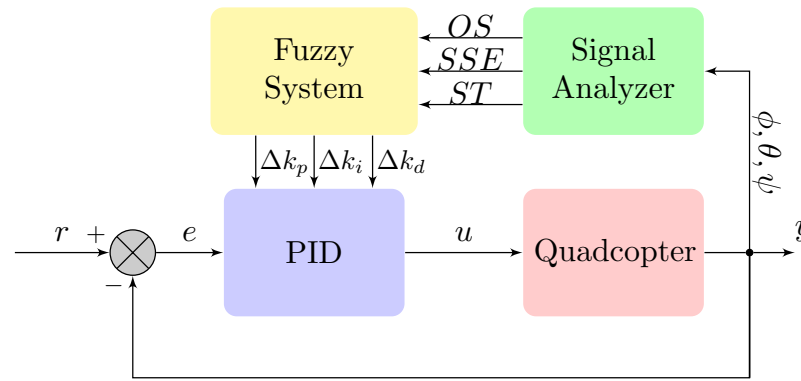


Figura 2.3: Esquema de control para lograr la estabilización de vuelo estacionario

2.3 SISTEMA DIFUSO

El sistema difuso es la parte más importante del esquema presentado, pues se encarga de reducir la magnitud del error que entra al controlador PID de acuerdo a un conjunto de reglas. El objetivo principal de éste será mejorar la respuesta del cuadricóptero en cada iteración.

Las partes principales del sistema difuso utilizado (ver figura 2.4) se describen a continuación:

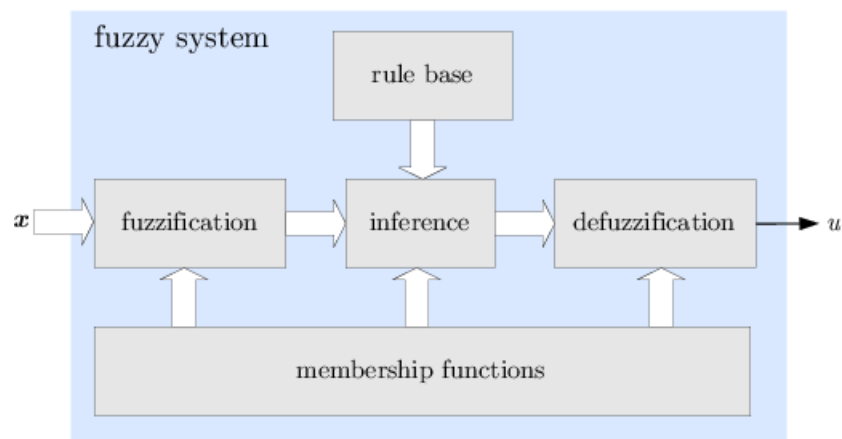


Figura 2.4: Sistema difuso general [24]

Fusificador: Las entradas del sistema difuso son normalizadas (los valores de entrada toman valores mapeados entre 0 y 1). De acuerdo a la magnitud del valor

normalizado, la entrada está sujeta a un término lingüístico (VL: Muy bajo, L: Bajo, M: Medio, H: Alto, VH: Muy alto) por medio de un valor de membresía que se asigna con una función predeterminada (ver figura 2.5).

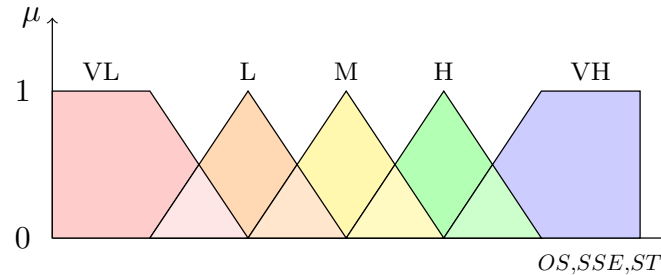


Figura 2.5: Cuantificación de valores de entrada

Base de reglas: Reglas del tipo SI-ENTONCES son almacenadas en esta base. Las reglas son proporcionadas por expertos que deciden como se comporta el sistema.

Mecanismo de inferencia: Utiliza los datos del fusificador (valores de membresía), la base de reglas y otras operaciones para regir el proceso de razonamiento difuso. En este caso, el mecanismo de inferencia determina si habrá un incremento o decremento en la ganancia.

Defusificador: La salida difusa del mecanismo de inferencia (VN: Muy negativo, N: Negativo, Z: Nulo, P: Positivo, VP: Muy positivo) es procesada para que sea comprensible a un mecanismo que procesa información numérica, retornando un valor exacto a la salida, el cual determina el ajuste en la ganancia (ver figura 2.6).

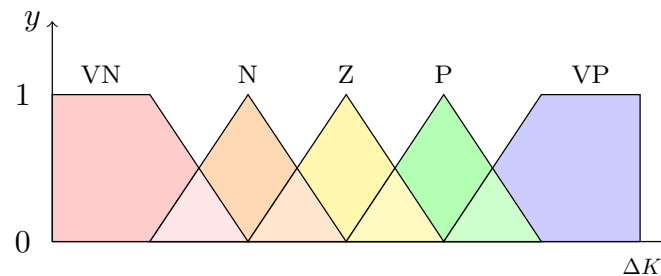


Figura 2.6: Magnitud del incremento/decremento en el ajuste de ganancia

2.4 BASE DE REGLAS DEL SISTEMA DIFUSO

El sistema difuso del esquema de la figura 12 se rige por ciertas reglas para ajustar las ganancias del controlador PID. La Tabla 2.1 es una muestra de cinco reglas (las reglas completas pueden verse en el Apéndice B) para incrementar o reducir la ganancia K_p .

OS	SS	ST	ΔK_p
MB	MB	MB	N
B	B	M	+
M	M	MA	N
MA	B	MB	-
MA	MA	MA	++

Tabla 2.1: Reglas para el comportamiento de ΔK_p

Traducción al lenguaje verbal de la primera y última regla del cuadro 1:

- SI sobretiro es MUY BAJO, error de estado estable es MUY BAJO y tiempo de asentamiento es MUY BAJO: ENTONCES incremento en K_p es NINGUNO.
- SI sobretiro es MUY ALTO, error de estado estable es MUY ALTO y tiempo de asentamiento es MUY ALTO: ENTONCES incremento en K_p es MUY POSITIVO.

CAPÍTULO 3

RESULTADOS EN SIMULACIÓN

El siguiente capítulo muestra los datos requeridos para realizar la simulación y los datos obtenidos después de ser ejecutada, ya que de esta manera es posible predecir el comportamiento del UAV con diferentes ganancias de prueba en el controlador PID. Posteriormente, se muestra una animación hecha para corroborar de forma visual que el UAV efectivamente se estabiliza.

3.1 SIMULACIÓN DE UAV

El esquema de control de la Figura 2.3 fue simulado en el software abierto Scilab (ir a Apéndice A para ver el código de simulación) para ilustrar la estabilización de vuelo estacionario a través de los controladores PID cuyos parámetros son ajustados por medio del sistema difuso.

Los parámetros numéricos para las simulaciones se encuentran en la Tabla 3.1. Estos valores corresponden a los mismos utilizados por Gibiansky en una simulación hecha en MATLAB y basados en un prototipo construido por él mismo [13].

Para propósitos de comparación, las ganancias de los controladores PID fueron ajustadas a través de tres métodos diferentes: 1) Ziegler-Nichols, 2) Sintonización Manual y 3) Sintonización por Sistema Difuso.

Parámetro	Valor	Unidad
g	9.81	m/s^2
m	0.5	kg
L	0.25	m
k	3×10^{-6}	$kg \cdot m$
b	1×10^{-7}	$kg \cdot m^2$
k_d	0.25	kg/s
I_{xx}	5×10^{-3}	$kg \cdot m^2$
I_{yy}	5×10^{-3}	$kg \cdot m^2$
I_{zz}	1×10^{-2}	$kg \cdot m^2$

Tabla 3.1: Parámetros de la simulación

La simulación consiste en someter al cuadricóptero a un impulso en la velocidad angular inicial y además conseguir que llegue a una altura deseada $z = 2$. Una vez que se llega a la altura deseada, se observa a través de los resultados como el cuadricóptero se estabiliza para lograr el vuelo estacionario.

En las Figuras 3.1 - 3.4 se muestra la comparativa entre los tres métodos para observar la evolución de cuatro variables diferentes (posición angular, velocidad angular, posición lineal y velocidad lineal respecto a la trama inercial).

■ Posición angular del UAV

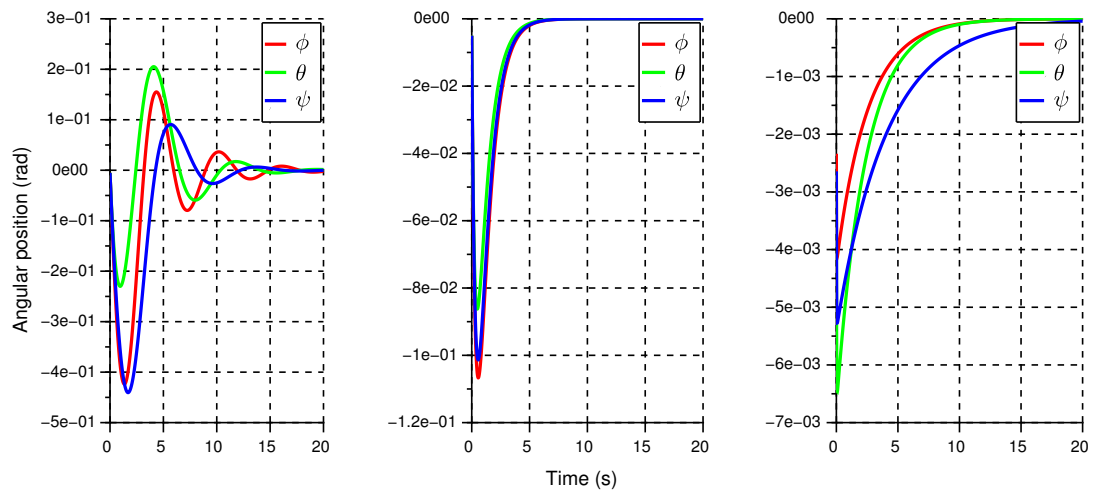


Figura 3.1: Posición angular: Ziegler Nichols, Sintonización Manual, Sistema Difuso

■ Velocidad angular del UAV

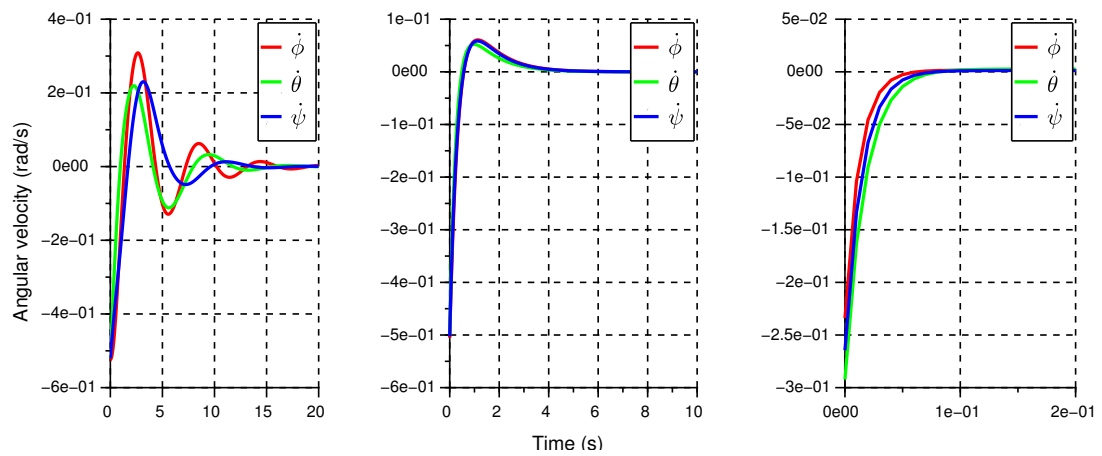


Figura 3.2: Velocidad angular: Ziegler Nichols, Sintonización Manual, Sistema Difuso

■ Posición lineal del UAV

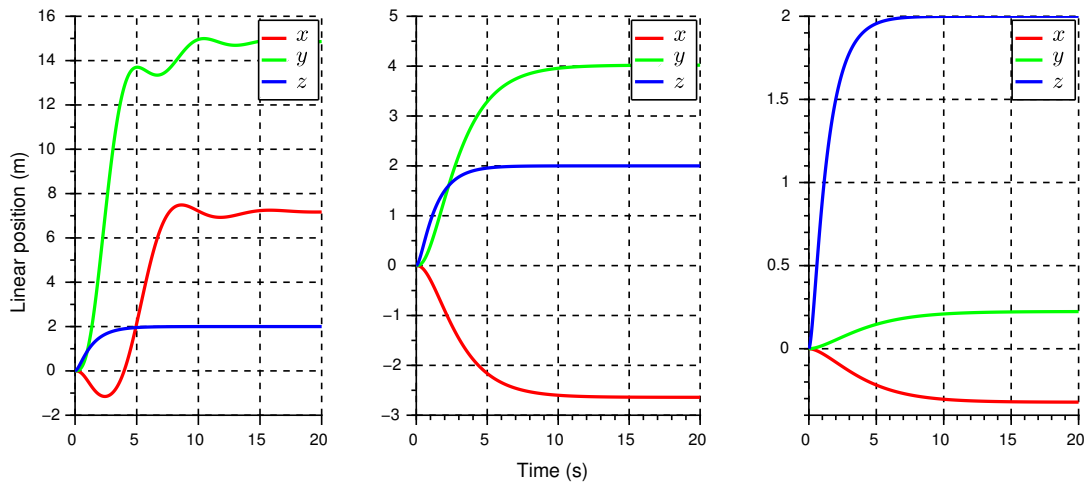


Figura 3.3: Posición lineal: Ziegler Nichols, Sintonización Manual, Sistema Difuso

■ Velocidad lineal del UAV

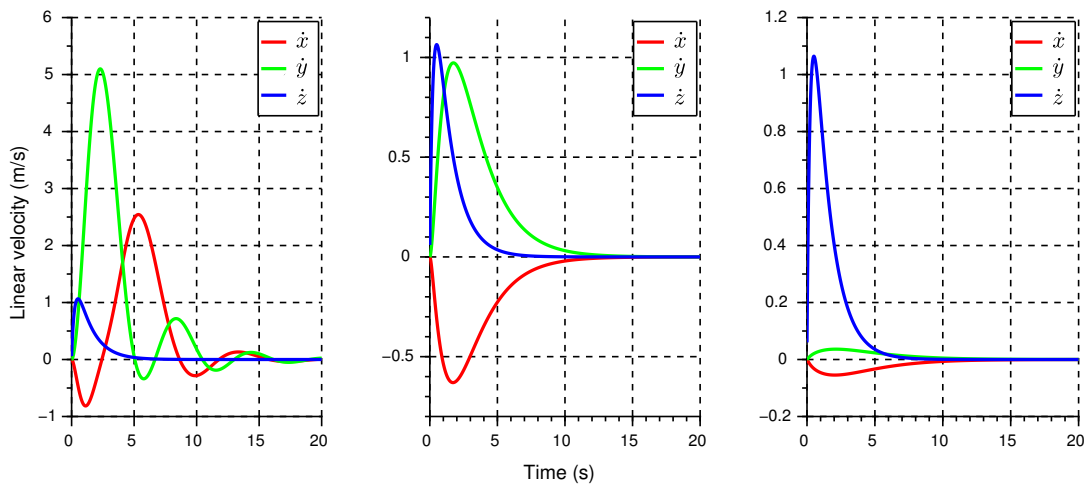


Figura 3.4: Velocidad lineal: Ziegler Nichols, Sintonización Manual, Sistema Difuso

3.2 ANIMACIÓN

La simulación es ilustrada gráficamente en una animación elaborada por medio de Scilab con los resultados del método de sistema difuso (ver Figura 3.5). En esta figura se observa el comportamiento del cuadricóptero en distintos instantes de tiempo.

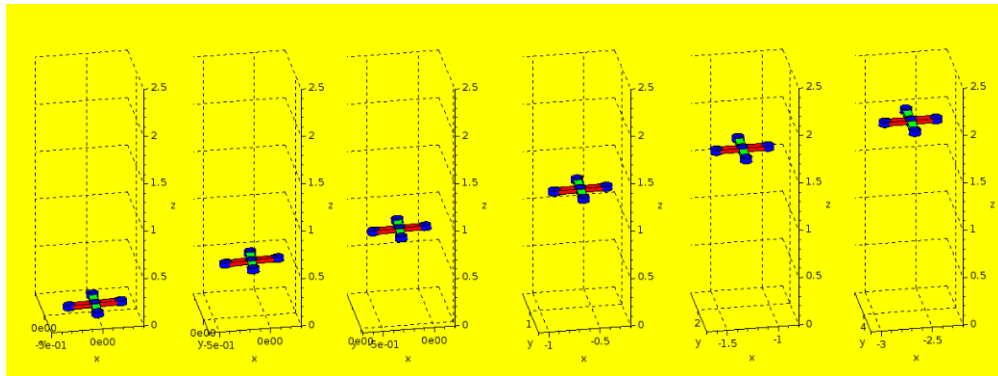


Figura 3.5: Animación de la simulación

La animación fue realizada con ayuda del sitio web de Sky Engineering Laboratory [25]. El código puede verse en el Apéndice B (Código 3).

CAPÍTULO 4

RESULTADOS EXPERIMENTALES

4.1 REPORTE TÉCNICO DEL PROTOTIPO

4.1.1 DISEÑO

El diseño del cuadricóptero (ver Figura 4.1) se compone de siete piezas impresas con tecnología 3D consistentes de un material conocido como ácido poliláctico (PLA), el cual es un polímero biodegradable. Sobre la cubierta superior, va colocado el controlador de vuelo, el receptor de radio y un registrador de datos del vuelo. Los cuatro brazos separan las cubiertas superior e inferior. En el extremo más alejado del centro del cuadricóptero de cada brazo y sobre este mismo, van posicionados los motores. El controlador de velocidad de los motores se unió a la parte inferior de cada brazo por medio de un sincho de velcro. La batería va colocada entre las cubiertas superior e inferior en la orientación indicada. Bajo la cubierta inferior se colocaron unas pequeñas gomas para amortiguar el aterrizaje del vehículo.

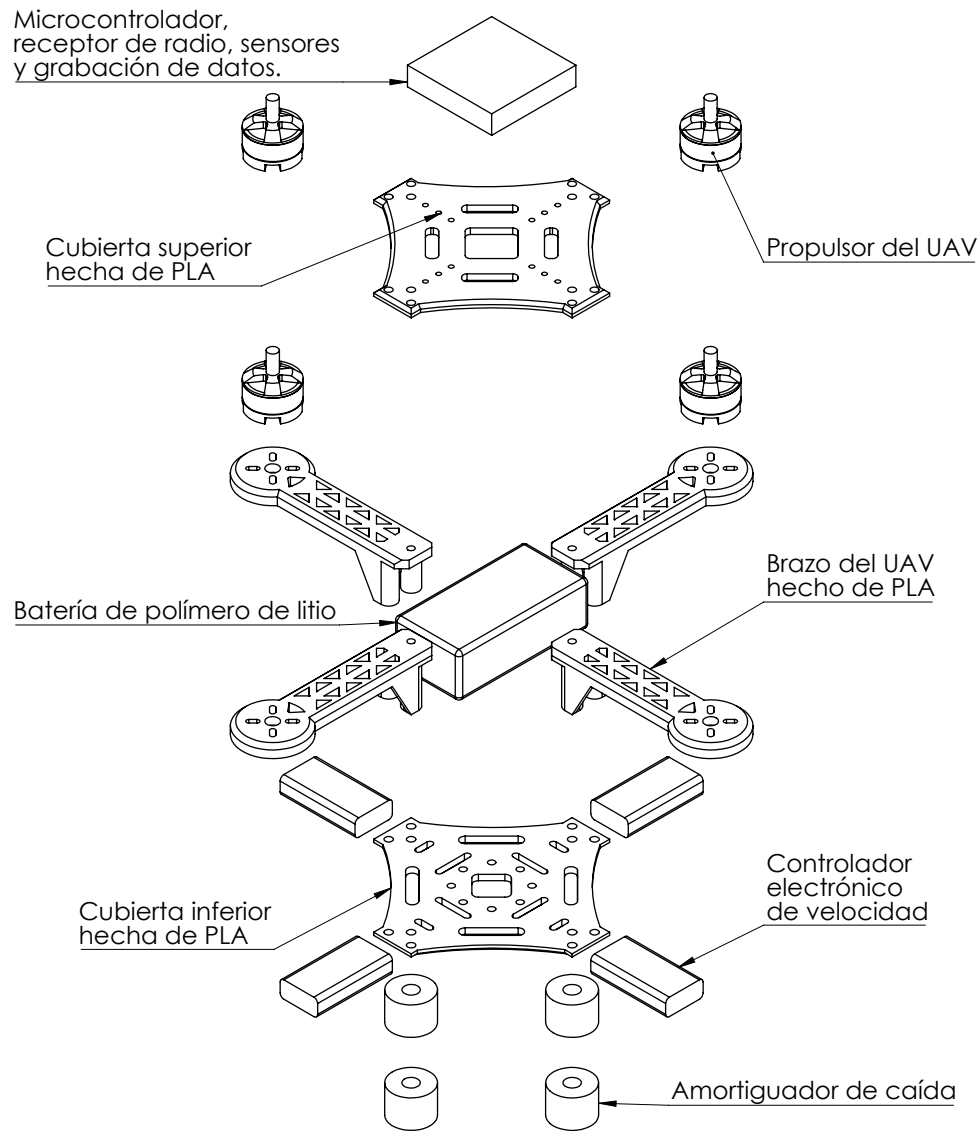


Figura 4.1: Vista explosionada de la estructura del cuadricóptero

4.1.2 HARDWARE

4.1.2.1 ALIMENTACIÓN

Todos los componentes electrónicos del cuadricóptero se alimentan a través de una batería de tecnología Li-Po (ver Figura 4.2). Esta batería es colocada en el interior del vehículo para evitar daños por colisiones. La Tabla 4.1 muestra sus principales características.



Figura 4.2: Batería de Tecnología Li-Po

Capacidad	1300mAh
Configuración	3s
Descarga	25c
Masa	119g
Max. índice de carga	5
Longitud	70mm
Altura	34mm
Anchura	22mm

Tabla 4.1: Características de la batería Turnigy ®

Es necesario contar con un cargador de baterías (ver Figura 4.3), ya que realizar pruebas de vuelo con un nivel de batería bajo causa que ésta misma se dañe.



Figura 4.3: Cargador/Balancedor de Tecnología Li-Po 50W/6A con capacidad de LiHV Accucell-6

4.1.2.2 TRANSMISIÓN Y RECEPCIÓN

El sistema R/C se encarga de comunicar inalámbricamente las señales enviadas por el transmisor, al receptor que se encuentra en el UAV. El sistema utilizado es el modelo Futaba® 9C (ver Figura 4.4)



Figura 4.4: Sistema R/C digital proporcional

4.1.2.3 CONTROLADOR DE VUELO

Las acciones a realizar por el cuadricóptero son recibidas, procesadas y controladas a través del controlador de vuelo (ver Figura 4.5). Las principales características se enlistan en la Tabla 4.2.

Microcontrolador	ATMega328P
Voltaje de operación	5V
Canales de entrada para receptor	6
Salidas (Propulsores)	8
Salidas (ϕ y θ de cámara integrada)	2
Masa	9.3g
Dimension	40mm x 40mm
Altura	11.6mm
Sensores integrados	MPU6050C 6 ejes giroscopio / acelerómetro con Processing Unit Motion; HMC5883L 3 ejes magnetómetro digital; Sensor de presión digital BMP085

Tabla 4.2: Características del controlador de vuelo

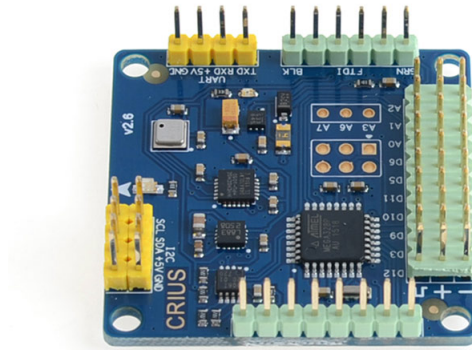


Figura 4.5: Controlador de vuelo estándar CRIUS MWC MultiWii SE V2.6

4.1.2.4 PROPULSIÓN DEL CUADRICÓPTERO

La velocidad angular de los rotores determinará el empuje, orientación y estabilidad del cuadricóptero.

Se utilizan motores sin escobillas (ver Figura 4.6), ya que son potentes y de alto rendimiento. Es importante seleccionar motores que puedan proporcionar un empuje lo suficientemente grande para levantar el peso neto de la estructura. Las características principales de los motores seleccionados se muestran en la Tabla 4.3

RPM/V	2280KV
Batería	2-3s Li-Po
Empuje máximo	460g
Masa	18g
Altura	26.7mm
Anchura	23mm

Tabla 4.3: Características de los motores



Figura 4.6: Motores MT1806 2280KV

Los ESCs (Electronic Speed Controllers, por sus siglas en inglés) son un componente esencial de los multirrotores ya que ofrecen la potencia y frecuencia necesarias para el motor en un circuito pequeño y ligero (ver Figura 4.7).

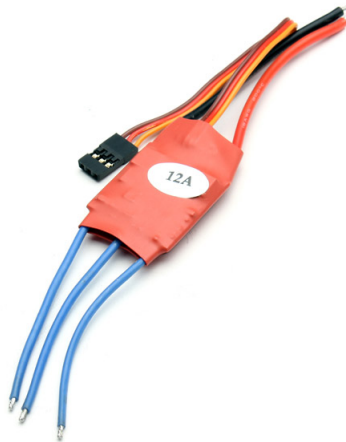


Figura 4.7: Controlador de velocidad para motor

La hélice es el dispositivo mecánico que permite al cuadricóptero desplazarse a través del aire utilizando el principio de Bernoulli. La hélice se compone de dos palas que poseen una forma curva, para que al girar sea posible desplazar el aire, debido a la diferencia de presión entre las caras de ambas palas.

Cabe recordar que en el cuadricóptero, dos motores giran en sentido horario; los otros dos giran en sentido contrario. Por esto mismo, el ángulo de torsión de las

palas está invertido entre las hélices que giran en sentido horario y antihorario (ver Figura 4.8).



Figura 4.8: Propulsores utilizados

4.1.2.5 GRABACIÓN DE DATOS

El registrador de datos se compone de un microcontrolador Arduino Pro Mini (ver Figura 4.9, izq.) y un modulo lector de tarjetas SD (ver Figura 4.9, der.). El microcontrolador se encarga de enviar comandos al Multiwii para obtener datos acerca de la orientación del cuadricóptero. Cada vez que se obtienen datos, estos se almacenan en un array y son grabados en la memoria SD.

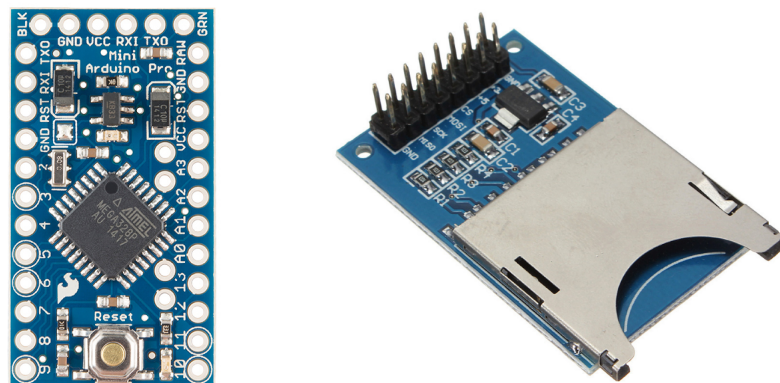


Figura 4.9: Microcontrolador Arduino Pro Mini y Lector SD

4.1.3 SOFTWARE

El controlador de vuelo MultiWii fue programado con el archivo más reciente (al 20 de noviembre de 2016) en el sitio wiki de MultiWii. Es preciso realizar ciertas modificaciones al código, tal como especificar el tipo de UAV a utilizar (cuadricóptero en este caso), cuántos motores serán utilizados, la configuración del cuadricóptero (QuadX/Quad+), etc.

Una vez se sube el código al controlador de vuelo, se abre la interfaz gráfica de usuario (GUI) de MultiWii: MultiWiiConf (ver Figura 4.10). En esta GUI es posible modificar las ganancias del controlador al valor deseado.

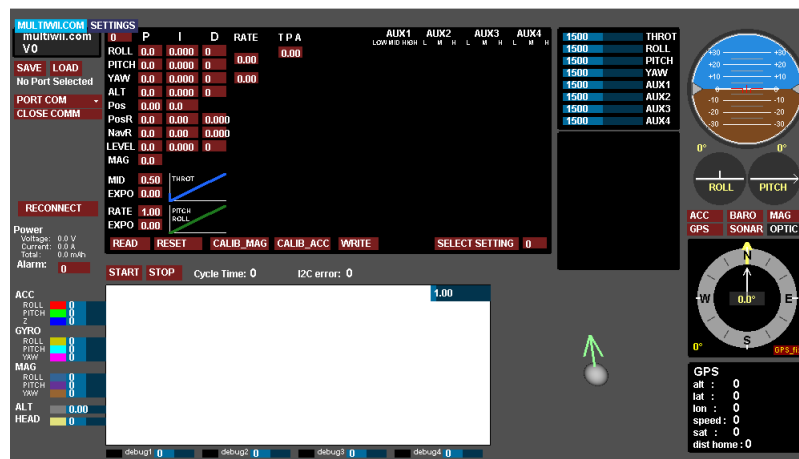


Figura 4.10: MultiWiiConf

4.1.4 FUNCIONAMIENTO

1. Se conecta la batería y se recomienda sea posicionada como en la figura 4.1.
2. Se procede a encender el control remoto o transmisor de la señal.
3. Se activan los motores. Esto se logra apuntando la palanca de mando izquierda completamente hacia abajo y luego hacia la derecha por aproximadamente tres segundos.

4. Para controlar la elevación del cuadricóptero se comienza a subir o bajar lentamente la palanca de mando derecha.
5. Una vez que el cuadricóptero se encuentre a la altura deseada puede proceder a modificar los ángulos de ϕ (arriba-abajo) y θ (izquierda-derecha) mediante la palanca de mando izquierda.
6. También es posible controlar el ángulo ψ (izquierda-derecha) mediante la palanca de mando derecha.
7. Para evitar daños en la batería es recomendable detener el vuelo cuando el nivel sea bajo. También es recomendable aterrizar el vehículo de manera suave, ya que los golpes bruscos también podrían dañar la batería.
8. Aterrizado el vehículo, es pertinente desconectar la alimentación antes de apagar el control remoto, ya que puede haber ruido en el exterior que descontrola por completo el vehículo.

4.2 IDENTIFICACIÓN DE PARÁMETROS

El modelo CAD (ver Figura 4.11) fue impreso en una impresora 3D de la marca *MakerBot*. Una vez impresa la estructura del cuadricóptero y las piezas puestas en el lugar correspondiente (ver Figura 4.12), la masa total m del cuadricóptero fue medida con una báscula.

Se obtuvieron L y los elementos de la diagonal en la matriz de inercias (I_{xx} , I_{yy} , I_{zz}) gracias al diseño asistido por computadora [*Computer Assisted Design (CAD)*] de la estructura del cuadricóptero en *SolidWorks*.

Las constantes k y b , conocidas como los factores de empuje y arrastre respectivamente, se obtuvieron por medio de dos experimentos diferentes (ver Figuras 4.13 y 4.14). Estos factores están relacionados a la magnitud del par en cada ángulo

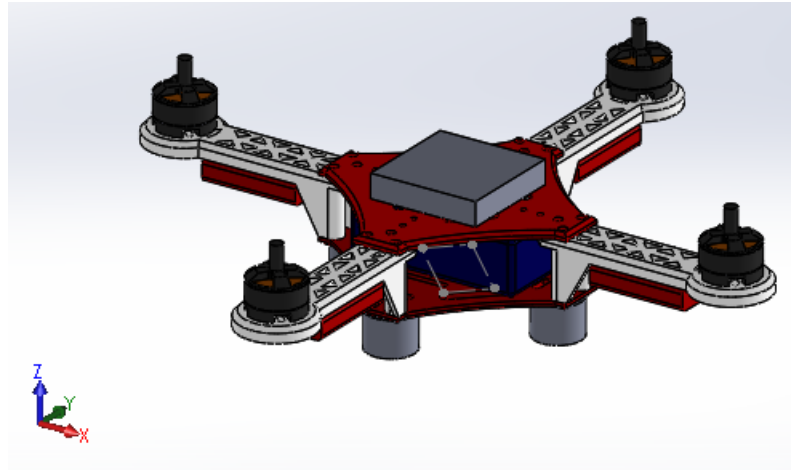


Figura 4.11: Modelo CAD del cuadricóptero



Figura 4.12: Impresión 3D de la estructura del cuadricóptero y sus componentes en el lugar correspondiente

de Euler (Ecuaciones 4.1 y 4.2). Las constantes l_1 y l_2 son la longitud de los dos segmentos de la escuadra; en este experimentos ambas longitudes son iguales.

- Experimento 1 (Pares ϕ y θ)

$$\begin{aligned}\tau_{\phi,\theta} &= T_B l_1 = mgl_2 \\ &= k\omega^2 l_1 = mgl_2\end{aligned}\tag{4.1}$$

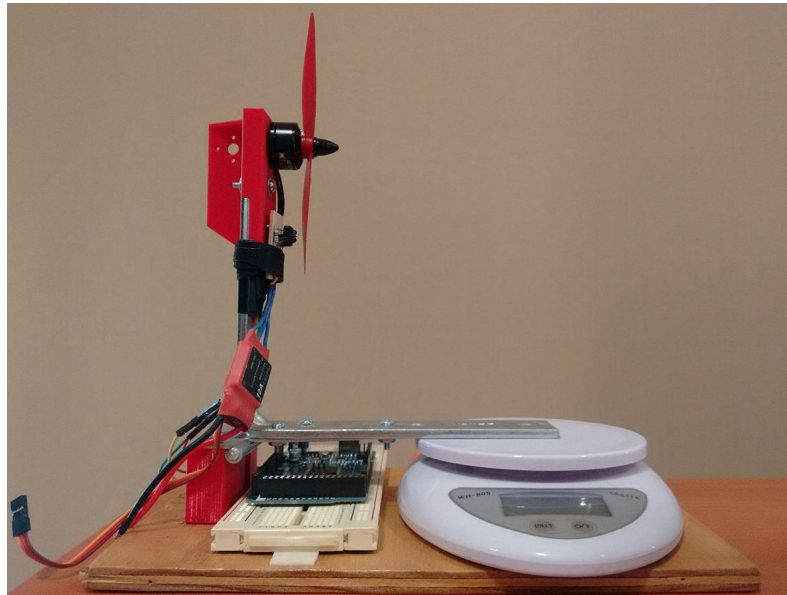


Figura 4.13: Experimento 1. Par generado en los ángulos de ϕ y θ

ω (RPM)	m(g)
7080	47
10560	106
14280	196
17520	297

Tabla 4.4: Datos del experimento 1

- Experimento 2 (Par ψ)

$$\tau_\psi = b\omega^2 = mgl_2 \quad (4.2)$$

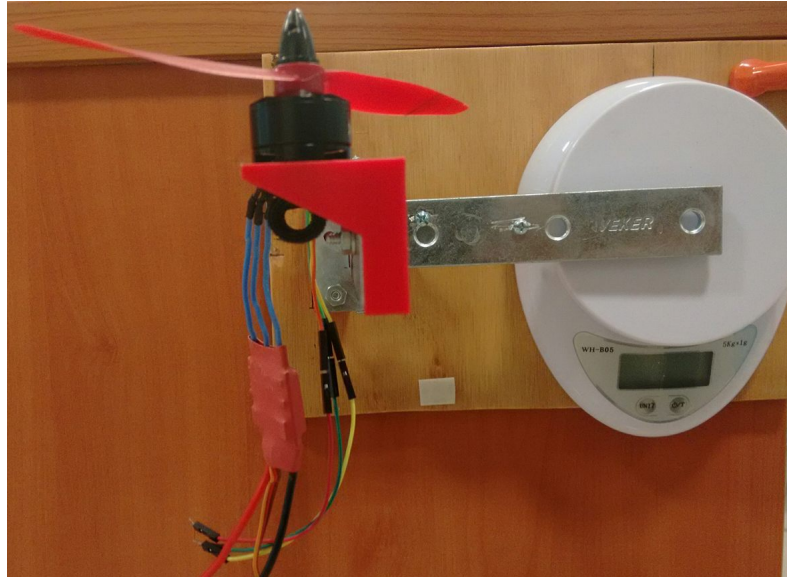


Figura 4.14: Experimento 2. Par generado en el ángulo ψ

ω (RPM)	m(g)
6660	10
9720	14
11850	18
15000	26
18210	30

Tabla 4.5: Datos del experimento 2

Finalmente, los parámetros requeridos en el modelo matemático para el prototipo construido se muestran en la tabla 4.6.

Parametro	Valor	Unidad
m	0.423	kg
g	9.81	m/s^2
L	0.117	m
k	8.5×10^{-7}	$kg \cdot m$
b	1.4675×10^{-8}	$kg \cdot m^2$
k_d	0.25	kg/s
I_{xx}	$1.056\ 41 \times 10^{-3}$	$kg \cdot m^2$
I_{yy}	$1.089\ 68 \times 10^{-3}$	$kg \cdot m^2$
I_{zz}	$1.990\ 25 \times 10^{-3}$	$kg \cdot m^2$

Tabla 4.6: Parámetros del prototipo construido

4.3 OBTENIENDO GANANCIAS DEL CONTROLADOR PARA EL PROTOTIPO CONSTRUIDO

Usando los parámetros del prototipo construido (Tabla 4.6), la simulación se ejecutó de nuevo para obtener las ganancias del controlador en el sistema real.

La evolución de las ganancias del controlador por medio del método propuesto y la mejora en los índices de desempeño se muestran en las Tablas 4.7, 4.8 y 4.9:

K_p	K_i	K_d	OS	SSE	ST
0.1	0.001	1.	17.468587	0.0793510	10.
0.8458149	0.0051675	2.6670136	6.3570395	0.0748264	10.
1.5916298	0.0093351	4.3340271	4.7947391	0.0040364	3.924
2.0233305	0.0116800	6.4688839	3.4598191	0.0000887	4.097
2.4290155	0.0124791	9.518456	2.4502921	0.0002529	3.509
2.428949	0.0124783	11.185476	2.119899	0.0004045	3.328
2.4288755	0.0125254	12.85246	1.8654155	0.0003442	3.035
2.4288046	0.0141250	14.519573	1.6636515	0.0002702	2.691
2.4305706	0.0171652	16.186551	1.5004191	0.0001982	2.317
2.4305038	0.0203210	17.417741	1.3983226	0.0000708	2.016
2.4304307	0.0229201	18.142167	1.3442095	0.0000574	1.83
2.4303598	0.0253845	18.580575	1.3133763	0.0001387	1.716
2.4302889	0.0277230	18.745506	1.3019963	0.0002050	1.668
2.430217	0.0300161	18.750808	1.3015577	0.0002484	1.662
2.430145	0.0323075	18.750448	1.3015638	0.0002613	1.66
2.430073	0.0345989	18.750088	1.3015774	0.0002648	1.66
2.430001	0.0368904	18.749728	1.3015897	0.0002658	1.66
2.429929	0.0391819	18.749368	1.3016028	0.0002660	1.66
2.429857	0.0414735	18.749008	1.3016125	0.0002660	1.66
2.429785	0.0437651	18.748648	1.3016185	0.0002660	1.66
2.429713	0.0460567	18.748288	1.3016235	0.0002660	1.66
2.4296411	0.0483484	18.747928	1.3016274	0.0002660	1.66
2.4295691	0.0506401	18.747568	1.3016329	0.0002660	1.66
2.4294971	0.0529317	18.747208	1.3016482	0.0002661	1.661
2.4294251	0.0552235	18.746848	1.3016848	0.0002662	1.661

Tabla 4.7: Evolución de las ganancias e índices de desempeño para el ángulo ϕ

K_p	K_i	K_d	OS	SSE	ST
0.1	0.001	1.	25.42214	0.1307779	10.
0.8458149	0.0051675	2.6670136	6.6300895	0.1234279	10.
1.5916298	0.0093351	4.3340271	5.2191167	0.0066575	3.873
1.9218939	0.0052708	7.9094839	3.1083714	0.0013441	4.702
2.1525994	0.0052700	11.081866	2.2479827	0.0005538	3.809
2.1525281	0.0052691	12.749223	1.9612138	0.0004297	3.445
2.1524575	0.0052682	14.416279	1.7380159	0.0003314	3.051
2.1523877	0.0068009	16.083171	1.5602401	0.0002506	2.639
2.2078897	0.0101008	17.750134	1.4137703	0.0000108	2.159
2.2078173	0.0127671	18.704812	1.3416429	0.0000382	1.914
2.2077462	0.0152205	19.288649	1.3008236	0.0001093	1.756
2.2076741	0.0175090	19.570828	1.281756	0.0001760	1.675
2.2076002	0.0197183	19.603079	1.2792444	0.0002338	1.658
2.207526	0.0219178	19.602708	1.2790977	0.0002567	1.655
2.2074517	0.0241167	19.602337	1.2790611	0.0002639	1.654
2.2073775	0.0263155	19.601965	1.279064	0.0002661	1.653
2.2073033	0.0285143	19.601594	1.2790777	0.0002668	1.653
2.207229	0.0307131	19.601223	1.279091	0.0002669	1.654
2.2071548	0.0329120	19.600852	1.2791008	0.0002670	1.654
2.2070806	0.0351110	19.600481	1.2791069	0.0002669	1.654
2.2070063	0.0373099	19.60011	1.279112	0.0002669	1.654
2.2069321	0.0395089	19.599738	1.279116	0.0002669	1.654
2.2068579	0.0417079	19.599367	1.2791218	0.0002669	1.654
2.2067836	0.0439069	19.598996	1.2791369	0.0002670	1.654
2.2067094	0.0461060	19.598625	1.2791727	0.0002671	1.654

Tabla 4.8: Evolución de las ganancias e índices de desempeño para el ángulo θ

K_p	K_i	K_d	OS	SSE	ST
0.1	0.001	1.	19.683504	0.0900751	10.
0.8458149	0.0051675	2.6670136	6.4627353	0.0848820	10.
1.5916298	0.0093351	4.3340271	4.9045256	0.0046554	3.908
1.8023958	0.0107665	5.4779669	4.0811564	0.0000207	4.427
2.5044853	0.0143640	6.055674	3.6589609	0.0012961	3.289
3.0079572	0.0160805	8.3048656	2.7753419	0.0000718	3.072
3.1283598	0.0160797	10.703382	2.2134853	0.0002592	2.9
3.1282878	0.0177861	12.370654	1.9384924	0.0001822	2.693
3.1282186	0.0204620	14.037845	1.7229181	0.0001493	2.439
3.1281542	0.0242397	15.537312	1.5649637	0.0000816	2.17
3.1280827	0.0276496	16.509886	1.4763887	0.0000406	1.972
3.1280131	0.0306104	17.162309	1.4222189	0.0001182	1.836
3.1279443	0.0334068	17.586633	1.3889119	0.0001709	1.745
3.1278747	0.0360577	17.818186	1.371298	0.0002116	1.694
3.1278041	0.0386344	17.907642	1.3645503	0.0002413	1.673
3.1277348	0.0411835	17.929965	1.3628399	0.0002570	1.666
3.1276653	0.0437255	17.929618	1.3628348	0.0002633	1.665
3.1275958	0.0462675	17.92927	1.3628494	0.0002644	1.665
3.1275264	0.0488095	17.928923	1.3628646	0.0002645	1.665
3.1274569	0.0513516	17.928576	1.3628771	0.0002645	1.665
3.1273874	0.0538938	17.928228	1.3628883	0.0002645	1.665
3.127318	0.0564360	17.927881	1.362898	0.0002645	1.665
3.1272485	0.0589782	17.927534	1.3629079	0.0002645	1.665
3.1271791	0.0615205	17.927186	1.3629244	0.0002646	1.665
3.1271096	0.0640628	17.926839	1.3629579	0.0002646	1.665

Tabla 4.9: Evolución de las ganancias e índices de desempeño para el ángulo ψ

4.4 RESULTADOS

Usando las últimas ganancias proporcionadas por los resultados de la simulación y aplicando un impulso vía control remoto en los ángulos de ϕ y θ (ver Figura 4.15, izq.), se llevaron a cabo las pruebas de vuelo.

La orientación del cuadricóptero se obtuvo mediante la comunicación serial entre la unidad de medición inercial, un microcontrolador y un lector de tarjetas SD (ver Figura 4.15, der.).



Figura 4.15: Control y grabación de orientación en el cuadricóptero

Una vez grabados los datos dentro de la tarjeta SD durante las pruebas de vuelo, se muestran los resultados en las Figuras 4.16 y 4.17. En la Figura 4.18 se muestran algunas capturas de una de las pruebas de vuelo.

El impulso en el ángulo ψ no fue aplicado porque el controlador no está programado para reajustarlo a una posición inicial.

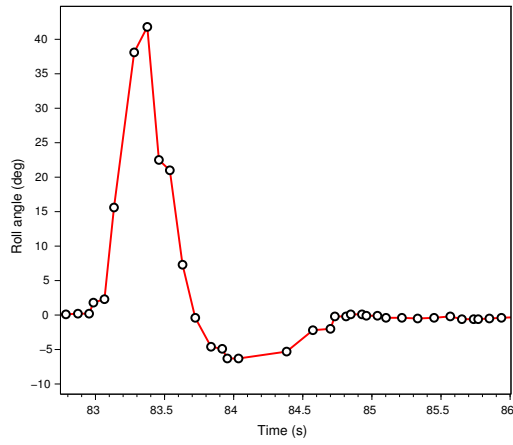


Figura 4.16: Medición del ángulo ϕ

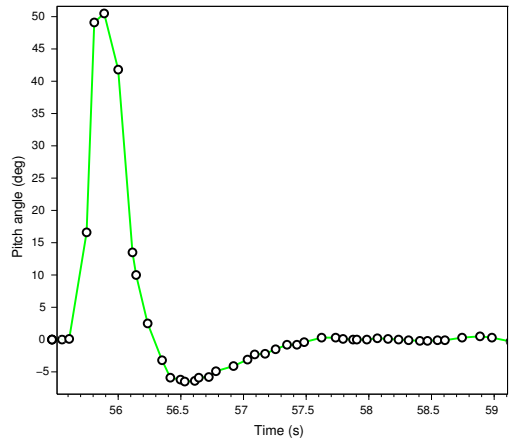


Figura 4.17: Medición del ángulo θ



Figura 4.18: Prueba de vuelo

CAPÍTULO 5

CONCLUSIONES

Los resultados de la simulación muestran que el método propuesto para sintonizar las ganancias del controlador convencional logra una mejor respuesta a un impulso en la velocidad angular en comparación con los métodos de Ziegler-Nichols y sintonización manual. Esto se afirma, a partir de los índices de desempeño del controlador en cada método y visualmente por medio de las animaciones, para corroborar efectivamente los resultados.

Asimismo, los resultados experimentales muestran que al evolucionar las ganancias por medio del esquema propuesto y programarlas dentro del controlador del cuadricóptero, se obtiene un comportamiento estable. No obstante, las pruebas se hicieron en un ambiente controlado y el modelo del UAV puede llegar a ser mucho más preciso que el utilizado. Además, la programación por default de la tarjeta MultiWii nos limitó a realizar algunas pruebas adicionales de la manera deseada.

El trabajo futuro se enfocará en obtener un modelo más aproximado (basado en Inteligencia Artificial), en programar la tarjeta MultiWii para que se ajuste a nuestras necesidades y en mejorar la respuesta del sistema en tiempo real, obteniendo datos sobre la orientación del cuadricóptero en línea para evaluar la respuesta del sistema físico.

BIBLIOGRAFÍA

- [1] ABBASI, E. y M. J. MAHJOOB, «Controlling of Quadrotor UAV Using a Fuzzy System for Tuning the PID Gains in Hovering Mode», , págs. 1–6, 2012.
- [2] ALAIMO, A., V. ARTALE, C. L. R. MILAZZO y A. RICCIARDELLO, «PID controller applied to hexacopter flight», *Journal of Intelligent and Robotic Systems: Theory and Applications*, **73**(1-4), págs. 261–270, 2014.
- [3] ALONSO-CARREÓN, A. A., M. PLATAS y L. M. TORRES-TREVIÑO, «Artificial Intelligence and Soft Computing», **9119**(1), págs. 443–451, 2015, URL <http://link.springer.com/10.1007/978-3-319-19324-3>.
- [4] ALVARADO-YAÑEZ, L. A., L. M. TORRES-TREVIÑO y A. RODRÍGUEZ, «An Embedded Fuzzy Agent for Online Tuning of a PID Controller for Position Control of a DC Motor», , págs. 225–232, 2013.
- [5] ALVARENGA, J., N. I. VITZILAIOS, K. P. VALAVANIS y M. J. RUTHERFORD, «Survey of Unmanned Helicopter Model-Based Navigation and Control Techniques», *Journal of Intelligent and Robotic Systems: Theory and Applications*, **80**(1), págs. 87–138, 2014.
- [6] ASSOCIATION FOR UNMANNED VEHICLE SYSTEMS INTERNATIONAL, «The Benefits of Unmanned Aircraft Systems : Saving Time, Saving Money, Saving Lives», , 2015, URL <https://epic.org/events/UAS-Uses-Saving-Time-Saving-Money-Saving-Lives.pdf>.

-
- [7] BEVILACQUA, V., E. GRASSO, G. MASTRONARDI y L. RICCARDI, «A soft computing approach to the intelligent control», *2006 IEEE International Conference on Industrial Informatics*, págs. 1312–1317, 2006.
- [8] CAMBONE, S. A., K. J. KRIEG, P. PACE y L. WELLS, «Unmanned aircraft systems roadmap 2005–2030. Technical report», *US Department of Defense*, 2005.
- [9] CHAO, H., Y. CAO y Y. CHEN, «Autopilots for small unmanned aerial vehicles: A survey», *International Journal of Control, Automation and Systems*, **8**(1), págs. 36–44, 2010.
- [10] CHOVANCOVÁ, A., T. FICO, L. CHOVANEC y P. HUBINSKÝ, «Mathematical modelling and parameter identification of quadrotor (a survey)», *Procedia Engineering*, **96**, págs. 172–181, 2014.
- [11] DULO, D., «Unmanned Aircraft Classifications», *The SciTech Lawyer*, **11**(4), pág. 16, 2015, URL <http://www.nasa.gov/centers/armstrong/news/FactSheets/FS-075-DFRC.html{#}.VgSEAJ55lwA.mendeley>.
- [12] ESWARAN, P., G. MAHENDAR, P. MUKUNDA y K. ZEESHAN, «Stabilization of UAV Quadcopter», , pág. 828, 2016.
- [13] GIBIANSKY, A., «Quadcopter Dynamics , Simulation , and Control Introduction Quadcopter Dynamics», , págs. 1–18, 2012, URL <https://github.com/gibiansky/experiments/blob/master/quadcopter/matlab/simulate.m>.
- [14] GOWRISANKAR, M. y A. N. KUMAR, «Soft Computing based Controllers for Electric Drives - A Comparative Approach», **10**(1), págs. 184–192, 2015.
- [15] JAIMES, A., S. KOTA y J. GOMEZ, «An approach to surveillance an area using swarm of fixed wing and quad-rotor unmanned aerial vehicles UAV(s)», *2008 IEEE International Conference on System of Systems Engineering, SoSE 2008*, 2008.

- [16] KABBABE, K., «Development of Procedures for Flight Testing Uavs Using the Ardupilot System», ... *school of mechanical, The University of ...*, 2011, URL <http://uavs.mace.manchester.ac.uk/files/2513/7971/0471/MScKabbabe2011.pdf>.
- [17] KETATA, R., D. DE GEEST y A. TITLI, «Fuzzy controller: design, evaluation, parallel and hierarchial combination with a PID controller», *Fuzzy Sets and Systems*, **71**(1), págs. 113–129, 1995.
- [18] KORCHENKO, A. G. y O. S. ILLYASH, «The generalized classification of Unmanned Air Vehicles», *2013 IEEE 2nd International Conference on Actual Problems of Unmanned Air Vehicles Developments, APUAVD 2013 - Proceedings*, págs. 28–34, 2013.
- [19] LEONG, B. T. M., S. M. LOW y M. P. L. OOI, «Low-cost microcontroller-based hover control design of a quadcopter», *Procedia Engineering*, **41**(IRIS), págs. 458–464, 2012.
- [20] LUUKKONEN, T., «Modelling and Control of Quadcopter», *Journal of the American Society for Mass Spectrometry*, **22**(7), págs. 1134–45, 2011, URL <http://www.ncbi.nlm.nih.gov/pubmed/21953095>.
- [21] MOHAMMADI, M. y A. M. SHAHRI, «Adaptive nonlinear stabilization control for a Quadrotor UAV: Theory, simulation and experimentation», *Journal of Intelligent and Robotic Systems: Theory and Applications*, **72**(1), págs. 105–122, 2013.
- [22] NAHAPETIAN, N., M. R. J. MOTLAGH y M. ANALOUI, «PID Gain Tuning using Genetic Algorithms and Fuzzy Logic For Robot Manipulator Control», *International Conference on Advanced Computer Control : Icac 2009 - Proceedings*, págs. 346–350, 2009.
- [23] PAPPALARDO, J., «Unmanned aircraft road- map reflects changing priorities», *National Defense*, **89**, 2005.

-
- [24] SCHMID, C., «The components of a fuzzy system», , 2005, URL <http://virtual.cvut.cz/course/syscontrol/node130.html>.
- [25] SKY ENGINEERING LABORATORY, «Making Animation with Scilab», , 2011, URL <http://www.sky-engin.jp/en/ScilabAnimation/index.html>.
- [26] SULLIVAN, J. M., «Evolution or revolution? The rise of UAVs», *IEEE Technology and Society Magazine*, (October), 2006.
- [27] VEMPATI, A. S., V. CHOUDHARY y L. BEHERA, «Quadrotor : Design , Control and Vision Based Localization», *Third International Conference on Advances in Control and Optimization of Dynamical Systems*, (2012), págs. 1105–1106, 2014.
- [28] WANG, Y., Y. CHENXIE, J. TAN, C. WANG, Y. WANG y Y. ZHANG, «Fuzzy Radial Basis Function Neural Network PID Control System for a Quadrotor UAV Based on Particle Swarm Optimization», (August), págs. 2580–2585, 2015.

APÉNDICE A

CÓDIGO DE SIMULACIÓN DE UAV (SCILAB)

■ Código 1 (Código general)

```
1 exec('FuzzySystem.sce',-1) //Calls the fuzzy system functions
2 exec('quadcopter_animation.sce',-1) //Calls the animation function
3
4 function [result ,evaluation ,tables]=simulate(start_time ,end_time ,dt ,cycles)
5     xdel(winsid());
6     times = start_time:dt:end_time; //Simulation times, in seconds.
7
8     //Number of points in the simulation.
9     N = length(times);
10
11     //Data taken from Modelling and Control of Quadcopter (Luukkonen)
12     //m=0.5; //Mass of the UAV (kg)
13     //g=9.81; //Gravity, units: m/s^2
14     //k=3e-6; //Lift/Thrust Constant
15     //b=1e-7; //Drag Constant
16     //kd_f=0.1; //Drag Force: 0.25
17     //L= 0.25; //Distance (center of the quad- any propeller)
18     //*****
19     //Inertia matrix elements
```

```

20 //*****
21 //I_xx=5e-3;
22 //I_yy=5e-3;
23 //I_zz=10e-3;
24
25 //Real Quadcopter
26 //m=0.423; //Mass of the UAV, units: kg
27 //g=9.81; //Gravity, units: m/s^2
28 //k=8.5e-7; //Lift/Thrust Constant
29 //b=1.4675e-8; //Drag Constant
30 //kd_f=0.25; //Drag Force
31 //L= 0.117; //Distance (center of the quad- any propeller)
32 //*****
33 //Inertia matrix elements
34 //*****
35 //I_xx=1.05641e-3;
36 //I_yy=1.08968e-3;
37 //I_zz=1.99025e-3;
38
39 //Manual Data
40 m=1.25; //Mass of the UAV, units: kg
41 g=9.81; //Gravity, units: m/s^2
42 k=2.92e-6; //Lift/Thrust Constant
43 b=1.12e-7; //Drag Constant
44 kd_f=0.25; //Drag Force
45 L= 0.2; //Distance (center of the quad- any propeller)
46 //*****
47 //Inertia matrix elements
48 //*****
49 I_xx=2.353e-3;
50 I_yy=2.353e-3;
51 I_zz=4.706e-3;
52
53 //Inertia matrix
54 I = diag([I_xx , I_yy , I_zz]); //Symmetric quadcopter
55

```

```
56     params=struct("mass",m,"gravity",g,"thrust",k,"drag",b,...
57
58     "dragforce",kd_f,"length",L,"inertia",I,"diff",dt);
59
60     //Desired Height
61     zd=2;
62
63     //Height control gains (Constants)
64     Kp_alt=3; //def:3
65     Kd_alt=4; //def:4
66
67     //Unit Gains
68     Kp=[0.1,0.1,0.1]';
69     Ki=[0.001,0.001,0.001]';
70     Kd=[1,1,1]';
71
72     //Paper Gains (Trial and error hand tuning)
73     //Kp=[3,3,3]';
74     //Ki=[5.5,5.5,5.5]';
75     //Kd=[4,4,4]';
76
77     //Ziegler-Nichols Gains
78     //Kcr: 2,1.3,1.2
79     //Pcr: 4.1,5,5.5
80     //Kp=[0.6*2,0.6*1.3,0.6*1.2]';
81     //Ki=[0.5*4.1,0.5*5,0.5*5.5]';
82     //Kd=[0.125*4.1,0.125*5,0.125*5.5]';
83
84     //Load rules (rules_kp, rules_ki, rules_kd)
85     load('/home/victorm/Desktop/Thesis/QuadModel/Variables/rules_kp.sod');
86     load('/home/victorm/Desktop/Thesis/QuadModel/Variables/rules_ki.sod');
87     load('/home/victorm/Desktop/Thesis/QuadModel/Variables/rules_kd.sod');
88
89     x_out=zeros(3,N,cycles);
90     xdot_out = zeros(3, N,cycles);
91     theta_out = zeros(3, N,cycles);
```

```

92     thetadot_out = zeros(3, N, cycles);
93     omega_out=zeros(3,N, cycles);
94     omegadot_out=zeros(3,N, cycles);
95     input_out=zeros(4,N, cycles);
96     tables=zeros(cycles ,6 ,3);
97
98     for cycle=1:cycles
99         //Initial simulation state.
100        x=zeros(3,1);
101        //x = [0; 0; 1];
102        xdot = zeros(3,1);
103        theta = zeros(3,1);
104        //theta=[- %pi/12;- %pi/12;- %pi/12]; // Initial deviation
105        thetadot = zeros(3,1);
106        omega=zeros(3,1);
107        omegadot=zeros(3,1);
108        os=zeros(3,1);
109        t_rise=zeros(3,1);
110        t_settle=zeros(3,1);
111        sse=zeros(3,1);
112
113        //Simulate some disturbance in the angular velocity.
114        deviation = -15; //units: deg/sec
115
116        //thetadot = deg2rad(2 * deviation * rand(3, 1) - deviation);
117        thetadot = deg2rad(2 * deviation * [0;0;0] - deviation);
118
119        index=1; //n will take values from 1 to N
120
121        //Step through the simulation, updating the state.
122        for t = times
123
124            [i, params]= pid_controller(Kp,Ki,Kd,Kp_alt ,Kd_alt ,params ,...
125            thetadot ,theta); //Take input from our controller.
126            omega = thetadot2omega(thetadot , theta);
127

```

```

128      //Compute linear and angular accelerations.
129      a = acceleration(i, theta, xdot, params.mass, params.gravity, ...
130      params.thrust, params.dragforce);
131
132      omegadot = angular_acceleration(i, omega, params.inertia, ...
133      params.length, params.drag, params.thrust);
134
135      omega = omega + params.diff * omegadot;
136      [thetadot, t_matrix] = omega2thetadot(omega, theta);
137      theta = theta + params.diff * thetadot;
138      xdot = xdot + params.diff * a;
139      x = x + params.diff * xdot;
140
141      x_out(:, index, cycle) = x;
142      xdot_out(:, index, cycle) = xdot;
143      theta_out(:, index, cycle) = theta;
144      thetadot_out(:, index, cycle) = thetadot;
145      omega_out(:, index, cycle) = omega;
146      omegadot_out(:, index, cycle) = omegadot;
147      input_out(:, index, cycle) = i;
148
149      index=index+1;
150  end
151
152  result = struct('x', x_out, 'theta', theta_out, 'vel', xdot_out, ...
153  'angvel', thetadot_out, 'omega', omega_out, 'omegadot', omegadot_out, ...
154  't', times, 'dt', dt, 'input', input_out);
155
156  //In case of just evaluating the gains for a random deviation...
157  //comment the code since ref=0
158  //evaluation=1;
159
160  ref=0;
161
162  //Analyze the 3 signals: Roll-Pitch-Yaw angular velocities
163  for jj=1:3

```

```

164         [os(jj), t_rise(jj), t_settle(jj), sse(jj)]...
165         =signalAnalyzer(result.theta(jj,:), cycle), result.t, ref);
166     end
167
168     evaluation=struct('os',os,'sse',sse,'rt',t_rise,'st',t_settle);
169
170     aux=[Kp Ki Kd evaluation.os evaluation.sse evaluation.st];
171
172     for ii=1:3
173         tables(cycle,:,ii)=aux(ii,:);
174         norm_os(ii)=normalize(evaluation.os(ii),0,5);
175         norm_sse(ii)=normalize(evaluation.sse(ii),0,1e-2);
176         norm_st(ii)=normalize(evaluation.st(ii),0,5);
177         //norm_rt(ii)=normalize(evaluation.rt(ii),0,100);
178     end
179
180     norm_data=[norm_sse'; norm_os'; norm_st'];
181
182     for kk=1:3
183         Kp_out(kk)=FuzzySysT1(norm_data(:,kk), rules_kp);
184         Ki_out(kk)=FuzzySysT1(norm_data(:,kk), rules_ki);
185         Kd_out(kk)=FuzzySysT1(norm_data(:,kk), rules_kd);
186
187         //Denormalizing output fuzzy data
188         DKp(kk) = denormalize(Kp_out(kk), -1, 1);
189         DKi(kk) = denormalize(Ki_out(kk), -1, 1);
190         DKd(kk) = denormalize(Kd_out(kk), -1, 1);
191
192         //Proportional to the limits on MultiWii
193         Ft1=1;
194         Ft2=0.0125;
195         Ft3=5;
196
197         Kp(kk) = Kp(kk) + (Ft1*DKp(kk));
198         Ki(kk) = Ki(kk) + (Ft2*DKi(kk));
199         Kd(kk) = Kd(kk) + (Ft3*DKd(kk));

```



```

200
201         //Multiwii limits
202         if Kp(kk)<0.1 Kp(kk)=0.1; end
203         if Ki(kk)<0.001 Ki(kk)=0.001; end
204         if Kd(kk)<1 Kd(kk)=1; end
205
206         if Kp(kk)>20 Kp(kk)=20; end
207         if Ki(kk)>0.25 Ki(kk)=0.25; end
208         if Kd(kk)>100 Kd(kk)=100; end
209     end
210 end
211
212
213 endfunction
214
215 function plotQuadData(result ,cycle)
216     scf(1)
217     subplot(121)
218     plot(result.t,result.x(1, :, cycle), 'ro-', result.t, result.x(2, :, cycle), ...
219         'g+-', result.t, result.x(3, :, cycle), 'b*--', 'markersize', 2);
220     p = get("hdl");
221     p.children.thickness = 1.5;
222
223     l=legend(["$x$"; "$y$"; "$z$"]);
224     l.font_size = 3;
225     l.font_style = 1;
226     l.visible='on';
227     xgrid(1)
228
229     subplot(122)
230     plot2d(result.t,[result.vel(1, :, cycle)' result.vel(2, :, cycle)'...
231         result.vel(3, :, cycle)'], [5 3 2], leg="L1@L2@L3");
232     p = get("hdl");
233     p.children.thickness = 2;
234     l=legend(["$\dot{x}$"; "$\dot{y}$"; "$\dot{z}$"]);
235     l.font_size = 3;

```

```
236     l.font_style = 1;
237     l.visible='on';
238     xgrid(1)
239
240     scf(2)
241     subplot(121)
242     plot2d(result.t,[result.theta(1, :, cycle)' result.theta(2, :, cycle)'...
243     result.theta(3, :, cycle)'],[5 3 2],leg="L1@L2@L3");
244     p = get("hdl");
245     p.children.thickness = 2;
246     l=legend([" $\phi$ "; " $\theta$ "; " $\psi$ "]);
247     l.font_size = 3;
248     l.font_style = 1;
249     l.visible='on';
250     xgrid(1)
251
252     subplot(122)
253     plot2d(result.t,[result.angvel(1, :, cycle)' result.angvel(2, :, cycle)'...
254     result.angvel(3, :, cycle)'],[5 3 2],leg="L1@L2@L3");
255     p = get("hdl");
256     p.children.thickness = 2;
257     l=legend([" $\dot{\phi}$ "; " $\dot{\theta}$ "; " $\dot{\psi}$ "]);
258     l.font_size = 3;
259     l.font_style = 1;
260     l.visible='on';
261     xgrid(1)
262
263     scf(3)
264     subplot(121)
265     plot2d(result.t,[result.omega(1, :, cycle)' result.omega(2, :, cycle)'...
266     result.omega(3, :, cycle)'],[5 3 2],leg="L1@L2@L3");
267     p = get("hdl");
268     p.children.thickness = 2;
269     l=legend([" $\omega_x$ "; " $\omega_y$ "; " $\omega_z$ "]);
270     l.font_size = 3;
271     l.font_style = 1;
```

```

272     l.visible='on';
273     xgrid(1)
274
275     subplot(122)
276     plot2d(result.t,[result.omegadot(1, :, cycle)' result.omegadot(2, :, cycle)'...
277     result.omegadot(3, :, cycle)'],[5 3 2],leg="L1@L2@L3");
278     p = get("hdl");
279     p.children.thickness = 2;
280     l=legend(["$\dot{\omega}_x$"; "$\dot{\omega}_y$"; "$\dot{\omega}_z$"]);
281     l.font_size = 3;
282     l.font_style = 1;
283     l.visible='on';
284     xgrid(1)
285
286     endfunction
287
288     function [radians]=deg2rad(degrees)
289         radians=degrees*(%pi/180);
290     endfunction
291
292     // Compute system inputs and updated state
293     //Note that input = [gamma_1,...,gamma_4]
294     function [u, params]= pid_controller(Kp,Ki,Kd,Kp_alt,Kd_alt,params,...
295     thetadot,theta,xdot,x,zd)
296     //*****
297     // Consider have 3 gains for each Euler Angle.
298     //*****
299
300     //Initialize the integral if necessary
301     if ~isfield(params, 'integral') then //create function isfield
302         params.integral = zeros(3,1);
303         params.integral2 = zeros(3,1);
304     end
305
306     //Prevent wind-up
307     if max(abs(params.integral2)) > 0.01 then //default=0.01

```

```

308     params.integral2(:)=0;
309 end
310
311     //Compute total thrust
312     total=(params.gravity+Kd.alt*(-xdot(3))+Kp.alt*(zd-x(3)))...
313     *((params.mass)/(params.thrust*cos(theta(1))*cos(theta(2))));
314
315     //Compute errors
316     e=(Kd.*thetadot)+(Kp.*params.integral)+(Ki.*params.integral2);
317     //e=(Kd.*thetadot)+(Kp.*theta)+(Ki.*params.integral2);
318
319     //e=(Kd.*thetadot)+(Kp.*theta)-(Ki.*params.integral2);
320
321     //Solve for the inputs, gamma_i
322     u= error2inputs(e,total,params);
323
324     //Update the state
325     params.integral=params.integral+params.diff.*thetadot;
326     params.integral2=params.integral2+params.diff.*params.integral;
327     //params.integral2=params.integral2+params.diff.*theta;
328
329 endfunction
330
331 function inputs=error2inputs(e,total_thrust,params)
332
333     e_phi=e(1);
334     e_theta=e(2);
335     e_psi=e(3);
336
337     Ixx=params.inertia(1,1);
338     Iyy=params.inertia(2,2);
339     Izz=params.inertia(3,3);
340
341     A=total_thrust/4;
342     B=(e_psi*Izz)/(4*params.drag);
343     C=(e_theta*Iyy)/(2*params.thrust*params.length);

```

```

344     D1=((2*params.drag*e_phi*Ixx)+(e_psi*Izz*params.thrust*params.length))...
345     /(4*params.drag*params.thrust*params.length);
346     D2=(-2*params.drag*e_phi*Ixx)+(e_psi*Izz*params.thrust*params.length))...
347     /(4*params.drag*params.thrust*params.length);
348
349     gamma_1=A-D1;
350     gamma_2=A+B-C;
351     gamma_3=A-D2;
352     gamma_4=A+B+C;
353
354     inputs=[gamma_1; gamma_2; gamma_3; gamma_4];
355
356 endfunction
357
358 function omega=thetadot2omega(thetadot,theta)
359
360     roll=theta(1);
361     pitch=theta(2);
362     t_matrix=[1, 0, -sin(pitch); 0, cos(roll), cos(pitch)*sin(roll);...
363             0, -sin(roll), cos(pitch)*cos(roll)];
364     omega=t_matrix*thetadot;
365
366 endfunction
367
368 function a=acceleration(inputs,angles,xdot,m,g,k,kd)
369
370     g_vector = [0; 0; -g];
371     R=rotation(angles); // create function rotation
372     T=R*thrust(inputs,k);
373     Fd=-kd.*xdot;
374     a= g_vector+((1/m).*(T+Fd));
375
376 endfunction
377
378 function R=rotation(rpy_angles)
379

```

```

380     phi_a=ropy_angles(1);
381     theta_a=ropy_angles(2);
382     psi_a=ropy_angles(3);
383
384     R=[(cos(psi_a)*cos(theta_a)),...
385         (cos(psi_a)*sin(theta_a)*sin(phi_a))-(cos(phi_a)*sin(psi_a)),...
386         (cos(psi_a)*sin(theta_a)*cos(phi_a))+(sin(psi_a)*sin(phi_a));...
387         sin(psi_a)*cos(theta_a),...
388         (sin(psi_a)*sin(theta_a)*sin(phi_a))+(cos(psi_a)*cos(phi_a)),...
389         (sin(psi_a)*sin(theta_a)*cos(phi_a))-(cos(psi_a)*sin(phi_a));...
390         -sin(theta_a), cos(theta_a)*sin(phi_a), cos(theta_a)*cos(phi_a)];
391
392     endfunction
393
394     function T=thrust(inputs,k)
395
396         //Inputs are values for omega_i^2
397         T=[0;0;k*sum(inputs)];
398
399     endfunction
400
401     function omega_dot=angular_acceleration(inputs,omega,I,L,b,k)
402         tau=torques(inputs,L,b,k);
403         omega_dot=inv(I)*(tau-cross(omega,I*omega));
404     endfunction
405
406     // Compute torques, given current inputs, length, drag coefficient and thrust
407     // coefficient
408     function tau=torques(inputs,L,b,k)
409
410         //Inputs are values for omega_i^2
411         tau=[L*k*(inputs(1)-inputs(3));...
412             L*k*(inputs(2)-inputs(4));...
413             b*(inputs(1)-inputs(2)+inputs(3)-inputs(4))];
414
415     endfunction

```

```

416
417 function [thetadot , t_matrix]=omega2thetadot(omega , theta)
418
419     roll=theta(1);
420     pitch=theta(2);
421     t_matrix=[1, 0, -sin(pitch); 0, cos(roll), cos(pitch)*sin(roll);...
422             0, -sin(roll), cos(pitch)*cos(roll)];
423     thetadot=inv(t_matrix)*omega;
424
425 endfunction
426
427 function [os , t_rise , t_settle , sse] = signalAnalyzer(y,t , ref)
428
429     last_out=length(y); //size of the output vector
430
431     //Overshoot
432     [p_max , v_max]=maxp(y);
433     [p_min , v_min]=minp(y);
434
435     //Recorrer valor minimo a cero
436     //max_value=v_max-v_min;
437     //sign_g=sign(y(last_out)-v_min);
438     //final_value=y(last_out)-v_min;
439     //os = ((v_max)/(ref-v_min))*100;
440     os = v_max*100;
441
442     //if sign_g== -1 then
443         //final_value=v_max-y(last_out);
444         //os = ((max_value-final_value)/(final_value+0.0000001))*100;
445     //else
446         //final_value=y(last_out)-v_min;
447         //os = ((max_value-final_value)/(final_value+0.0000001))*100;
448     //end
449
450     //Rise time
451     out=0;k=1;

```

```
452     while out==0
453         if y(k)>ref then //greater than reference (in this case 0)
454             out=1;t_rise=t(k);
455         end
456         k=k+1;
457         if k>last_out then
458             out=1;t_rise=t(k-1);
459         end
460     end
461
462     //Settle time (Original: 2% Tolerance)
463     toler=0.01; //1% Tolerance
464
465     for m=1:last_out
466         error_out = abs(ref - y(m));
467         if error_out > toler then
468             t_settle=t(m);
469         end
470     end
471
472     //Steady-state Error (Distance of the output from the reference)
473     sse=abs(ref-y(last_out));
474
475 endfunction
476
477 function [posmax, valmax]=maxp(v)
478
479     v_size=length(v);
480     posmax=1;
481     valmax=v(1);
482
483     for ii=1:v_size
484         if v(ii)>valmax
485             valmax=v(ii);
486             posmax=ii;
487         end
```



```

488         end
489
490     endfunction
491
492     function [posmin, valmin]=minp(v)
493
494         v_size=length(v);
495
496         posmin=1;
497         valmin=v(1);
498
499         for ii=1:v_size
500             if v(ii)<valmin
501                 valmin=v(ii);
502                 posmin=ii;
503             end
504         end
505
506     endfunction

```

■ Código 2 (Función del sistema difuso)

```

1     function y=FuzzySysT1(X,DB) // 'Inputs' could be obtained from DB variable.
2
3         //Remember that input X must be normalized
4         //Remember that output must be denormalized
5         [rules IO]=size(DB);
6         inputs=IO-1;
7
8         //General parameter of fuzzy systems type I
9         PARAM=[0 0.1666 0.3333;
10              0.1666 0.3333 0.5;
11              0.3333 0.5 0.6666;
12              0.5 0.6666 0.8333;
13              0.6666 0.8333 1];
14

```

```

15     //AC=[0,0.25,0.5,0.75,1];
16     FO=zeros(1, rules);
17
18     for r=1:rules
19         sumin=1;
20         for i=1:inputs
21             n=DB(r, i);
22             if n>0
23                 V=PARAM(DB(r, i), :);
24             end
25             mf=fuzzificate(X(i), n, V);
26             sumin=min(sumin, mf);
27         end
28         FO(r) = sumin;
29     end
30
31     sum1=0;sum2=0.00000001;
32
33     //Centroid defuzzifier
34     for dy=0:0.01:1
35         ss=0;
36         for r=1:rules
37             n=DB(r, IO);
38             if n>0
39                 // Extract the parameter of the 'n' fuzzy set
40                 // involved in rule r
41                 V=PARAM(DB(r, IO), :);
42             end
43             mf=fuzzificate(dy, n, V);
44             ss=max(ss, min(mf, FO(r)));
45         end
46         sum1=sum1+ss*dy;
47         sum2=sum2+ss;
48     end
49
50     y=sum1/sum2;

```

```
51
52 endfunction
53
54 // Obtains membresy value for a linguistic value
55 //x: Normalized input
56 //n: Linguistic value to be evaluated
57 //V[]: Parameters for triangles and trap.
58 function mf=fuzzificate(x,n,V)
59
60     a=V(1);
61     b=V(2);
62     c=V(3);
63
64     if n==1
65         mf=trapezoidmf(x,a-0.1666,a,b,c);
66     end
67
68     if n==2
69         mf=trianglemf(x,a,b,c);
70     end
71
72     if n==3
73         mf=trianglemf(x,a,b,c);
74     end
75
76     if n==4
77         mf=trianglemf(x,a,b,c);
78     end
79
80     if n==5
81         mf=trapezoidmf(x,a,b,c,c+0.1666);
82     end
83
84     if n==0
85         mf=1;
86     end
```

```
87
88     if n<0 | n>5
89         'Unknown_membership_function.'
90     end
91
92 endfunction
93
94 function norm_val=normalize(val,l_min,l_max)
95
96     norm_val=(val-l_min)/((l_max-l_min)+0.000001);
97     if norm_val>1
98         norm_val=1;
99     end
100
101     if norm_val<0
102         norm_val=0;
103     end
104
105 endfunction
106
107 function val=denormalize(norm_val,l_min,l_max)
108
109     val=norm_val*(l_max-l_min+0.000001)+l_min;
110
111     if val>l_max
112         val=l_max;
113     end
114
115     if val<l_min
116         val=l_min;
117     end
118
119 endfunction
120
121 function mf=trapezoidmf(x,a,b,c,d)
122     //Trapezoidal activation function
```

```

123      //The parameters a,b,c,d specifies the structure of the fuzzy set
124      mf=max(min(min((x-a)/(b-a+0.000001),1),(d-x)/(d-c+0.000001)),0);
125  endfunction
126
127  function mf=trianglemf(x,a,b,c)
128      //Triangular fuzzy setg
129      //The parameters a,b,c,d specifies the structure of the fuzzy set.
130      mf = max(min((x-a)/(b-a+0.000001),(c-x)/(c-b+0.000001)),0);
131  endfunction

```

■ Código 3 (Función para realizar animación de la simulación)

```

1  function showLastData()
2
3  load('/home/victorm/Desktop/Thesis/Variables/Data.sod');
4  load('/home/victorm/Desktop/Thesis/Variables/AnimData.sod');
5
6      QuadAnimation(AnimData,result);
7
8  endfunction
9
10 function AnimData=QuadAnimData(Data,cycle)
11
12 // Block 1 specification
13 Lx_1 = 0.5;
14 Ly_1 = 0.08;
15 Lz_1 = 0.04;
16
17 // Block 2 specification
18 Lx_2 = 0.08;
19 Ly_2 = 0.5;
20 Lz_2 = 0.04;
21
22 //Cylinder 3 specifications (Center)
23 Radius_3 = 0.06;
24 Height_3 = 0.05;

```

```

25 SideCount_3=10;
26
27 //Cylinder 4,5,6,7 specifications (Propellers)
28 Radius_P = 0.06;
29 Height_P = 0.05;
30 SideCount_P=10;
31
32 // Motion data
33 t = Data.t'; // Time data
34 r = Data.x(:, :, cycle)'; // Position data
35 A = Data.theta(:, :, cycle)'; // Orientation data (x-y-z Euler angle)
36
37 n_time = length(t);
38
39 // Compute propagation of vertices and patches
40 for i_time=1:n_time
41
42     R = Euler2R(A(i_time, :));
43     VertexData_1(:, :, i_time) = GeoVerMakeBlock(r(i_time, :), R, ...
44     [Lx_1, Ly_1, Lz_1]);
45     VertexData_2(:, :, i_time) = GeoVerMakeBlock(r(i_time, :), R, ...
46     [Lx_2, Ly_2, Lz_2]);
47     VertexData_3(:, :, i_time) = GeoVerMakeCylinder(r(i_time, :), R, ...
48     Radius_3, Height_3, SideCount_3);
49     VertexData_4(:, :, i_time) = GeoVerMakePropeller1(r(i_time, :), R, ...
50     Radius_P, Height_P, SideCount_P, Lx_1);
51     VertexData_5(:, :, i_time) = GeoVerMakePropeller2(r(i_time, :), R, ...
52     Radius_P, Height_P, SideCount_P, Lx_1);
53     VertexData_6(:, :, i_time) = GeoVerMakePropeller3(r(i_time, :), R, ...
54     Radius_P, Height_P, SideCount_P, Ly_2);
55     VertexData_7(:, :, i_time) = GeoVerMakePropeller4(r(i_time, :), R, ...
56     Radius_P, Height_P, SideCount_P, Ly_2);
57     [X_1, Y_1, Z_1] = GeoPatMakeBlock(VertexData_1(:, :, i_time));
58     [X_2, Y_2, Z_2] = GeoPatMakeBlock(VertexData_2(:, :, i_time));
59     [X1_3, Y1_3, Z1_3, X2_3, Y2_3, Z2_3] = ...
60     GeoPatMakeCylinder(VertexData_3(:, :, i_time), SideCount_3);

```

```
61 [X1_4, Y1_4, Z1_4, X2_4, Y2_4, Z2_4] = ...
62 GeoPatMakeCylinder(VertexData_4(:, :, i_time), SideCount_P);
63 [X1_5, Y1_5, Z1_5, X2_5, Y2_5, Z2_5] = ...
64 GeoPatMakeCylinder(VertexData_5(:, :, i_time), SideCount_P);
65 [X1_6, Y1_6, Z1_6, X2_6, Y2_6, Z2_6] = ...
66 GeoPatMakeCylinder(VertexData_6(:, :, i_time), SideCount_P);
67 [X1_7, Y1_7, Z1_7, X2_7, Y2_7, Z2_7] = ...
68 GeoPatMakeCylinder(VertexData_7(:, :, i_time), SideCount_P);
69 Block1_X(:, :, i_time) = X_1;
70 Block1_Y(:, :, i_time) = Y_1;
71 Block1_Z(:, :, i_time) = Z_1;
72 Block2_X(:, :, i_time) = X_2;
73 Block2_Y(:, :, i_time) = Y_2;
74 Block2_Z(:, :, i_time) = Z_2;
75 Cylinder3_SideX(:, :, i_time) = X1_3;
76 Cylinder3_SideY(:, :, i_time) = Y1_3;
77 Cylinder3_SideZ(:, :, i_time) = Z1_3;
78 Cylinder3_BottomX(:, :, i_time) = X2_3;
79 Cylinder3_BottomY(:, :, i_time) = Y2_3;
80 Cylinder3_BottomZ(:, :, i_time) = Z2_3;
81 Cylinder4_SideX(:, :, i_time) = X1_4;
82 Cylinder4_SideY(:, :, i_time) = Y1_4;
83 Cylinder4_SideZ(:, :, i_time) = Z1_4;
84 Cylinder4_BottomX(:, :, i_time) = X2_4;
85 Cylinder4_BottomY(:, :, i_time) = Y2_4;
86 Cylinder4_BottomZ(:, :, i_time) = Z2_4;
87 Cylinder5_SideX(:, :, i_time) = X1_5;
88 Cylinder5_SideY(:, :, i_time) = Y1_5;
89 Cylinder5_SideZ(:, :, i_time) = Z1_5;
90 Cylinder5_BottomX(:, :, i_time) = X2_5;
91 Cylinder5_BottomY(:, :, i_time) = Y2_5;
92 Cylinder5_BottomZ(:, :, i_time) = Z2_5;
93 Cylinder6_SideX(:, :, i_time) = X1_6;
94 Cylinder6_SideY(:, :, i_time) = Y1_6;
95 Cylinder6_SideZ(:, :, i_time) = Z1_6;
96 Cylinder6_BottomX(:, :, i_time) = X2_6;
```

```

97     Cylinder6_BottomY (:, :, i_time) = Y2_6;
98     Cylinder6_BottomZ (:, :, i_time) = Z2_6;
99     Cylinder7_SideX (:, :, i_time) = X1_7;
100    Cylinder7_SideY (:, :, i_time) = Y1_7;
101    Cylinder7_SideZ (:, :, i_time) = Z1_7;
102    Cylinder7_BottomX (:, :, i_time) = X2_7;
103    Cylinder7_BottomY (:, :, i_time) = Y2_7;
104    Cylinder7_BottomZ (:, :, i_time) = Z2_7;
105    end
106
107    AnimData = struct('Block1_X',Block1_X,'Block1_Y', Block1_Y,'Block1_Z',...
108    Block1_Z,'Block2_X',Block2_X,'Block2_Y', Block2_Y,'Block2_Z', Block2_Z,...
109    'Cylinder3_SideX',Cylinder3_SideX,'Cylinder3_SideY',Cylinder3_SideY,...
110    'Cylinder3_SideZ',Cylinder3_SideZ,'Cylinder3_BottomX',Cylinder3_BottomX...
111    ,'Cylinder3_BottomY',Cylinder3_BottomY,'Cylinder3_BottomZ',Cylinder3_BottomZ,...
112    'Cylinder4_SideX',Cylinder4_SideX,'Cylinder4_SideY',Cylinder4_SideY,...
113    'Cylinder4_SideZ',Cylinder4_SideZ,'Cylinder4_BottomX',Cylinder4_BottomX,...
114    'Cylinder4_BottomY',Cylinder4_BottomY,'Cylinder4_BottomZ',Cylinder4_BottomZ,...
115    'Cylinder5_SideX',Cylinder5_SideX,'Cylinder5_SideY',Cylinder5_SideY,...
116    'Cylinder5_SideZ',Cylinder5_SideZ,'Cylinder5_BottomX',Cylinder5_BottomX,...
117    'Cylinder5_BottomY',Cylinder5_BottomY,'Cylinder5_BottomZ',Cylinder5_BottomZ,...
118    'Cylinder6_SideX',Cylinder6_SideX,'Cylinder6_SideY',Cylinder6_SideY,...
119    'Cylinder6_SideZ',Cylinder6_SideZ,'Cylinder6_BottomX',Cylinder6_BottomX,...
120    'Cylinder6_BottomY',Cylinder6_BottomY,'Cylinder6_BottomZ',Cylinder6_BottomZ,...
121    'Cylinder7_SideX',Cylinder7_SideX,'Cylinder7_SideY',Cylinder7_SideY,...
122    'Cylinder7_SideZ',Cylinder7_SideZ,'Cylinder7_BottomX',Cylinder7_BottomX,...
123    'Cylinder7_BottomY',Cylinder7_BottomY,'Cylinder7_BottomZ',Cylinder7_BottomZ,...
124    'LengthTime',n_time);
125
126    endfunction
127
128    function QuadAnimation(AnimData,Data,cycle)
129
130    n_time=AnimData.LengthTime;
131
132    //Graph data

```



```

133 subplot(1,2,1)
134 plot2d(Data.t,[Data.x(1, :, cycle)' Data.x(2, :, cycle)' Data.x(3, :, cycle)'] ,...
135 [5 3 2], leg="L1@L2@L3");
136 xlabel("Time_(s)", 'fontsize', 2);
137 ylabel("Position_(m)", 'fontsize', 2);
138 p = get("hdl");
139 p.children.thickness = 2;
140 l=legend(["$x$"; "$y$"; "$z$"]);
141 l.font_size = 3;
142 l.font_style = 1;
143 l.visible='on';
144 xgrid(1)
145
146 subplot(1,2,2)
147 plot2d(Data.t,[Data.theta(1, :, cycle)' Data.theta(2, :, cycle)' ...
148 Data.theta(3, :, cycle)'] , [5 3 2], leg="L1@L2@L3");
149 xlabel("Time_(s)", 'fontsize', 2);
150 ylabel("Attitude_(rad)", 'fontsize', 2);
151 p = get("hdl");
152 p.children.thickness = 2;
153 l=legend(["$\phi$"; "$\theta$"; "$\psi$"]);
154 l.font_size = 3;
155 l.font_style = 1;
156 l.visible='on';
157 xgrid(1)
158
159 // Draw initial figure
160 figure(2)
161 h_fig=gcf();
162 h_fig.background=7;
163 plot3d(AnimData.Block1_X(:, :, 1), AnimData.Block1_Y(:, :, 1), ...
164 AnimData.Block1_Z(:, :, 1));
165 h_fac3d1 = gce();
166 h_fac3d1.color_mode = 5;
167 h_fac3d1.foreground = 1;
168 h_fac3d1.hiddencolor = 5;

```

```
169
170 plot3d(AnimData.Block2_X(:, :, 1), AnimData.Block2_Y(:, :, 1), ...
171 AnimData.Block2_Z(:, :, 1));
172 h_fac3d2 = gce();
173 h_fac3d2.color_mode = 3;
174 h_fac3d2.foreground = 1;
175 h_fac3d2.hiddencolor = 3;
176
177 plot3d(AnimData.Cylinder3_SideX(:, :, 1), AnimData.Cylinder3_SideY(:, :, 1), ...
178 AnimData.Cylinder3_SideZ(:, :, 1));
179 h_fac3d3(1) = gce();
180 h_fac3d3(1).color_mode = -2;
181 h_fac3d3(1).foreground = 1;
182 h_fac3d3(1).hiddencolor = 2;
183
184 plot3d(AnimData.Cylinder3_BottomX(:, :, 1), AnimData.Cylinder3_BottomY(:, :, 1), ...
185 AnimData.Cylinder3_BottomZ(:, :, 1));
186 h_fac3d3(2) = gce();
187 h_fac3d3(2).color_mode = 2;
188 h_fac3d3(2).foreground = 1;
189 h_fac3d3(2).hiddencolor = 2;
190
191 plot3d(AnimData.Cylinder4_SideX(:, :, 1), AnimData.Cylinder4_SideY(:, :, 1), ...
192 AnimData.Cylinder4_SideZ(:, :, 1));
193 h_fac3d4(1) = gce();
194 h_fac3d4(1).color_mode = -2;
195 h_fac3d4(1).foreground = 1;
196 h_fac3d4(1).hiddencolor = 2;
197
198 plot3d(AnimData.Cylinder4_BottomX(:, :, 1), AnimData.Cylinder4_BottomY(:, :, 1), ...
199 AnimData.Cylinder4_BottomZ(:, :, 1));
200 h_fac3d4(2) = gce();
201 h_fac3d4(2).color_mode = 2;
202 h_fac3d4(2).foreground = 1;
203 h_fac3d4(2).hiddencolor = 2;
204
```

```
205 plot3d(AnimData.Cylinder5_SideX (:, :, 1), AnimData.Cylinder5_SideY (:, :, 1), ...
206 AnimData.Cylinder5_SideZ (:, :, 1));
207 h_fac3d5(1) = gce();
208 h_fac3d5(1).color_mode = -2;
209 h_fac3d5(1).foreground = 1;
210 h_fac3d5(1).hiddencolor = 2;
211
212 plot3d(AnimData.Cylinder5_BottomX (:, :, 1), AnimData.Cylinder5_BottomY (:, :, 1), ...
213 AnimData.Cylinder5_BottomZ (:, :, 1));
214 h_fac3d5(2) = gce();
215 h_fac3d5(2).color_mode = 2;
216 h_fac3d5(2).foreground = 1;
217 h_fac3d5(2).hiddencolor = 2;
218
219 plot3d(AnimData.Cylinder6_SideX (:, :, 1), AnimData.Cylinder6_SideY (:, :, 1), ...
220 AnimData.Cylinder6_SideZ (:, :, 1));
221 h_fac3d6(1) = gce();
222 h_fac3d6(1).color_mode = -2;
223 h_fac3d6(1).foreground = 1;
224 h_fac3d6(1).hiddencolor = 2;
225
226 plot3d(AnimData.Cylinder6_BottomX (:, :, 1), AnimData.Cylinder6_BottomY (:, :, 1), ...
227 AnimData.Cylinder6_BottomZ (:, :, 1));
228 h_fac3d6(2) = gce();
229 h_fac3d6(2).color_mode = 2;
230 h_fac3d6(2).foreground = 1;
231 h_fac3d6(2).hiddencolor = 2;
232
233 plot3d(AnimData.Cylinder7_SideX (:, :, 1), AnimData.Cylinder7_SideY (:, :, 1), ...
234 AnimData.Cylinder7_SideZ (:, :, 1));
235 h_fac3d7(1) = gce();
236 h_fac3d7(1).color_mode = -2;
237 h_fac3d7(1).foreground = 1;
238 h_fac3d7(1).hiddencolor = 2;
239
240 plot3d(AnimData.Cylinder7_BottomX (:, :, 1), AnimData.Cylinder7_BottomY (:, :, 1), ...
```

```

241 AnimData.Cylinder7_BottomZ(:, :, 1));
242 h_fac3d7(2) = gce();
243 h_fac3d7(2).color_mode = 2;
244 h_fac3d7(2).foreground = 1;
245 h_fac3d7(2).hiddencolor = 2;
246
247 // Axes settings
248 xlabel("x", 'fontsize', 2);
249 ylabel("y", 'fontsize', 2);
250 zlabel("z", 'fontsize', 2);
251 h_axes = gca();
252 h_axes.font_size = 2;
253 h_axes.isoview = "on";
254 h_axes.box = "off";
255 //h_axes.rotation_angles = [63.5, -127];
256 h_axes.rotation_angles = [65, -100];
257 //h_axes.data_bounds = [-4, -0.2, 0; 0.25, 4.1, 2.5];
258 //h_axes.data_bounds = [0, -17, 0; 36, 0, 2.5];
259 //h_axes.data_bounds = [-4, -1, 0; 0.25, 1, 2.3];
260 h_axes.data_bounds = [-0.4, -0.4, 0; 0.4, 0.4, 2.5];
261 xgrid;
262
263 // Animation Loop
264 //figure(1)
265 n_fig=1;
266
267 for i_time=1:n_time
268
269     drawlater();
270     //h_axes.data_bounds = [-4, Data.x(2, i_time) - 1, 0; 0.25, ...
271     //Data.x(2, i_time) + 1, 2.5];
272     h_fac3d1.data.x = AnimData.Block1_X(:, :, i_time);
273     h_fac3d1.data.y = AnimData.Block1_Y(:, :, i_time);
274     h_fac3d1.data.z = AnimData.Block1_Z(:, :, i_time);
275     h_fac3d2.data.x = AnimData.Block2_X(:, :, i_time);
276     h_fac3d2.data.y = AnimData.Block2_Y(:, :, i_time);

```

```
277     h_fac3d2.data.z = AnimData.Block2_Z(:, :, i_time);
278     h_fac3d3(1).data.x = AnimData.Cylinder3_SideX(:, :, i_time);
279     h_fac3d3(1).data.y = AnimData.Cylinder3_SideY(:, :, i_time);
280     h_fac3d3(1).data.z = AnimData.Cylinder3_SideZ(:, :, i_time);
281     h_fac3d3(2).data.x = AnimData.Cylinder3_BottomX(:, :, i_time);
282     h_fac3d3(2).data.y = AnimData.Cylinder3_BottomY(:, :, i_time);
283     h_fac3d3(2).data.z = AnimData.Cylinder3_BottomZ(:, :, i_time);
284     h_fac3d4(1).data.x = AnimData.Cylinder4_SideX(:, :, i_time);
285     h_fac3d4(1).data.y = AnimData.Cylinder4_SideY(:, :, i_time);
286     h_fac3d4(1).data.z = AnimData.Cylinder4_SideZ(:, :, i_time);
287     h_fac3d4(2).data.x = AnimData.Cylinder4_BottomX(:, :, i_time);
288     h_fac3d4(2).data.y = AnimData.Cylinder4_BottomY(:, :, i_time);
289     h_fac3d4(2).data.z = AnimData.Cylinder4_BottomZ(:, :, i_time);
290     h_fac3d5(1).data.x = AnimData.Cylinder5_SideX(:, :, i_time);
291     h_fac3d5(1).data.y = AnimData.Cylinder5_SideY(:, :, i_time);
292     h_fac3d5(1).data.z = AnimData.Cylinder5_SideZ(:, :, i_time);
293     h_fac3d5(2).data.x = AnimData.Cylinder5_BottomX(:, :, i_time);
294     h_fac3d5(2).data.y = AnimData.Cylinder5_BottomY(:, :, i_time);
295     h_fac3d5(2).data.z = AnimData.Cylinder5_BottomZ(:, :, i_time);
296     h_fac3d6(1).data.x = AnimData.Cylinder6_SideX(:, :, i_time);
297     h_fac3d6(1).data.y = AnimData.Cylinder6_SideY(:, :, i_time);
298     h_fac3d6(1).data.z = AnimData.Cylinder6_SideZ(:, :, i_time);
299     h_fac3d6(2).data.x = AnimData.Cylinder6_BottomX(:, :, i_time);
300     h_fac3d6(2).data.y = AnimData.Cylinder6_BottomY(:, :, i_time);
301     h_fac3d6(2).data.z = AnimData.Cylinder6_BottomZ(:, :, i_time);
302     h_fac3d7(1).data.x = AnimData.Cylinder7_SideX(:, :, i_time);
303     h_fac3d7(1).data.y = AnimData.Cylinder7_SideY(:, :, i_time);
304     h_fac3d7(1).data.z = AnimData.Cylinder7_SideZ(:, :, i_time);
305     h_fac3d7(2).data.x = AnimData.Cylinder7_BottomX(:, :, i_time);
306     h_fac3d7(2).data.y = AnimData.Cylinder7_BottomY(:, :, i_time);
307     h_fac3d7(2).data.z = AnimData.Cylinder7_BottomZ(:, :, i_time);
308     h_axes.data_bounds = [Data.x(1, i_time, cycle) - 0.4, ...
309     Data.x(2, i_time, cycle) - 0.4, 0; Data.x(1, i_time, cycle) + 0.4, ...
310     Data.x(2, i_time, cycle) + 0.4, 2.5];
311     drawnow();
312
```

```

313     mul=modulo(i_time,5);
314
315     if mul == 0 then
316         s_fig=string(n_fig);
317         if n_fig < 10 then
318             file_fig="FS_quad000"+s_fig+".gif";
319         elseif n_fig <100 then
320             file_fig="FS_quad00"+s_fig+".gif";
321         elseif n_fig <1000 then
322             file_fig="FS_quad0"+s_fig+".gif";
323         else
324             file_fig="FS_quad"+s_fig+".gif";
325         end
326         xs2gif(gcf(), file_fig);
327         n_fig=n_fig+1;
328     end
329
330 end
331
332 endfunction
333
334 function R = Euler2R(A)
335
336 // Euler angle ZYX (Modified as paper Rotation Matrix)-> Orientation matrix
337 phi_a=A(1);
338 theta_a=A(2);
339 psi_a=A(3);
340
341 R=[(cos(psi_a)*cos(theta_a)),...
342     (cos(psi_a)*sin(theta_a)*sin(phi_a))-(cos(phi_a)*sin(psi_a)),...
343     (cos(psi_a)*sin(theta_a)*cos(phi_a))+(sin(psi_a)*sin(phi_a));...
344     sin(psi_a)*cos(theta_a),...
345     (sin(psi_a)*sin(theta_a)*sin(phi_a))+(cos(psi_a)*cos(phi_a)),...
346     (sin(psi_a)*sin(theta_a)*cos(phi_a))-(cos(psi_a)*sin(phi_a));...
347     -sin(theta_a), cos(theta_a)*sin(phi_a), cos(theta_a)*cos(phi_a)];
348

```

```

349 endfunction
350
351 function VertexData=GeoVerMakeBlock(Location , Orientation , SideLength)
352
353     r=Location; //Location of center of block
354     R=Orientation;
355     Lx=SideLength(1);
356     Ly=SideLength(2);
357     Lz=SideLength(3);
358
359     //Vertices
360     VertexData_0=[Lx*ones(8,1),Ly*ones(8,1),Lz*ones(8,1)]...
361     .*[0,0,0;
362         1,0,0;
363         0,1,0;
364         0,0,1;
365         1,1,0;
366         0,1,1;
367         1,0,1;
368         1,1,1];
369
370     n_ver=8;
371
372     for i_ver=1:n_ver
373         //Move center to location specified
374         VertexData(i_ver,:)= r+(VertexData_0(i_ver,:)-0.5*SideLength)*R';
375     end
376
377 endfunction
378
379 function [PatchData_X , PatchData_Y , PatchData_Z]=GeoPatMakeBlock(VertexData)
380
381     //Patches
382     Index_Patch=...
383     [1,2,5,3;
384     1,3,6,4;

```

```

385     1,4,7,2;
386     4,7,8,6;
387     2,5,8,7;
388     3,6,8,5];
389
390     n_pat=6;
391
392     for i_pat=1:n_pat
393         PatchData_X(:,i_pat)=VertexData(Index_Patch(i_pat,:),1);
394         PatchData_Y(:,i_pat)=VertexData(Index_Patch(i_pat,:),2);
395         PatchData_Z(:,i_pat)=VertexData(Index_Patch(i_pat,:),3);
396     end
397
398 endfunction
399
400 function VertexData=GeoVerMakeCylinder(Location,Orientation,Radius,...
401 Height,SideCount)
402
403     r=Location; //Location of center of block
404     R=Orientation;
405
406     //Vertices
407     n_side=SideCount;
408
409     for i_ver=1:n_side
410         //Height adjusted to locate center of the cylinder
411         //in location 'r' specified
412         VertexData_0(i_ver,:)= [Radius*cos(2*%pi/n_side*i_ver),...
413             Radius*sin(2*%pi/n_side*i_ver), -0.5*Height];
414         VertexData_0(n_side+i_ver,:)= [Radius*cos(2*%pi/n_side*i_ver),...
415             Radius*sin(2*%pi/n_side*i_ver),0.5*Height];
416     end
417
418     n_ver=2*n_side;
419
420     for i_ver=1:n_ver

```



```

421         VertexData(i_ver,:) = r + VertexData_0(i_ver,:) * R';
422     end
423
424 endfunction
425
426 function VertexData = GeoVerMakePropeller1(Location, Orientation, Radius, ...
427 Height, SideCount, Lx)
428
429     r = Location; //Location of center of block
430     R = Orientation;
431
432     //Vertices
433     n_side = SideCount;
434
435     for i_ver = 1:n_side
436         //X,Y: Adjusted to fix axis of rotation, Height: Adjusted to
437         //locate center of the cylinder
438         VertexData_0(i_ver,:) = [Radius*cos(2*%pi/n_side*i_ver) + (Lx/2), ...
439             Radius*sin(2*%pi/n_side*i_ver), -0.5*Height];
440         VertexData_0(n_side+i_ver,:) = [Radius*cos(2*%pi/n_side*i_ver) + (Lx/2), ...
441             Radius*sin(2*%pi/n_side*i_ver), 0.5*Height];
442     end
443
444     n_ver = 2*n_side;
445
446     for i_ver = 1:n_ver
447         VertexData(i_ver,:) = r + VertexData_0(i_ver,:) * R';
448     end
449
450 endfunction
451
452 function VertexData = GeoVerMakePropeller2(Location, Orientation, Radius, ...
453 Height, SideCount, Lx)
454
455     r = Location; //Location of center of block
456     R = Orientation;

```

```

457
458     //Vertices
459     n_side=SideCount;
460
461     for i_ver=1:n_side
462         //X,Y: Adjusted to fix axis of rotation, Height: Adjusted to
463         //locate center of the cylinder
464         VertexData_0(i_ver,:)=[Radius*cos(2*%pi/n_side*i_ver)-(Lx/2),...
465         Radius*sin(2*%pi/n_side*i_ver), -0.5*Height];
466         VertexData_0(n_side+i_ver,:)=[Radius*cos(2*%pi/n_side*i_ver)-(Lx/2),...
467         Radius*sin(2*%pi/n_side*i_ver),0.5*Height];
468     end
469
470     n_ver=2*n_side;
471
472     for i_ver=1:n_ver
473         VertexData(i_ver,:)=r+VertexData_0(i_ver,:)*R';
474     end
475
476 endfunction
477
478 function VertexData=GeoVerMakePropeller3(Location,Orientation,Radius,...
479 Height,SideCount,Ly)
480
481     r=Location; //Location of center of block
482     R=Orientation;
483
484     //Vertices
485     n_side=SideCount;
486
487     for i_ver=1:n_side
488         //X,Y: Adjusted to fix axis of rotation, Height: Adjusted to
489         //locate center of the cylinder
490         VertexData_0(i_ver,:)=[Radius*cos(2*%pi/n_side*i_ver),...
491         Radius*sin(2*%pi/n_side*i_ver)+(Ly/2), -0.5*Height];
492         VertexData_0(n_side+i_ver,:)=[Radius*cos(2*%pi/n_side*i_ver),...

```

```

493         Radius*sin(2*%pi/n_side*i_ver)+(Ly/2),0.5*Height];
494     end
495
496     n_ver=2*n_side;
497
498     for i_ver=1:n_ver
499         VertexData(i_ver,:)=r+VertexData_0(i_ver,:)*R';
500     end
501
502 endfunction
503
504 function VertexData=GeoVerMakePropeller4(Location,Orientation,Radius,...
505 Height,SideCount,Ly)
506
507     r=Location; //Location of center of block
508     R=Orientation;
509
510     //Vertices
511     n_side=SideCount;
512
513     for i_ver=1:n_side
514         //X,Y: Adjusted to fix axis of rotation, Height: Adjusted to
515         //locate center of the cylinder
516         VertexData_0(i_ver,:)= [Radius*cos(2*%pi/n_side*i_ver),...
517         Radius*sin(2*%pi/n_side*i_ver)-(Ly/2), -0.5*Height];
518         VertexData_0(n_side+i_ver,:)= [Radius*cos(2*%pi/n_side*i_ver),...
519         Radius*sin(2*%pi/n_side*i_ver)-(Ly/2),0.5*Height];
520     end
521
522     n_ver=2*n_side;
523
524     for i_ver=1:n_ver
525         VertexData(i_ver,:)=r+VertexData_0(i_ver,:)*R';
526     end
527
528 endfunction

```

```
529
530 function [PatchData1_X, PatchData1_Y, PatchData1_Z, PatchData2_X, ...
531 PatchData2_Y, PatchData2_Z]=GeoPatMakeCylinder(VertexData, SideCount)
532
533     n_side=SideCount;
534
535     //Side Patches
536     for i_pat=1:n_side-1
537         Index_Patch1(i_pat, :)= [i_pat, i_pat+1, n_side+i_pat+1, n_side+i_pat];
538     end
539
540     Index_Patch1(n_side, :)= [n_side, 1, n_side+1, 2*n_side];
541
542     for i_pat=1:n_side
543         //Side Patches Data
544         PatchData1_X(:, i_pat)=VertexData(Index_Patch1(i_pat, :), 1);
545         PatchData1_Y(:, i_pat)=VertexData(Index_Patch1(i_pat, :), 2);
546         PatchData1_Z(:, i_pat)=VertexData(Index_Patch1(i_pat, :), 3);
547     end
548
549     //Bottom Patches
550     Index_Patch2(1, :)= [1:n_side];
551     Index_Patch2(2, :)= [n_side+1:2*n_side];
552
553     for i_pat=1:2
554         //Bottom Patches Data
555         PatchData2_X(:, i_pat)=VertexData(Index_Patch2(i_pat, :), 1);
556         PatchData2_Y(:, i_pat)=VertexData(Index_Patch2(i_pat, :), 2);
557         PatchData2_Z(:, i_pat)=VertexData(Index_Patch2(i_pat, :), 3);
558     end
559
560 endfunction
```

APÉNDICE B

REGLAS DEL SISTEMA DIFUSO

El sistema difuso tiene una base de reglas con las cuales realiza la sintonización de cada ganancia. Las entradas al sistema corresponden a OS, SSE y ST; cuyos valores lingüísticos son 1: Muy Bajo, 2: Bajo, 3: Medio, 4: Alto y 5: Muy Alto. La salida del sistema corresponde a ΔK ; cuyos valores lingüísticos son 1: Muy Negativo, 2: Negativo, 3: Nulo, 4: Positivo y 5: Muy Positivo.

B.1 REGLAS PARA EL AJUSTE DE K_p

<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p	<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p	<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p
1	1	1	3	1	3	1	3	1	5	1	3
1	1	2	3	1	3	2	3	1	5	2	3
1	1	3	3	1	3	3	3	1	5	3	4
1	1	4	4	1	3	4	3	1	5	4	5
1	1	5	4	1	3	5	3	1	5	5	5
1	2	1	3	1	4	1	4	2	1	1	4
1	2	2	3	1	4	2	4	2	1	2	4
1	2	3	3	1	4	3	4	2	1	3	4
1	2	4	3	1	4	4	4	2	1	4	4
1	2	5	3	1	4	5	4	2	1	5	4

<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p	<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p	<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p
2	2	1	4	3	2	5	3	4	3	4	3
2	2	2	4	3	3	1	3	4	3	5	4
2	2	3	4	3	3	2	3	4	4	1	3
2	2	4	4	3	3	3	3	4	4	2	3
2	2	5	4	3	3	4	3	4	4	3	3
2	3	1	4	3	3	5	3	4	4	4	3
2	3	2	4	3	4	1	3	4	4	5	4
2	3	3	4	3	4	2	3	4	5	1	4
2	3	4	4	3	4	3	3	4	5	2	3
2	3	5	4	3	4	4	3	4	5	3	3
2	4	1	3	3	4	5	3	4	5	4	4
2	4	2	3	3	5	1	3	4	5	5	4
2	4	3	3	3	5	2	3	5	1	1	2
2	4	4	3	3	5	3	3	5	1	2	2
2	4	5	4	3	5	4	3	5	1	3	3
2	5	1	4	3	5	5	3	5	1	4	4
2	5	2	5	4	1	1	3	5	1	5	5
2	5	3	5	4	1	2	3	5	2	1	2
2	5	4	5	4	1	3	3	5	2	2	2
2	5	5	5	4	1	4	3	5	2	3	2
3	1	1	3	4	1	5	4	5	2	4	2
3	1	2	3	4	2	1	3	5	2	5	2
3	1	3	3	4	2	2	3	5	3	1	2
3	1	4	3	4	2	3	3	5	3	2	2
3	1	5	3	4	2	4	3	5	3	3	3
3	2	1	3	4	2	5	4	5	3	4	3
3	2	2	3	4	3	1	3	5	3	5	3
3	2	3	3	4	3	2	3	5	4	1	3
3	2	4	3	4	3	3	3	5	4	2	3

OS	SSE	ST	ΔK_p	OS	SSE	ST	ΔK_p	OS	SSE	ST	ΔK_p
5	4	3	3	5	5	1	3	5	5	4	4
5	4	4	3	5	5	2	3	5	5	5	5
5	4	5	3	5	5	3	4				

B.2 REGLAS PARA EL AJUSTE DE K_i

OS	SSE	ST	ΔK_p	OS	SSE	ST	ΔK_p	OS	SSE	ST	ΔK_p
1	1	1	3	1	5	1	4	2	4	1	4
1	1	2	3	1	5	2	4	2	4	2	4
1	1	3	3	1	5	3	4	2	4	3	4
1	1	4	3	1	5	4	4	2	4	4	4
1	1	5	3	1	5	5	4	2	4	5	4
1	2	1	4	2	1	1	3	2	5	1	4
1	2	2	4	2	1	2	3	2	5	2	4
1	2	3	4	2	1	3	3	2	5	3	4
1	2	4	3	2	1	4	3	2	5	4	4
1	2	5	3	2	1	5	3	2	5	5	4
1	3	1	3	2	2	1	3	3	1	1	3
1	3	2	3	2	2	2	3	3	1	2	3
1	3	3	3	2	2	3	3	3	1	3	3
1	3	4	3	2	2	4	3	3	1	4	3
1	3	5	3	2	2	5	3	3	1	5	3
1	4	1	3	2	3	1	3	3	2	1	3
1	4	2	3	2	3	2	3	3	2	2	3
1	4	3	3	2	3	3	3	3	2	3	3
1	4	4	3	2	3	4	3	3	2	4	3
1	4	5	3	2	3	5	3	3	2	5	3

<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p	<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p	<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p
3	3	1	3	4	2	3	3	5	1	5	3
3	3	2	3	4	2	4	3	5	2	1	3
3	3	3	3	4	2	5	3	5	2	2	3
3	3	4	3	4	3	1	3	5	2	3	3
3	3	5	3	4	3	2	3	5	2	4	3
3	4	1	4	4	3	3	3	5	2	5	3
3	4	2	4	4	3	4	3	5	3	1	3
3	4	3	4	4	3	5	3	5	3	2	3
3	4	4	4	4	4	1	3	5	3	3	3
3	4	5	4	4	4	2	3	5	3	4	3
3	5	1	4	4	4	3	2	5	3	5	3
3	5	2	4	4	4	4	2	5	4	1	3
3	5	3	4	4	4	5	2	5	4	2	3
3	5	4	4	4	5	1	2	5	4	3	3
3	5	5	3	4	5	2	2	5	4	4	4
4	1	1	3	4	5	3	2	5	4	5	4
4	1	2	3	4	5	4	2	5	5	1	5
4	1	3	3	4	5	5	2	5	5	2	5
4	1	4	3	5	1	1	3	5	5	3	5
4	1	5	3	5	1	2	3	5	5	4	4
4	2	1	3	5	1	3	3	5	5	5	4
4	2	2	3	5	1	4	3				

B.3 REGLAS PARA EL AJUSTE DE K_d

OS	SSE	ST	ΔK_p	OS	SSE	ST	ΔK_p	OS	SSE	ST	ΔK_p
1	1	1	3	2	1	1	4	3	1	1	3
1	1	2	3	2	1	2	4	3	1	2	3
1	1	3	4	2	1	3	4	3	1	3	4
1	1	4	4	2	1	4	5	3	1	4	5
1	1	5	4	2	1	5	5	3	1	5	5
1	2	1	3	2	2	1	4	3	2	1	3
1	2	2	3	2	2	2	4	3	2	2	3
1	2	3	4	2	2	3	4	3	2	3	4
1	2	4	4	2	2	4	4	3	2	4	4
1	2	5	4	2	2	5	4	3	2	5	5
1	3	1	4	2	3	1	4	3	3	1	3
1	3	2	4	2	3	2	4	3	3	2	3
1	3	3	4	2	3	3	4	3	3	3	3
1	3	4	4	2	3	4	5	3	3	4	4
1	3	5	4	2	3	5	5	3	3	5	5
1	4	1	4	2	4	1	3	3	4	1	3
1	4	2	5	2	4	2	3	3	4	2	4
1	4	3	5	2	4	3	3	3	4	3	4
1	4	4	5	2	4	4	4	3	4	4	5
1	4	5	5	2	4	5	5	3	4	5	5
1	5	1	3	2	5	1	3	3	5	1	3
1	5	2	3	2	5	2	3	3	5	2	3
1	5	3	3	2	5	3	3	3	5	3	4
1	5	4	3	2	5	4	4	3	5	4	4
1	5	5	3	2	5	5	5	3	5	5	3

<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p	<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p	<i>OS</i>	<i>SSE</i>	<i>ST</i>	ΔK_p
4	1	1	4	4	4	3	4	5	2	5	3
4	1	2	4	4	4	4	5	5	3	1	5
4	1	3	4	4	4	5	5	5	3	2	5
4	1	4	4	4	5	1	4	5	3	3	5
4	1	5	4	4	5	2	4	5	3	4	5
4	2	1	4	4	5	3	4	5	3	5	5
4	2	2	4	4	5	4	5	5	4	1	4
4	2	3	4	4	5	5	5	5	4	2	4
4	2	4	4	5	1	1	3	5	4	3	4
4	2	5	5	5	1	2	3	5	4	4	4
4	3	1	4	5	1	3	4	5	4	5	4
4	3	2	4	5	1	4	5	5	5	1	5
4	3	3	4	5	1	5	5	5	5	2	5
4	3	4	5	5	2	1	3	5	5	3	4
4	3	5	5	5	2	2	3	5	5	4	4
4	4	1	4	5	2	3	3	5	5	5	4
4	4	2	4	5	2	4	3				

RESUMEN AUTOBIOGRÁFICO

Víctor de Jesús Medrano Zarazúa

Candidato para obtener el grado de
Maestro en Ciencias de la Ingeniería Eléctrica
con orientación en Mecatrónica

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Tesis:

ESTABILIZACIÓN DE VUELO EN UAVS CON SINTONIZACIÓN
INTELIGENTE DE CONTROLADORES CONVENCIONALES

Nació en la ciudad de Monterrey, Nuevo León el 17 de octubre de 1992. Hijo de Víctor Medrano y Lucy Zarazúa. Estudió y se tituló como Ingeniero en Mecatrónica por parte de la FIME en la UANL (Generación 2009 - 2014). Ha tenido experiencia profesional en el campo de la electrónica. Sus principales intereses son la electrónica, programación, tecnologías emergentes, matemáticas, física, música, cine y literatura.