UNIVERSIDAD CARLOS III DE MADRID

Vision-based fuzzy controller for unmanned aerial vehicles (UAVs)

TRABAJO FIN DE GRADO

Rubén Portillo Bustos

Grado en ingeniería electrónica industrial y automática

Tutor: *Abdulla Hussein Abdulrahman Al-Kaff*

Director: *José María Armingol Moreno*

Departamento: *Ingeniería de sistemas y automática*

2014/2015

AGRADECIMIENTOS

Me gustaría agradecer por todo su esfuerzo y paciencia a mi familia, mis amigos y Cris, que han hecho de mí lo que soy y me ayudarán a hacer de mí lo que seré.

RESUMEN

En el presente trabajo se propone desarrollar un controlador de estabilidad y posición para un UAV, más en concreto para el Parrot ARDrone 2.0, basado en lógica difusa.

Para ello primero se creará un filtro diseñado para sistemas físicos que se encargará de atenuar el posible ruido proveniente de los sensores. A continuación, se iniciará el desarrollo del controlador, el cual contará con dos versiones, una que sólo tendrá como entradas la velocidad en los ejes "X" e "Y" detectada por el IMU (Inertial Measure Unit) y los ángulos (pitch, roll, yaw) detectados por el giróscopo. Y otra más avanzada que, manteniendo los datos obtenidos del IMU, obtendrá los ángulos y la posición en "X" e "Y" de una cámara de alta definición a partir de odometría y algoritmos derivados de la visión por computador que permitirán al usuario establecer el punto en el que el dron ha de situarse.

Cabe destacar que con la correcta configuración de los filtros, adecuándose a la capacidad de los sensores y el sistema, no importa el origen de la fuente de datos, siempre y cuando se mantengan los ejes de coordenadas utilizados en el presente trabajo.

ABSTRACT

In this project, it is proposed to develop a stability and position controller based on fuzzy logic for UAVs, specifically the ARDrone 2.0 from Parrot firm.

Firstly it will be created a filter properly designed for physical systems which will attenuate any noise in the sensors. Next to this, it will start the controller development. It will include two different versions: the first one will use as inputs IMU (Inertial Measure Unit) and gyroscope data. Instead, the later will use in addition to IMU data, positional and angular one obtained by visual algorithms from a high definition camera, which will let the user to set the point where the system has to move to.

It should be noted however that it is irrelevant the source of data provided that there is a proper configuration of filters, adjusted to sensors and system properties, as well as keeping axes orientation used in this work.

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ABREVIATURAS

- **UAV:** *Unmanned Aerial Vehicle, vehículo aéreo no tripulado.*
- **UAS:** *Unmanned Aircraft System, vehículo aéreo autónomo.*
- **IMU:** Inertial Measure Unit, unidad de medición inercial
- **PID:** Proporcional Integral Derivativo

1 INTRODUCCIÓN

1.1 MOTIVACIÓN

La motivación del presente trabajo reside en la creciente evolución de la tecnología de los vehículos aéreos no tripulados en la actualidad (conocidos en inglés como Unmanned Aerial Vehicles) y la importancia de su entendimiento y capacidad de manejo como ejemplo de sistema de control.

En este caso al tratarse de un equipo con configuración de quadrotor o cuadrirrotor hereda todas las propiedades de este tipo de helicópteros como son su estabilidad en vuelo o su capacidad de despegue en poco espacio. Lo que hace interesante el estudio del funcionamiento de su sistema de control y su desarrollo desde otra perspectiva como puede ser un controlador borroso.

Actualmente el Laboratorio de Sistemas Inteligentes de la universidad Carlos III trabaja en el desarrollo de una aplicación de control de un cuadricóptero utilizando conceptos tan interesantes como la visión por computador lo que no añade sino valor al presente trabajo.

1.2 OBJETIVOS Y ENFOQUE

El objetivo de este trabajo consiste en desarrollar un sistema de control basado en lógica "Fuzzy" que utilice datos proporcionados por un sistema de visión por computador que permita la estabilización y la correcta navegación del equipo.

1.3 ELECCIÓN Y DESCRIPCIÓN DEL HARDWARE

El hardware usado para la realización del presente proyecto es el "Parrot ARDrone 2.0" elegido por su bajo coste, su accesibilidad de compra y la calidad y cantidad de sensores que incorpora: cámaras, IMU y giróscopo entre otros.

Ilustración 1: Parrot ARDrone 2.0

El equipo dispone de hardware cualificado como sus cuatro rotores sin escobillas de 14,5W, su capacidad de control por red WiFi o una cámara HD a 720p frontal. [1]

2 ESTADO DEL ARTE

2.1 TECNOLOGÍA UAV

Hasta hace relativamente poco tiempo todo lo relacionado con los UAVs era muy inusual, debido al gran coste que suponía desarrollar este tipo de tecnología. Su origen, como todo lo relacionado con innovación tecnológica, nace del ejército a mediados del siglo XIX, donde un arcaico UAV sirvió en un ataque austriaco a Venecia [2].

Actualmente, gracias a la reducción general de costes en componentes como sensores, motores, o cámaras, se ha disparado la investigación y desarrollo de los UAVs poniendo a disposición de científicos y desarrolladores las herramientas necesarias para su correcto estudio.

Dependiendo de la función con la que se diseñe, el UAV variará sus características principales: tamaño, tipo de sensores... Cuando se dice que un vehículo es no tripulado, se refiere a que físicamente no hay una persona en el interior de la nave, aunque puede ser teleoperado desde una estación de tierra.

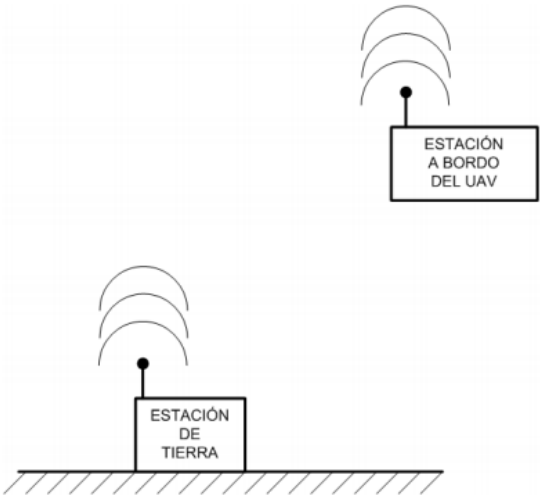


Ilustración 2: Estaciones de un UAV

Con el progreso de la investigación tecnológica, nació lo que se conoce como UAS (Unmanned Aerial System). La principal diferencia es que el último cuenta con un sistema autónomo, que previamente programado podrá seguir una trayectoria además de que estará preparado para tomar decisiones autónomas en función de su entorno.

Es importante resaltar que no todos los vehículos aéreos no tripulados pueden ser considerados UAVs o UASs, para esto tienen que cumplir ciertos requisitos como no ser desechables, y poder realizar un vuelo de retorno. Por ejemplo, un misil aunque posea tecnología propia de los UAVs no entra dentro de esta clasificación.

Destacamos que la empresa de origen francés Parrot, sacó al mercado uno de los primeros UAVs del mercado; el ARDrone 1.0 que supone un hito en los UAVs con fines recreativos. Este dron con configuración de cuadricóptero, una amplia gama de sensores procesados por su microprocesador y su conexión WiFi le permite ser manejado a través de un "Smartphone" convencional. [3]

Ilustración 3: ARDrone 1.0

2.2 VISIÓN POR COMPUTADOR

Desde hace unos años, el uso de cámaras no se ha reducido solo a tomar fotos o grabar vídeos. Gracias al gran avance en algoritmos de análisis de imágenes, actualmente podemos usar las cámaras para un sinfín de utilidades, tales como detección de objetos, situación en el espacio, o interpretación de la cara de una persona.

Primero vamos a definir qué es la visión por computador, para que el nombre no nos lleve a error. La visión por computador es el proceso de realizar transformaciones en una imagen para destacar unos eventos sobre otros. [4]

Para utilizar las cámaras como un sensor, se ha desarrollado una gran variedad de tipos de cámara dependiendo del uso posterior que se le vaya a dar. Estamos hablando de cámaras infrarrojas, que detectan el calor; cámaras ultravioleta usadas para la visión nocturna, etc.

En este trabajo, aunque no directamente, se utilizan las librerías EMGU CV, que es una plataforma que permite usar las librerías OPEN CV en lenguajes de tipo .NET como C#, VB, VC++, IronPython etc.

3 HARDWARE Y SOFTWARE DEL SISTEMA

3.1 HARDWARE

El elemento hardware principal elegido para realizar este trabajo es el ARDrone 2.0. Se trata de un vehículo aéreo no tripulado de radiocontrol diseñado para uso recreativo civil distribuido por la empresa "Parrot".

Consta de cuatro motores eléctricos en configuración de cuadricóptero para propulsarse. La primera versión de este producto fue presentada y lanzada en "Las Vegas International Consumer Electronics Show" del año 2010. Tras una exhaustiva revisión y actualización del producto, en el mismo evento del año 2012, Parrot lanzó al mercado el ARDrone 2.0.

Este equipo, debido a su fin recreativo, es similar a otros drones disponibles en el mercado, pero se diferencia del resto en la cantidad y calidad de los sensores que incorpora y en la inclusión de un microprocesador muy potente. Algunos de los sensores más significativos son: un sensor de ultrasonidos, un acelerómetro en los 3 ejes o dos cámaras (una frontal y otra vertical).

Una de sus elementos diferenciadores sin duda es su forma de control, que obvia el "típico" mando físico de control y opta por un sistema mucho más moderno. El propio ARDrone crea una red Wi-Fi local a través de la cual nos podemos conectar desde un dispositivo móvil personal que corra los sistemas operativos de Apple o Google o directamente desde nuestro PC lo que nos permite recibir sus datos de telemetría e incluso una recepción de imágenes en directo de forma remota.

Ahora vamos a analizar brevemente las características principales analizando la siguiente ilustración. [5]

Ilustración 4: Características principales del ARDrone 2.0

1. Cámara HD. 720p 30 fps.

2. Objetivo gran angular: 92°, diagonal. Perfil básico de codificación H264. Difusión en tiempo real de baja latencia.
3. Placa Madre
    a. Procesador AR M Cortex A8 de 32 bits a 1 GHz con DSP de video TMS320DMC64x a 800 MHz. Linux 2.6.32RAM DDR2 de 1 GB a 200 MHz.
    b. USB 2.0 de alta velocidad para extensiones y grabación.
    c. Wi-Fi b/g/n.
    d. Giroscopio de 3 ejes 2.0000/s.
    e. Acelerómetro de 3 ejes +/-50 mg.
    f. Magnetómetro de 3 ejes, precisión 6°.
    g. Sensor de presión +/- 10 Pa.
    h. Sensor de ultrasonidos para medición de altitud respecto al suelo.
    i. Cámara QVGA vertical a 60 fps.
4. Tubos de fibra de carbono: peso total de 380 g con el casco de protección para el exterior, 420 g con el casco de protección para el interior.
5. Piezas de plástico nylon cargado con 30% de fibra de vidrio de alta calidad.
6. Espuma para aislar el centro de inercia de las vibraciones de los motores.
7. Casco de protección de polipropileno expandido (PPE) inyectado por un molde metálico.
8. Nanorevestimiento repelente a los líquidos en los sensores de ultrasonidos.
9. 4 motores de rotor interno ("inrunner") sin escobillas. 14,5 W.
10. Rodamiento de bolas en miniatura.
11. Engranajes de Nylatron de bajo ruido para reductor de hélice de 1/8,75.
12. Eje de las hélices de acero templado.
13. Cojinete de bronce autolubricante.
14. Hélices de alta fuerza de propulsión para mayor maniobrabilidad.
15. Microcontrolador AVR de 8 MIPS.
16. Batería recargable Li-Po de 3 elementos, 1.000 mAh.
17. Conexión USB.

3.1.1 MODIFICACIONES EN EL HARDWARE

Debido a que la posición donde se requiere mayor calidad de imagen es la inferior, a partir de la cual se obtendrán datos como posición o ángulos se ha realizado una pequeña modificación orientando la cámara de alta resolución hacia el suelo.

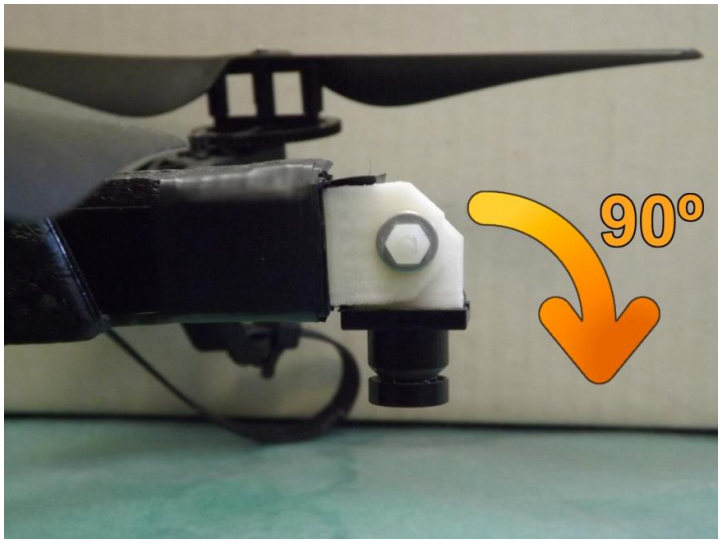


Ilustración 5: Modificación cámara

3.2 SOFTWARE

El Laboratorio de Sistemas Inteligentes de la Universidad Carlos III de Madrid está desarrollando un software de control para el ARDrone (llamado LSIDroneInterface) con diversas características de navegación autónoma basado en el SDK (Software Development Kit) distribuido abiertamente por la compañía Parrot para desarrolladores externos. Este SDK contiene las librerías necesarias para poder realizar la conexión con el dron desde la lectura de los datos de sus sensores hasta el envío de comandos a los actuadores.

A partir de estas librerías y utilizando el entorno de desarrollo integrado Microsoft Visual Studio y el lenguaje de programación C#, se ha desarrollado una aplicación de Windows muy completa que permite múltiples funcionalidades.

3.2.1 LENGUAJE C#

C# es un lenguaje de programación que se ha diseñado para compilar diversas aplicaciones que se ejecutan en .NET Framework. C# es simple, eficaz, con seguridad de tipos y orientado a objetos, que permite desarrollar aplicaciones rápidamente y mantener la expresividad y elegancia de los lenguajes de estilo de C.

La sintaxis de C# es muy fácil de aprender y utilizar, a la par que muy expresiva. Se basa en signos de llave, por lo que es fácilmente reconocible por cualquier persona familiarizada con C, C++ o Java. El lenguaje C# simplifica muchas de las complejidades de C++ y proporciona características eficaces como tipos de valor que admiten valores NULL, enumeraciones, delegados, expresiones lambda y acceso directo a memoria, que no se encuentran en Java. C# admite métodos y tipos genéricos, que proporcionan mayor rendimiento y seguridad de tipos, e iteradores, que permiten a los implementadores de clases de colección definir comportamientos de iteración personalizados que el código cliente puede utilizar fácilmente. Todas estas características convierten C# en un lenguaje de primera clase.

Como lenguaje orientado a objetos, C# admite los conceptos de encapsulación, herencia y polimorfismo. Todas las variables y métodos, incluido el método Main, que es el punto de entrada de la aplicación, se encapsulan dentro de definiciones de clase. Una clase puede heredar directamente de una clase primaria, pero puede implementar cualquier número de interfaces. Los métodos que reemplazan a los métodos virtuales en una clase primaria requieren la palabra clave override como medio para evitar redefiniciones accidentales. En C#, una struct es como una clase sencilla; es un tipo asignado en la pila que puede implementar interfaces pero que no admite la herencia.

Además de estos principios básicos orientados a objetos, C# facilita el desarrollo de componentes de software a través de varias construcciones de lenguaje innovadoras, entre las que se incluyen las siguientes:

- Firmas de métodos encapsulados denominadas delegados, que habilitan notificaciones de eventos con seguridad de tipos.

- Propiedades, que actúan como descriptores de acceso para variables miembro privadas.

- Atributos, que proporcionan metadatos declarativos sobre tipos en tiempo de ejecución.

- Comentarios en línea de documentación XML.

- Language-Integrated Query (LINQ) que proporciona funciones de consulta integradas en una gran variedad de orígenes de datos.

Esta información ha sido extraída de la web oficial de Microsoft C# [6].

3.2.2 SDK

El SDK o Kit de Desarrollo de Software es una serie de librerías que permiten a desarrolladores externos distribuir nuevas aplicaciones o juegos basados en el ARDrone 2.0. Para descargar estas librerías de desarrollo en necesario registrarse en la página web del fabricante y aceptar una serie acuerdos y condiciones.

Este SDK incluye:

* Un documento explicativo de cómo usar el SDK, donde también se describen los protocolos de comunicación del drone.

* La librería AR.Drone 2.0 (ARDroneLIB), la cual proporciona las APIs necesarias para una fácil comunicación y configuración del drone.

* La librería de herramientas (ARDroneTool), que ofrece un cliente del drone donde los desarrolladores sólo tienen que insertar su código específico de aplicaciones personalizadas;

* La librería de control de motores.

* El código fuente para iOS y Android de la aplicación AR.FreeFlight 2.0.

4 FILTROS

4.1 NECESIDAD

Todos los sensores tienen un margen de error que puede "entorpecer" las funciones de sistemas tan fundamentales como es el sistema de control de un dispositivo.

Para evitar esa incertidumbre en la recepción de datos es necesaria la implementación de filtros capaces de ofrecer un mínimo de fiabilidad a los datos.

Para este caso, implementaremos un filtro a paso bajo que elimine las altas frecuencias.

4.2 MEDIANA

En un primer momento se optó por el filtro "mediana". Para explicar la mediana, realizaremos un pequeño y simple ejemplo. Imaginemos que tenemos un sensor que da los siguientes valores, de los cuales usaremos para hacer el filtro los 5 últimos.

Ilustración 6: Ejemplo mediana 1

El siguiente paso sería recolocar los datos en orden ascendente y eligir el valor que quede en la posición central, en este caso 7.

Ilustración 7: Ejemplo mediana 2

Por lo que nuestro valor filtrado sería el 7 y nuestra sucesión quedaría así:

Ilustración 8: Ejemplo mediana 3

Este filtro es efectivo eliminando las altas frecuencias, pero en un sistema dinámico no permite realizar un seguimiento preciso. Por lo que quedó descartado.

En este punto, se empezó a desarrollar un nuevo filtro al que llamaremos "Follower Filter".

4.3 FOLLOWER FILTER

El filtro utilizado para este caso ha sido hecho a medida. A diferencia de otros filtros, este se ajusta a las limitaciones del propio sistema siendo válido, con la correcta configuración, para todos los sistemas.

Este sistema requiere para su funcionamiento un historial de los últimos valores recibidos del sensor y los últimos valores filtrados, un número de datos a analizar, una variable determinada por el ingeniero denominada "rango", que indicará la variación máxima que puede realizar ese valor considerando las limitaciones del sistema y una variable denominada "Follower_Range" comprendida entre el 0 y el 1 que determinará la agresividad del sistema cuando un valor se salga del rango establecido.

El proceso del filtro sería:

1) Comprobar si el nuevo dato está en rango con el último dato filtrado
    a. Si está en rango, se acepta el dato
    b. Si no está en rango, pasa al paso 2.
2) Comprobar si el nuevo dato está en rango con el número de valores anteriores sin filtrar (determinado como número de datos a analizar).
    a. Si está en rango, se acepta el dato (ya que el sistema, debido a un estímulo externo, ha excedido las limitaciones físicas previstas)
    b. Si no está en rango, pasa al paso 3.
3) Se aplica el filtro al valor recibido, multiplicando el rango máximo que consideramos plausible por la variable "follower_range" y sumarlo o restarlo al último valor filtrado.

Podemos observar el código en el anexo A del trabajo.

Para entender este filtro vamos a establecer un ejemplo concreto:

Imaginemos que queremos filtrar la posición en el eje X de un ratón, y que este lleva un GPS, que puede dar error. El ratón según un estudio no puede variar su posición por sí mismo 10 cm entre cada muestreo, por lo que el valor de rango será "rango = 10". Por otro lado decidimos que queremos que el filtro sea agresivo, por lo que definimos "follower_range=0,75". También decidimos que analizaremos la dinámica del sistema de los últimos 3 valores.

En el primer caso vamos a suponer que hemos recibido los siguientes datos del sensor:

- Valores dados por el sensor: 1, 2, 5, 20.

De 1 a 2 (|2-1|=1<10) y de 2 a 5 (|5-2|=3<10) es una variación que podría ser real y está dentro de rango. Ahora la variación de 5 a 20 (|100-5|=95>10) está fuera de rango; como los 3 valores anteriores sin filtrar tampoco están en rango, se aplica el filtro y el valor almacenado es 5+10*0.75=12.5.

Ahora imaginemos que hemos tenido esta progresión:

- Valores dados: 1, 2, 5, 20, 8, 18, 105, 106, 108.
- Valores filtrados: 1, 2, 5, 12.5, 8, 18, 25.5, 33, 40.5.

El siguiente dato recibido por el sensor es 110, valor que está fuera de rango con respecto al último valor filtrado. Pero al llegar al apartado 2 del filtro, nos damos cuenta que está en rango con los 3 últimos datos dados por el sensor: 105, 106, 108. Por lo que el sistema realmente se encuentra en esa posición y el filtro acepta ese dato como cierto.

X_Sensor	X_Filtrado_0.5	X_Filtrado_0.75	X_Filtrado_1	Rango_Sup	Rango_Inf
1	1	1	1	0	0
2	2	2	2	11	-9
5	5	5	5	12	-8
20	10	12,5	15	15	-5
8	8	8	8	22,5	2,5
18	18	18	18	18	-2
105	23	25,5	28	28	8
106	28	33	38	35,5	15,5
108	33	40,5	48	43	23
110	110	110	110	50,5	30,5
112	112	112	112	120	100
109	109	109	109	119	102

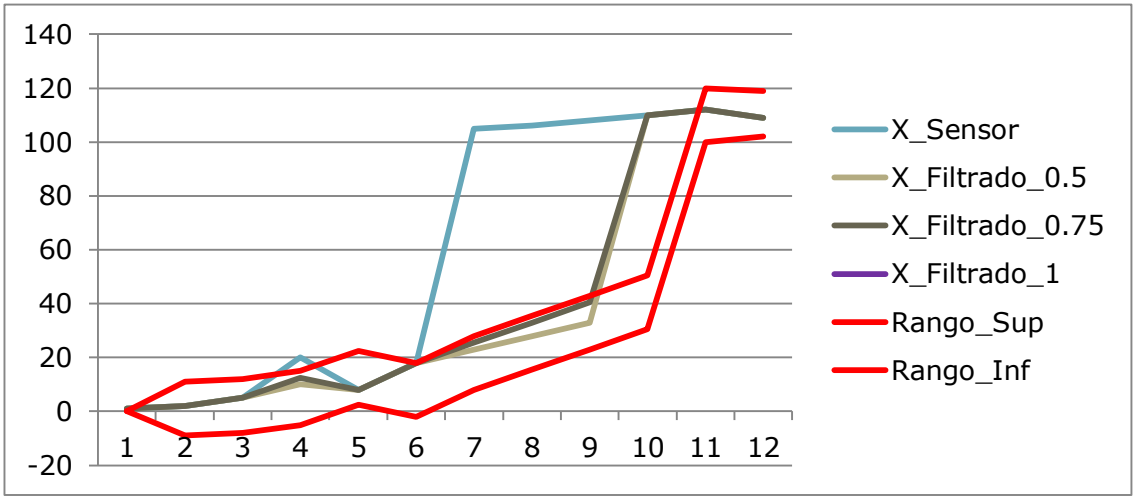


Tabla 1: Ejemplo Follower Filter

Podemos observar como cuando el valor dado por el sensor, no entra dentro del rango del valor anterior, el filtro toma el valor como falso atenuándolo en el sentido del error en función del valor asignado a la variable "fuerza".

Si el sistema varía bruscamente debido a alguna condición especial, el filtro tardará el número de ciclos establecido (en este caso 3) en aceptar la nueva situación.

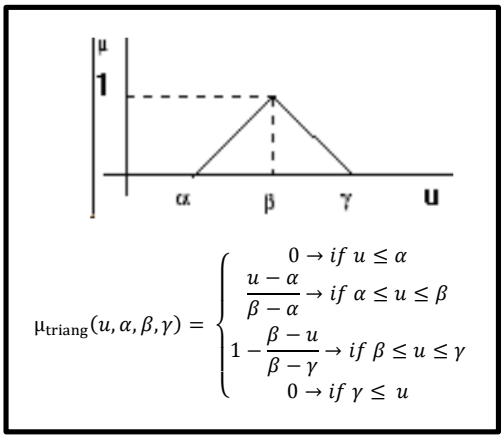
5 CONTROLADOR FUZZY

5.1 LÓGICA FUZZY

La lógica "fuzzy" o difusa es una técnica incluida en el concepto "Control Inteligente" que trata de expresar el conocimiento común expresado en términos lingüísticos y poco precisos a un lenguaje matemático que se rige por los conjuntos borrosos.

Estas variables lingüísticas son representadas por conjuntos borrosos, que a diferencia de los conjuntos precisos, no tienen un valor fijo sino uno dependiente de las llamadas funciones de pertenencia. Estas darán un valor entre 0 y 1 al conjunto borroso y pueden ser de multitud de formas. Aquí explicaremos las más básicas. [7]

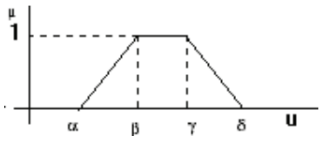
- Función triangular: Es la más simple de todas después de la línea recta. Es muy inestable y abrupta.



$$\mu_{\text{triang}}(u,\alpha,\beta,\gamma) = \begin{cases} 0 \rightarrow if\ u \leq \alpha \\ \dfrac{u-\alpha}{\beta-\alpha} \rightarrow if\ \alpha \leq u \leq \beta \\ 1 - \dfrac{\beta-u}{\beta-\gamma} \rightarrow if\ \beta \leq u \leq \gamma \\ 0 \rightarrow if\ \gamma \leq\ u \end{cases}$$

Ilustración 9: Función de pertenencia triangular

- Función trapezoidal: Es la evolución de la función triangular, incluye una zona de estabilidad, lo que la convierte en muy buena opción debido a que conserva su facilidad computacional.



$$\mu_{\text{trap}}(u,\alpha,\beta,\gamma,\delta) = \begin{cases} 0 \rightarrow if\ u \leq \alpha \\ \dfrac{u-\alpha}{\beta-\alpha} \rightarrow if\ \alpha \leq u \leq \beta \\ 1 \rightarrow if\ \beta \leq u \leq \gamma \\ 1 - \dfrac{\gamma-u}{\gamma-\delta} \rightarrow if\ \gamma \leq u \leq \delta \\ 0 \rightarrow if\ \delta \leq u \end{cases}$$

Ilustración 10: Función de pertenencia trapezoidal

- Función de campana de Gauss: Conserva la forma de la función trapezoidal añadiendo una suavidad que hace unos sistemas de control muy estables. En contraste, computacionalmente se hace más pesada.

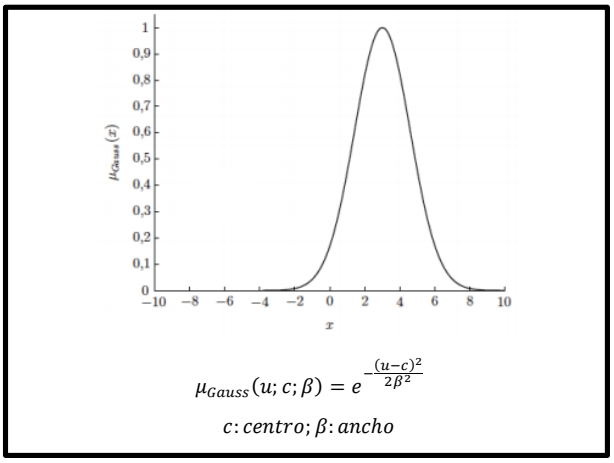


Ilustración 11: Función de transferencia de campana de Gauss

Las fases del control fuzzy serían las siguientes:

1) Fuzzificación: Se obtienen los datos del mundo real a través de sensores y se atribuyen a una variable lingüística definida por una función de pertenencia de entrada.
2) Inferencia borrosa: Siguiendo una tabla de reglas, se asigna una función de pertenencia de salida (en este trabajo se utilizarán valores en vez de funciones para facilitar los cálculos) y eligiendo el mínimo de las μ de entrada, se asigna la μ de salida.
3) Defuzzificación: Por último hay que convertir los datos borrosos en datos precisos. Existen diversos métodos matemáticos para esta tarea. Los más conocidos son el método de Mandami o Sugeno. En este proyecto usaremos el método de Mandami con la pequeña fórmula matemática: $output = \frac{\sum(\mu_{regla} * Valor_{regla})}{\sum \mu_{regla}}$ [8]

5.2 CONTROLADORES FUZZY VS CONTROLADORES PID

Actualmente, el control PID (Proporcional Integral Derivativo) está ampliamente extendido por el mundo en todo tipo de controladores industriales, debido a sus rápidos tiempos de respuesta y a un proceso de ajuste bastante simple. El problema es que llegados a cierto grado de complejidad el ajuste de estos controladores se vuelve extremadamente complicado, dejando aparte que no tiene la capacidad de responder ante no linealidades. [9] En este punto aparecen los controladores fuzzy, que al depender de variables lingüísticas para el control y no necesitar conocimientos matemáticos previos, hacen de su ajuste mucho más intuitivo. [10]

5.3 DIAGRAMA ARDRONE 2.0

Antes de analizar el control del dron, primero hemos de saber que entradas y salidas usaremos, y cual será la notación.

En primer lugar empezaremos analizando los ángulos que trataremos en este control, en primer lugar los obtendremos a través del giróscopo incorporado, aunque más tarde utilizaremos los que nos proporciona la cámara relacionándolos con las velocidades en las coordenadas x e y obtenidas por el IMU. Obviaremos el control de yaw en este trabajo:

- Pitch:

Ilustración 12: Notación ARDrone 2.0 pitch

- Roll:

Ilustración 13: Notación ARDrone 2.0 roll

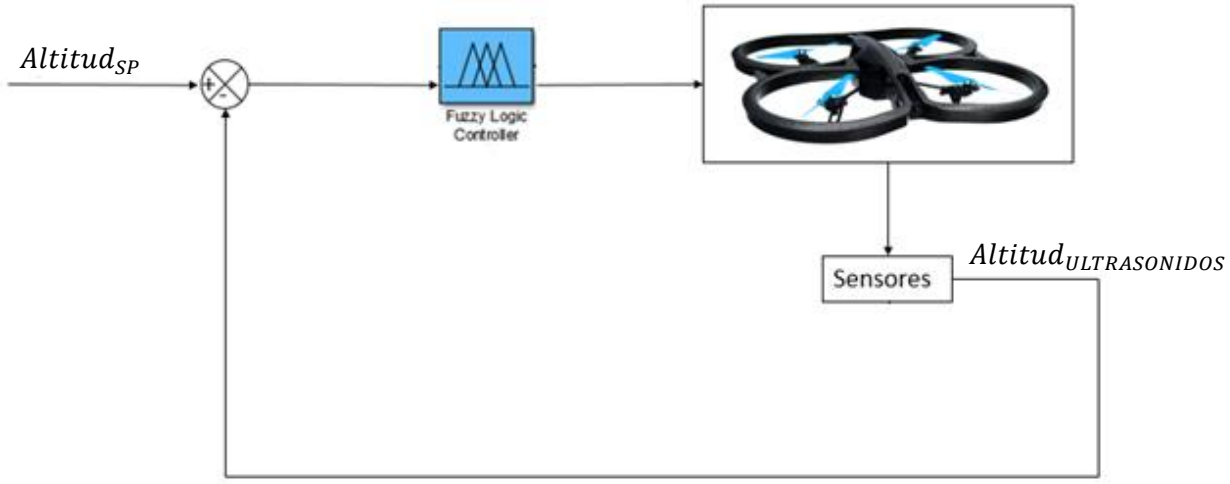
Obtendremos la altitud de un sensor de ultrasonidos ubicado en la parte inferior del dron.

- Altitud:

Ilustración 14: Notación ARDrone 2.0 altitude

5.4 CONTROLADOR DE ALTITUD

En todos los casos, se utiliza el mismo controlador de altitud. Este es un controlador muy sencillo basado en funciones de pertenencia **campanas de Gauss** para dotar a los cambios de altura de una suavidad indiscutible.



Podremos encontrar el código asociado en el Anexo D.

Ahora vamos a enseñar las funciones de pertenencia de entrada y salida del controlador, así como la tabla de reglas que lo rige y su superficie de control.

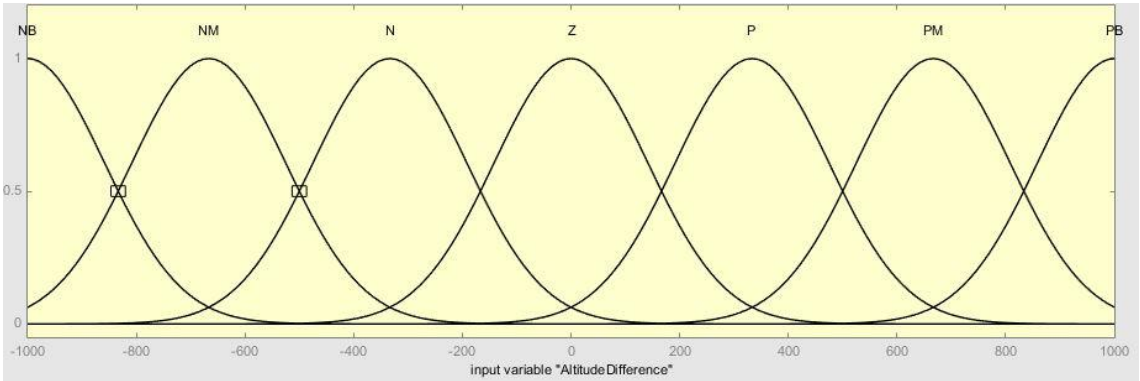


Ilustración 15: Funciones de pertenencia entrada control de altitud

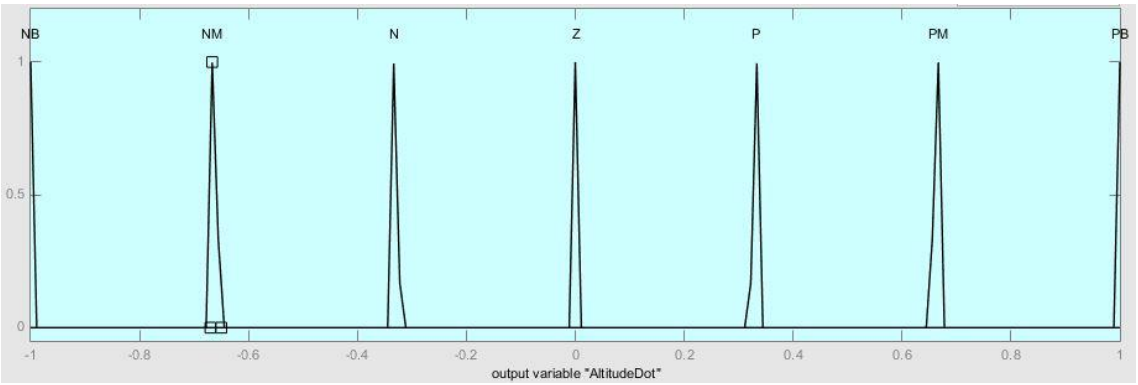


Ilustración 16: Funciones de pertenencia salida control de altitud

Altitud	*NB*	*NM*	*N*	*Z*	*P*	*PM*	*PB*
Output	NB	NM	N	Z	P	PM	PB

Tabla 2: Tabla de reglas de control de altitud

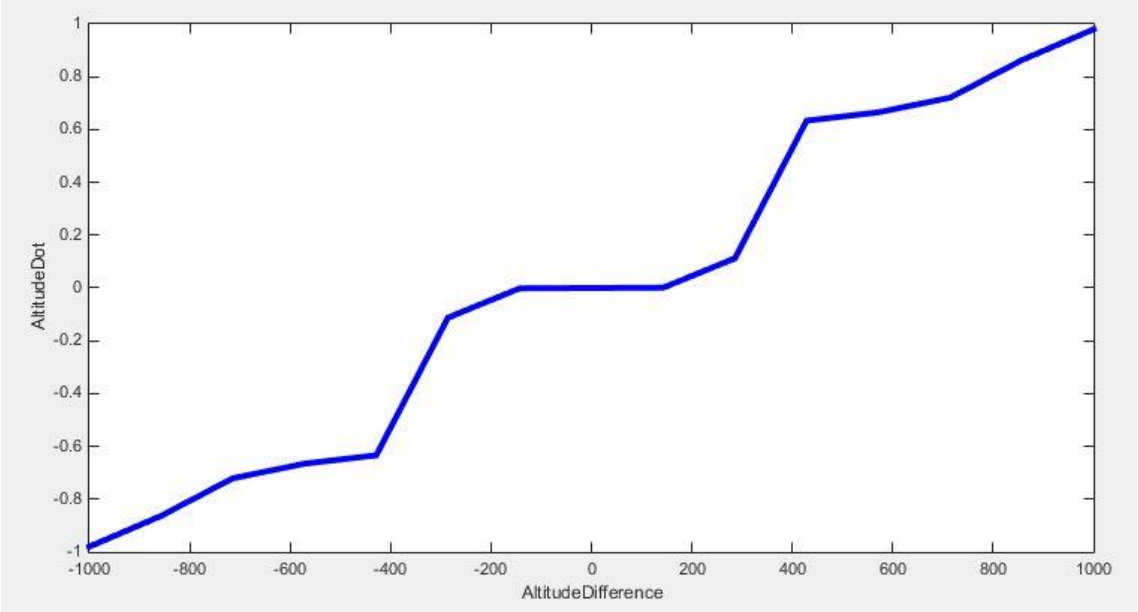


Ilustración 17: Superficie control de altitud

Aprovechando su sencillez, realizaremos un ejemplo práctico para mostrar el funcionamiento del filtro. Imaginemos que el sensor de ultrasonidos devuelve un valor de 600 mm. La altura a la que queremos llegar es de 1000mm, por lo que la entrada del sensor será: [1000 – 600 = 400].

Si miramos las variables lingüísticas de las entradas y la tabla de reglas observamos que entra en los siguientes conjuntos borrosos:

- Si Δ Altitud es PM, entonces PM
- Si Δ Altitud es P, entonces P
- Si Δ Altitud es Z, entonces Z

Ahora calculamos los distintos μ que se obtiene de cada conjunto.

- $\mu_{PM_{Input}} = e^{-\frac{(400-666.7)^2}{141.6^2}} = 0.0288$
- $\mu_{P_{Input}} = e^{-\frac{(400-333.3)^2}{141.6^2}} = 0.801$
- $\mu_{Z_{Input}} = e^{-\frac{(400-0)^2}{141.6^2}} = 0.00034$

Ahora calculamos la μ de las salidas:

- $\mu_{PM_{Output}} = \min\left(\mu_{PM\,Input}\right) = 0.0288$
- $\mu_{P_{Output}} = \min\left(\mu_{P_{Input}}\right) = 0.801$
- $\mu_{Z_{Input}} = \min\left(\mu_{Z_{Input}}\right) = 0.00034$

Ahora sabiendo los valores de salida establecidos que aparecen en las reglas:

- PM: 0.66
- P: 0.33

- Z: 0

Ahora aplicamos la ecuación:

$$Salida = \frac{0.0288 * 0.66 + 0.801 * 0.33 + 0.00034 * 0}{0.0288 + 0.801 + 0.00034} = 0.34$$

Podemos apreciar que como el que mayor μ de salida tenía era la regla de Positive, es el valor que más se ha acercado mientras que el resto ha servido para matizar.

5.5 CONTROLADOR DE ESTABILIDAD

Podemos comprobar que el controlador "fuzzy" sigue un esquema de controlador con realimentación negativa convencional. En primer lugar se desarrollará el controlador de estabilidad, que intentará mantener todos los ángulos giroscópicos (pitch, roll, yaw) a cero, teniendo en cuenta la velocidad en los ejes "X" e "Y" para intentar mantener una posición constante aunque sin referencia posicional.

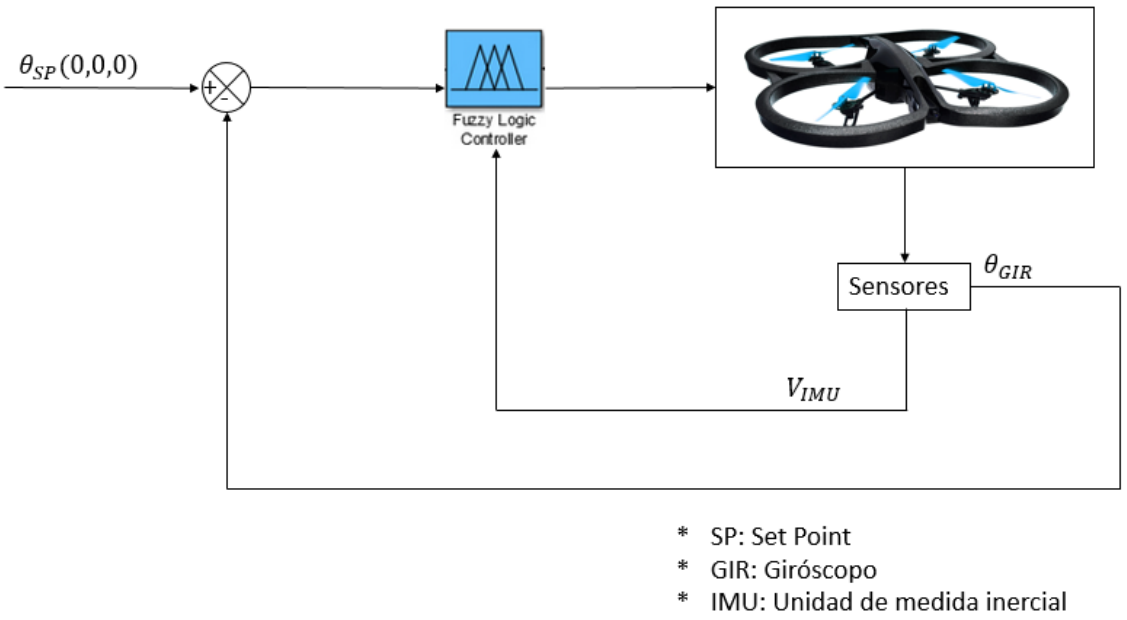


Ilustración 18: Esquema de control de estabilidad

Como funciones de pertenencia en este sistema usaremos trapecios, los cuales computacionalmente son muy asequibles y permiten un control lo

suficientemente suave para ser aptos en el control del cuadricóptero. Usaremos valores puntuales como "output" para facilitar el cálculo. Habrá que explicar por separado el controlador de pitch y el controlador de roll, aunque solo se diferencian en el esquema de reglas. Esto es debido a sus notaciones (cuando pitch es positivo Vx es negativo, mientras que cuando roll es positivo Vy es positivo). Se puede ver el código en el Anexo B.

5.5.1 PITCH

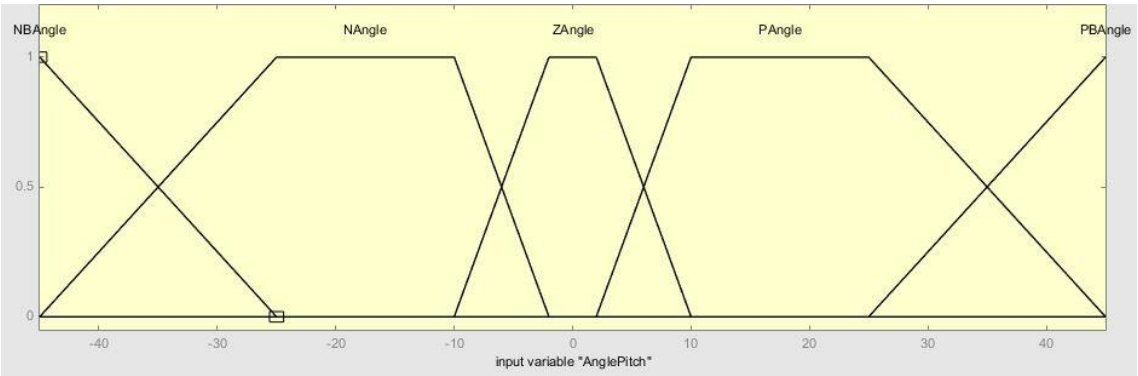


Ilustración 19: Funciones de pertenencia de input ángulo de pitch control de estabilidad

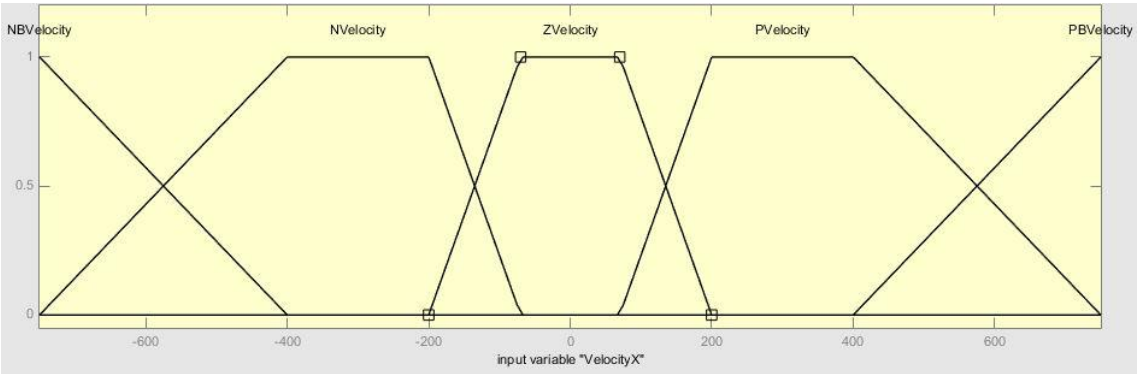


Ilustración 20: Funciones de pertenencia de input velocidad de pitch control de estabilidad

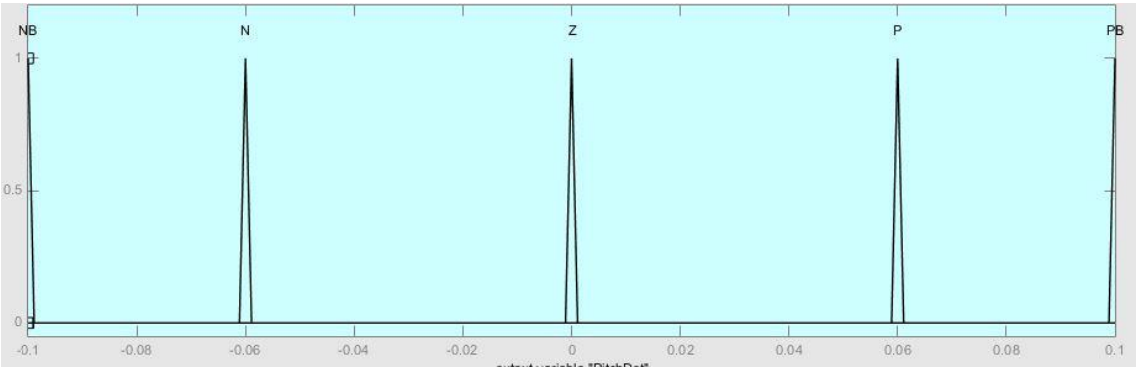


Ilustración 21: Función de pertenencia de output de control de estabilidad

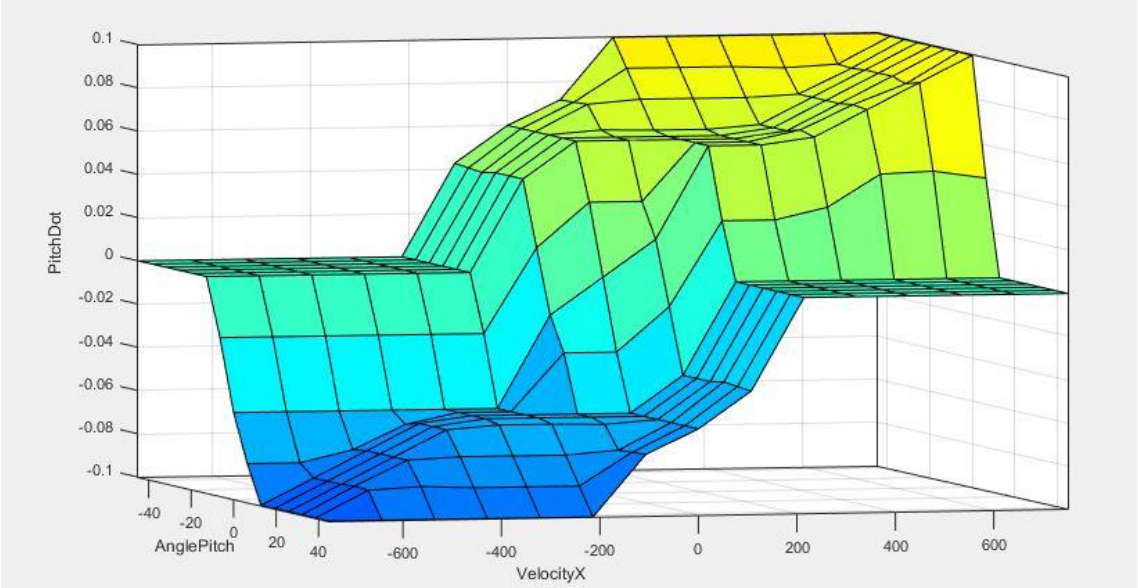


Ilustración 22: Superficie pitch sistema de control de estabilidad

Vx\θ	*NB*	*N*	*Z*	*P*	*PB*
NB	Z	Z	NB	NB	NB
N	Z	Z	N	N	NB
Z	P	P	Z	N	NB
P	PB	P	P	Z	Z
PB	PB	PB	PB	Z	Z

Tabla 3: Tabla de reglas pitch control de estabilidad

5.5.2 ROLL

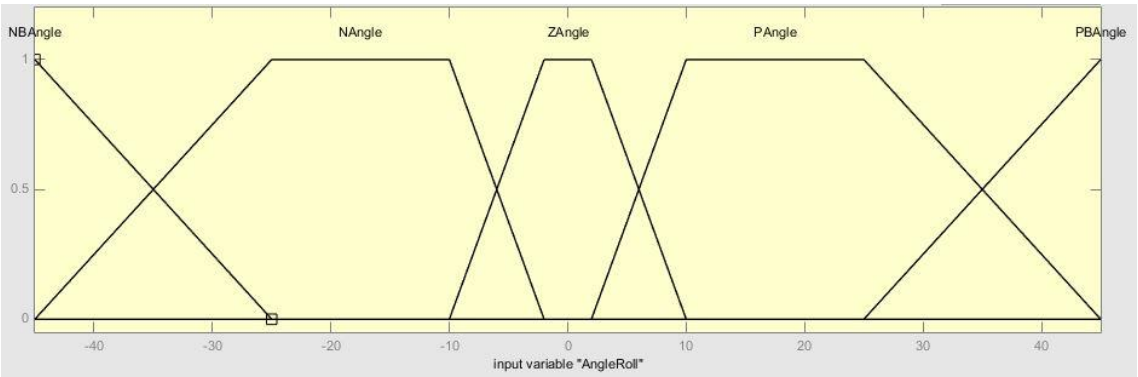


Ilustración 23: Funciones de pertenencia input ángulo roll control de estabilidad

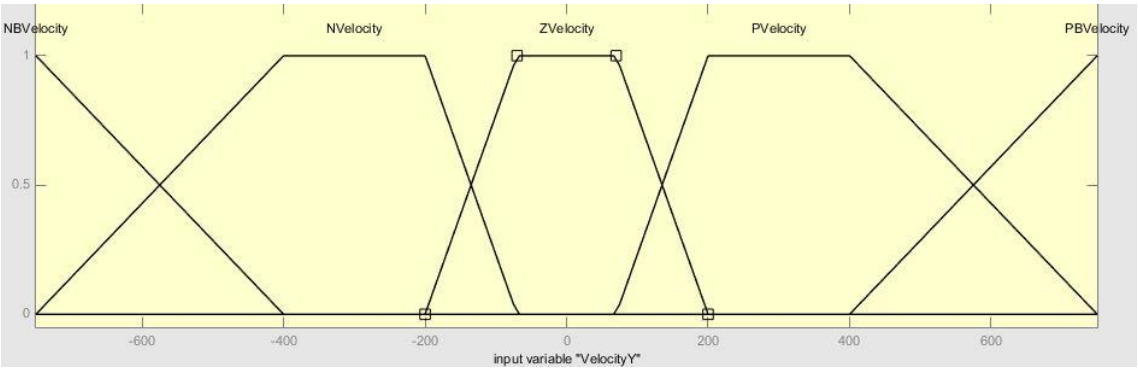


Ilustración 24: Funciones de pertenencia input velocidad roll control de estabilidad

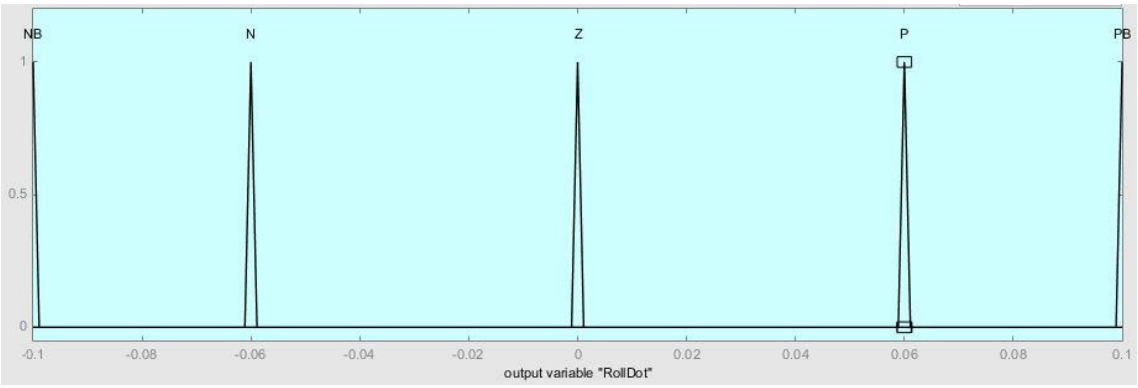


Ilustración 25: Funciones de perteneciente output control de estabilidad

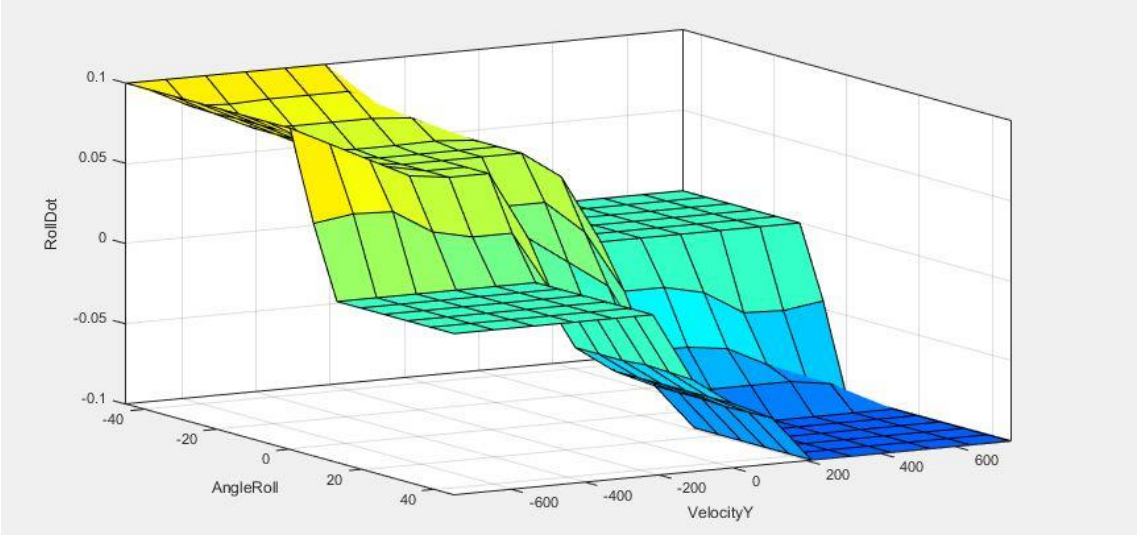


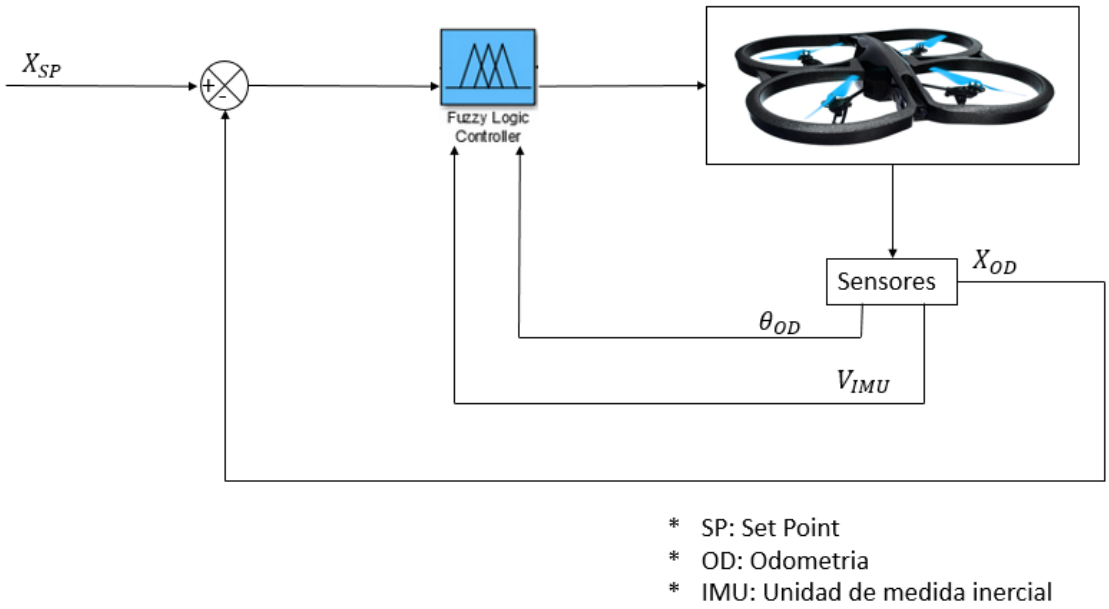
Ilustración 26: Superficie roll del sistema de control de estabilidad

Vy\ϕ	*NB*	*N*	*Z*	*P*	*PB*
NB	PB	PB	PB	Z	Z
N	PB	P	P	Z	Z
Z	P	P	Z	N	N
P	Z	Z	N	N	NB
PB	Z	Z	NB	NB	NB

Tabla 4: Tabla de reglas roll control de estabilidad

5.6 CONTROLADOR DE POSICIÓN

Tras desarrollar el sistema de estabilidad, nos centramos en algo más complejo como es el de posición. Éste lleva incluido el propio sistema de estabilidad anteriormente desarrollado, sólo que ampliado para un control de posición. Si observamos, el sistema de estabilidad se aprecia cuando no hay diferencia de posición.



* SP: Set Point
* OD: Odometria
* IMU: Unidad de medida inercial

Ilustración 27: Esquema de control de posición

En este caso utilizaremos las mismas funciones de pertenencia que en el control de estabilidad, solo que añadiremos una variable más que será la distancia. Esta variable será sencilla y solo indicará en cada eje si está a un sentido o al otro de un rango de tolerancia.

Se podrá ver el código en el Anexo C.

5.6.1 PITCH

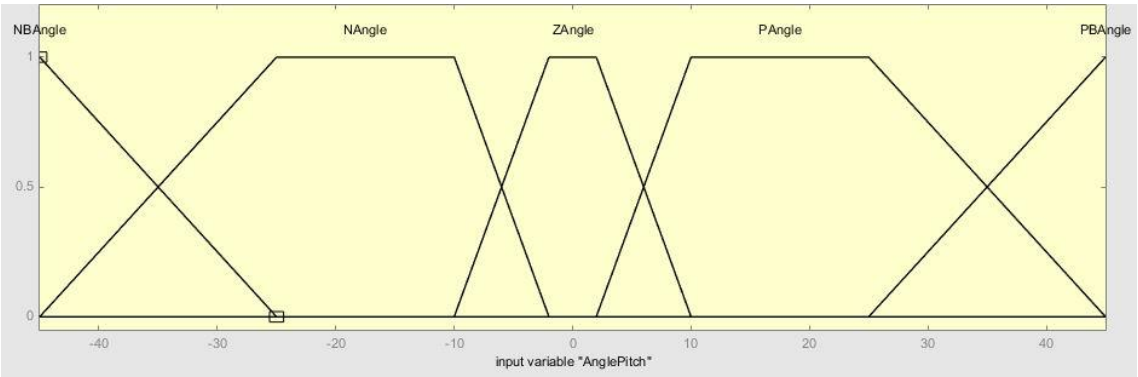


Ilustración 28: Funciones de pertenencia de input ángulo de pitch control de posición

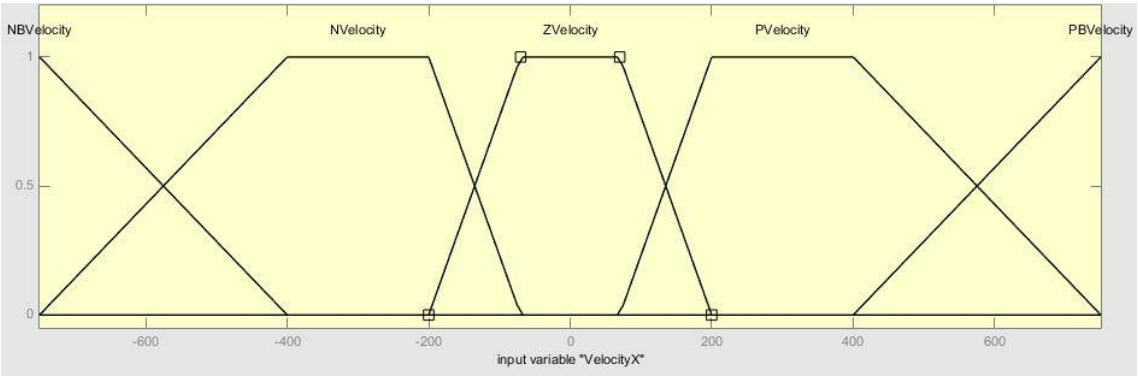


Ilustración 29: Funciones de pertenencia de input velocidad de pitch control de posición

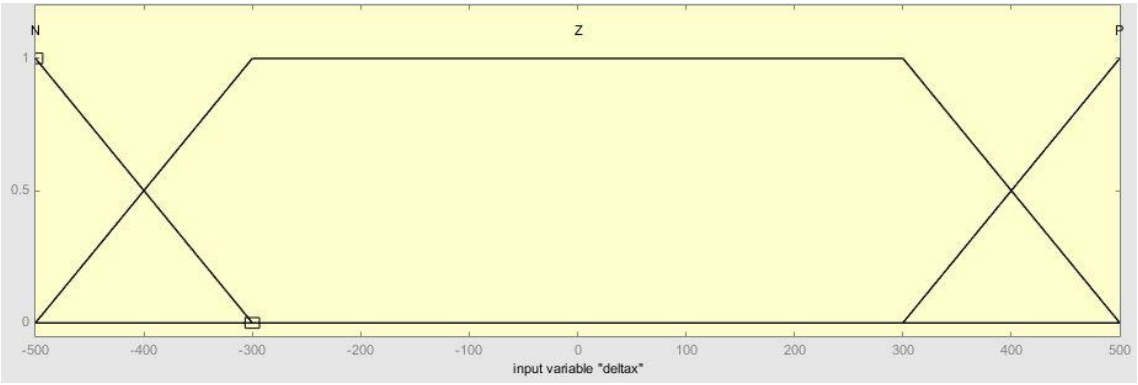


Ilustración 30: Función de pertenencia de input delta x

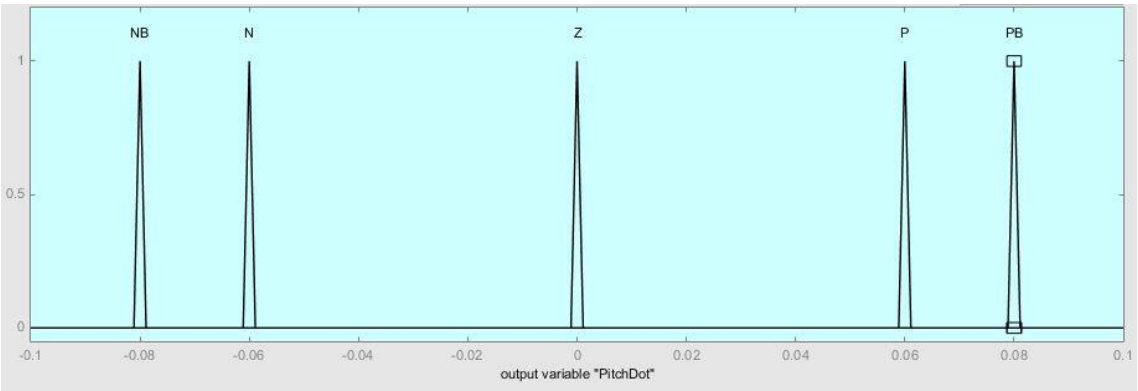


Ilustración 31: Funciones de pertenencia de output de pitch en control de posición

$\Delta x < 0$

Vx\θ	*NB*	*N*	*Z*	*P*	*PB*
NB	Z	Z	N	N	NB
N	P	P	Z	N	N
Z	P	P	P	Z	N
P	PB	PB	P	Z	N
PB	PB	PB	PB	Z	Z

Tabla 5: Tabla de reglas pitch control de posición 1

$\Delta x = 0$

Vx\θ	*NB*	*N*	*Z*	*P*	*PB*
NB	Z	Z	NB	NB	NB
N	Z	Z	N	N	NB
Z	P	P	Z	N	NB
P	PB	P	P	Z	Z
PB	PB	PB	PB	Z	Z

Tabla 6: Tabla de reglas pitch control de posición 2

$\Delta x > 0$

Vx\θ	*NB*	*N*	*Z*	*P*	*PB*
NB	Z	Z	NB	NB	NB
N	P	Z	N	NB	NB
Z	P	Z	N	N	N
P	P	P	Z	N	N
PB	PB	P	P	Z	Z

Tabla 7: Tabla de reglas pitch control de posición 3

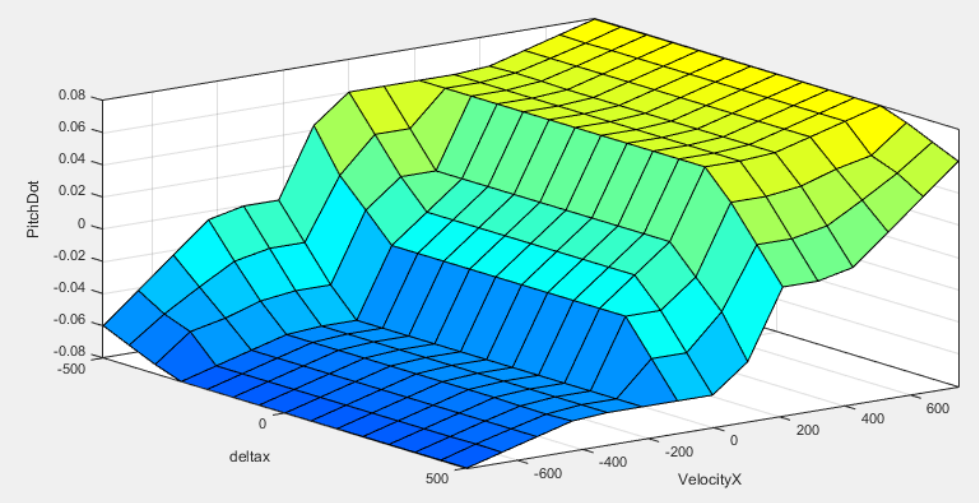


Ilustración 32: Superficie pitch control de posición 1

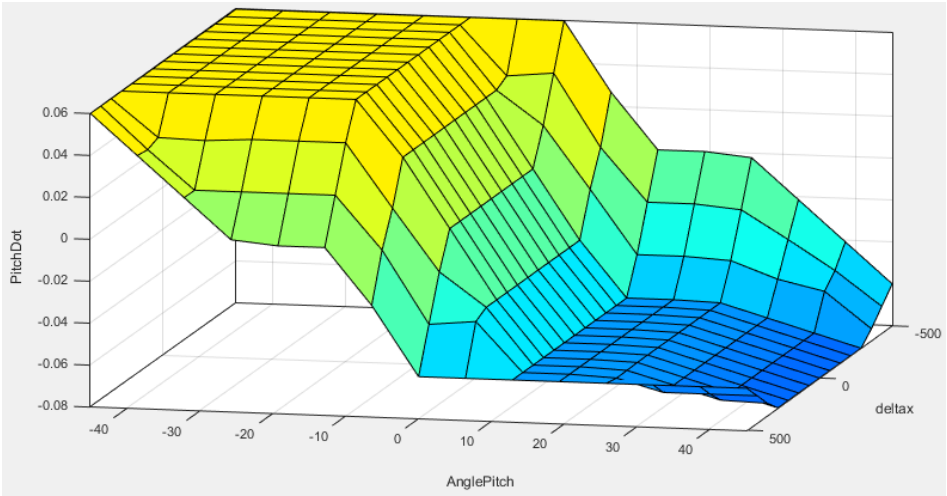


Ilustración 33: Superficie pitch control de posición 2

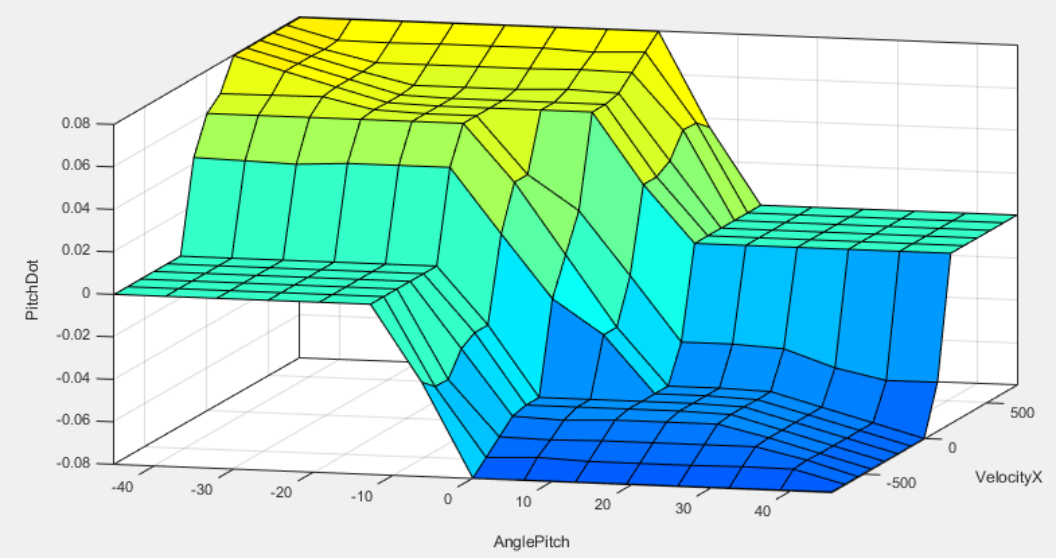


Ilustración 34: Superficie pitch control de posición 3

5.6.2 ROLL

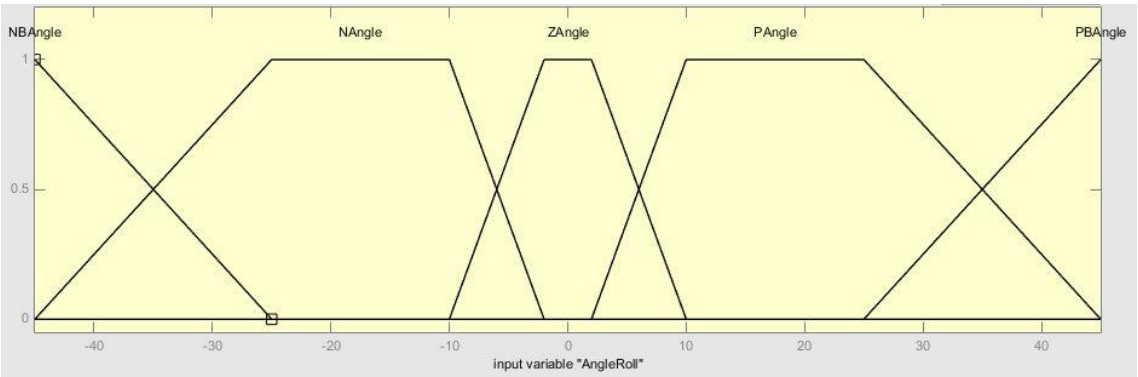


Ilustración 35: Funciones de pertenencia input ángulo roll control de estabilidad

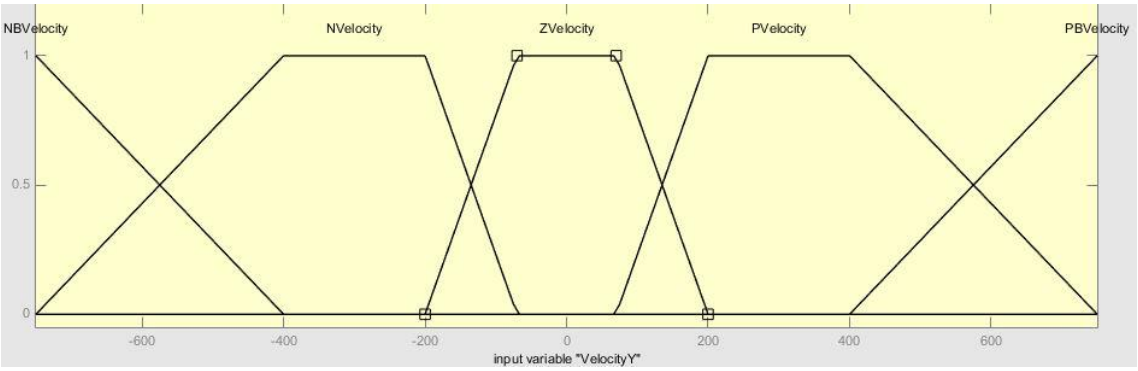


Ilustración 36: Funciones de pertenencia input velocidad roll control de estabilidad

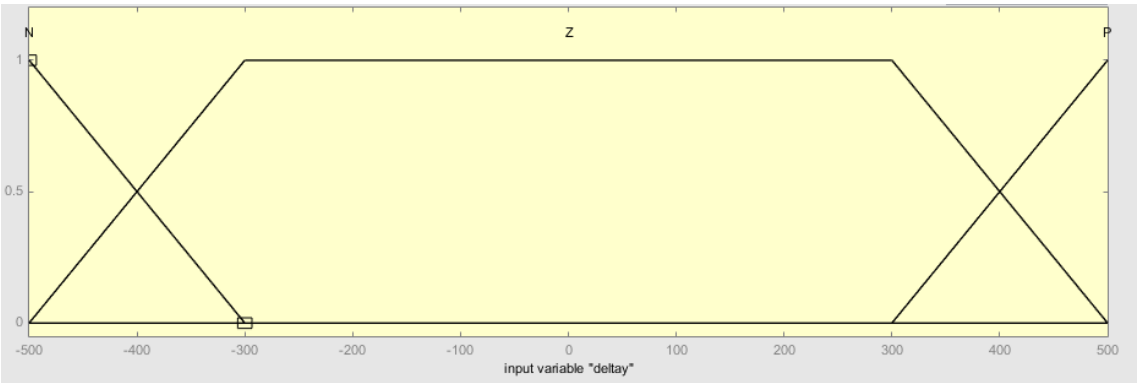


Ilustración 37: Funciones de pertenencia input delta y control de estabilidad

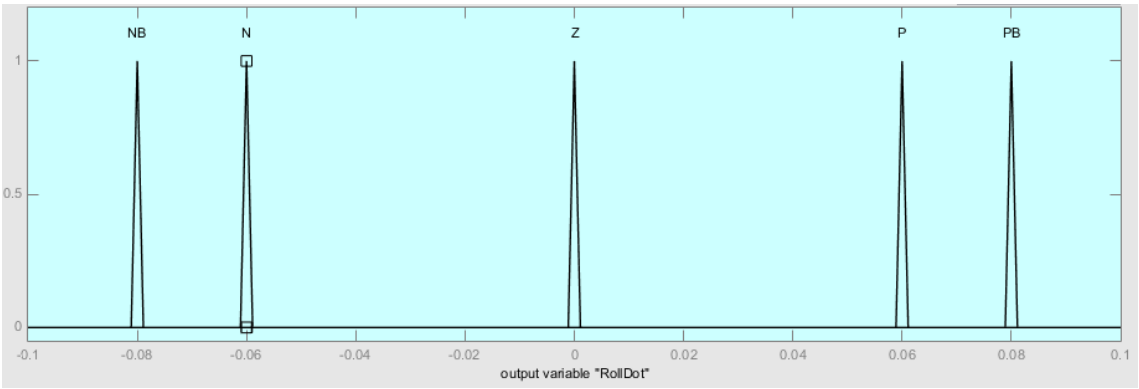


Ilustración 38: Funciones de pertenencia output control de estabilidad

Δ y < 0

Vy\ϕ	*NB*	*N*	*Z*	*P*	*PB*
NB	PB	P	P	Z	Z
N	P	P	Z	N	N
Z	P	Z	N	N	N
P	P	Z	N	NB	NB
PB	Z	Z	NB	NB	NB

Tabla 8: Tabla de reglas pitch control de posición 1

Δ y = 0

Vy\ϕ	*NB*	*N*	*Z*	*P*	*PB*
NB	PB	PB	PB	Z	Z
N	PB	P	P	Z	Z
Z	P	P	Z	N	N
P	Z	Z	N	NB	NB
PB	Z	Z	NB	NB	NB

Tabla 9: Tabla de reglas pitch control de posición 2

Δ y > 0

Vy\ϕ	*NB*	*N*	*Z*	*P*	*PB*
NB	PB	PB	PB	Z	Z
N	PB	PB	P	Z	N
Z	P	P	P	Z	N
P	P	P	Z	N	N
PB	Z	Z	N	N	NB

Tabla 10: Tabla de reglas pitch control de posición 3

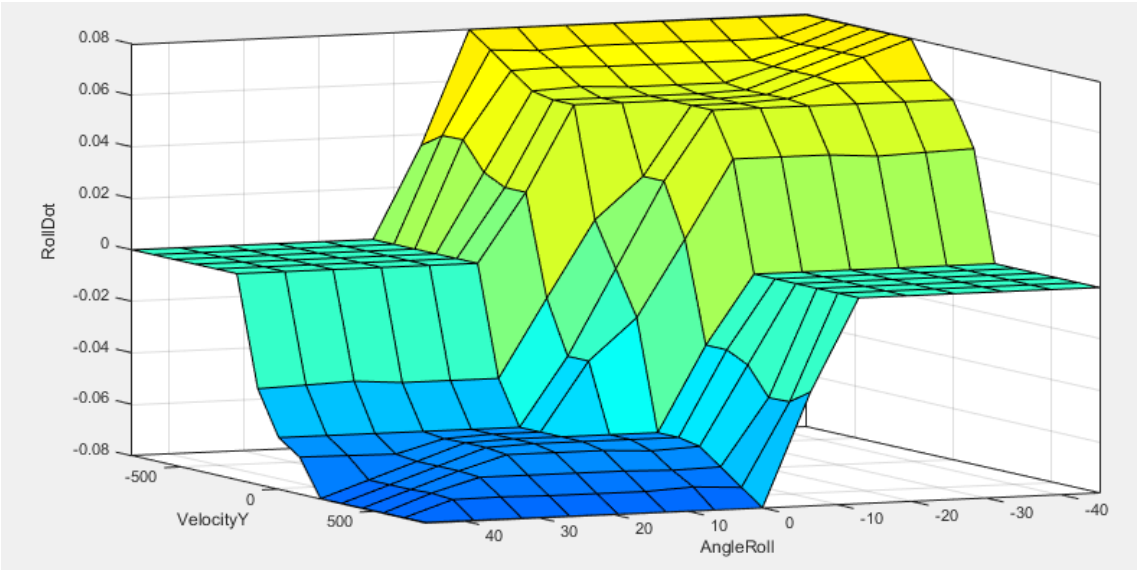


Ilustración 39: Superficie de control de estabilidad 1

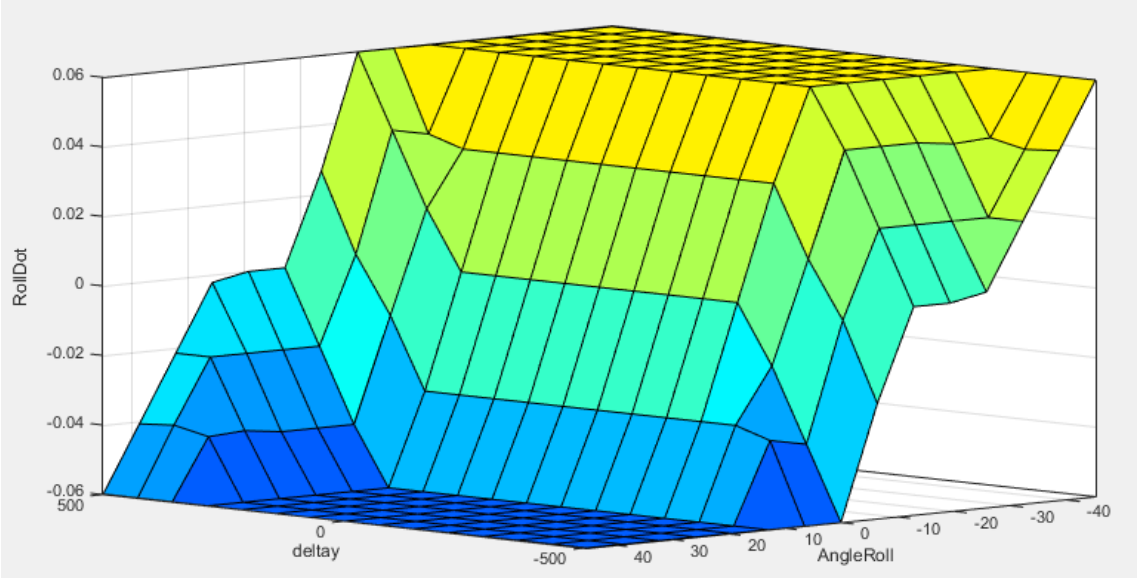


Ilustración 40: Superficie de control de estabilidad 2

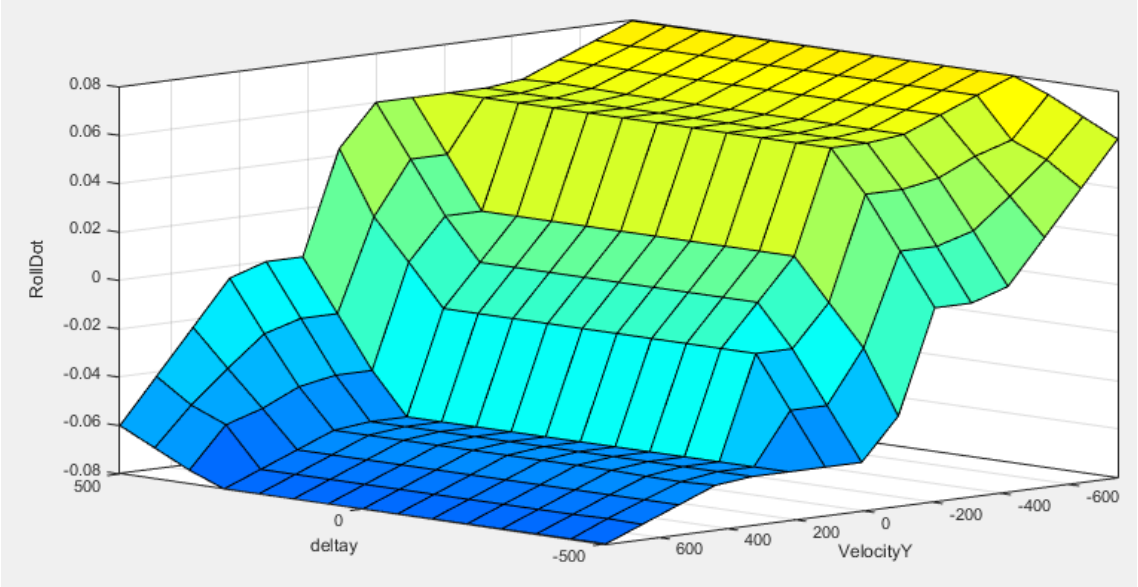


Ilustración 41: Superficie de control de estabilidad 3

6 INTEGRACIÓN DE LOS ELEMENTOS EN EL PROGRAMA

6.1 CLASE INTELLIGENT_TOOLS

Para poder facilitar la integración de las herramientas desarrolladas en este proyecto dentro del proyecto del laboratorio de sistemas inteligentes, se ha creado una clase que incorpora, aparte de todos los filtros y controladores, funciones complementarias para facilitar el trabajo.

6.1.1 ESTRUCTURA DRONEPROPERTIES

Es una estructura que almacenará las propiedades que puedan ser de utilidad sobre el dron, para tenerlas centralizadas y poder acceder a ellas y modificarlas de una forma más simple.

```
public struct DroneProperties
    {
        public float Yaw;
        public float Pitch;
        public float Roll;
        public float Altitude;
        public float Vx;
        public float Vy;
        public float X;
        public float Y;

    }
```

6.1.2 ESTRUCTURA STOREDPOSITION

Esta estructura se utiliza para almacenar el punto al que el controlador de posición se ha de ubicar, actualizándolo cuando haga falta.

```
public struct StoredPosition
    {
        public float x;
        public float y;

    }
```

6.1.3 ESTRUCTURA FUZZY OUTPUT

Se utiliza esta estructura para tener en una variable las salidas que tienen que tener el sistema de control de estabilidad y de posición. Por otro lado hay unas variables booleanas, que permiten en cada ciclo del control, saber que variables han sido utilizadas. Aunque finalmente no se han usado en este proyecto.

```
public struct FuzzyOutput
    {
        public float PB;
        public bool PBbool;
        public float P;
        public bool Pbool;
        public float Z;
        public bool Zbool;
        public float N;
        public bool Nbool;
        public float NB;
        public bool NBbool;
        public float Value;  //Value will represent input and output
   }
```

6.1.4 FUNCIÓN GAUSSBELL

Utilizamos esta función para facilitar el uso de las campanas de Gauss en el controlador de altitud.

```
  private static float Gaussbell(float value, float center, float width)
        {
            float probability;
            float e = (float)Math.E;
probability = (float)Math.Pow(e, (-Math.Pow((value - center), 2) / (2 *
Math.Pow(width, 2))));
            return probability;

        }
```

7 EXPERIMENTOS Y RESULTADOS

7.1 FILTRO

Para poder calibrar el filtro, primero hubo que analizar los datos obtenidos del dron en vuelo, para ello usaremos la función "Navigate" desarrollada en LSIDrone.

Ilustración 42: Gráfica variación absoluta pitch

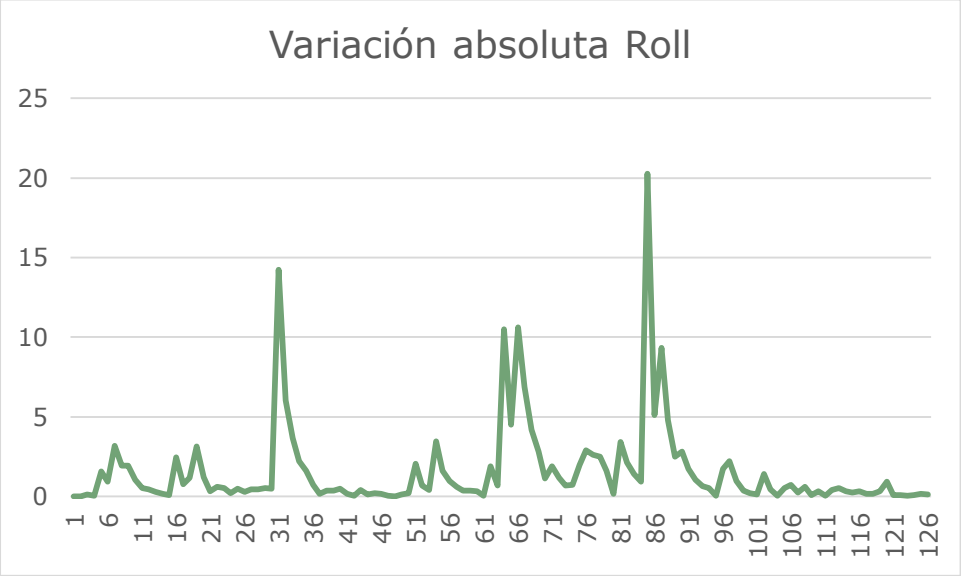


Ilustración 43: Gráficca variación absoluta roll

Observando las variaciones en valor absoluto de los grados, observamos que en pitch son 35 grados, mientras que en roll ha sido detectado 20 grados. Como utilizaremos el mismo filltro para ambos, decidiremos que **la máxima variación en los ángulos será de 25 grados**.

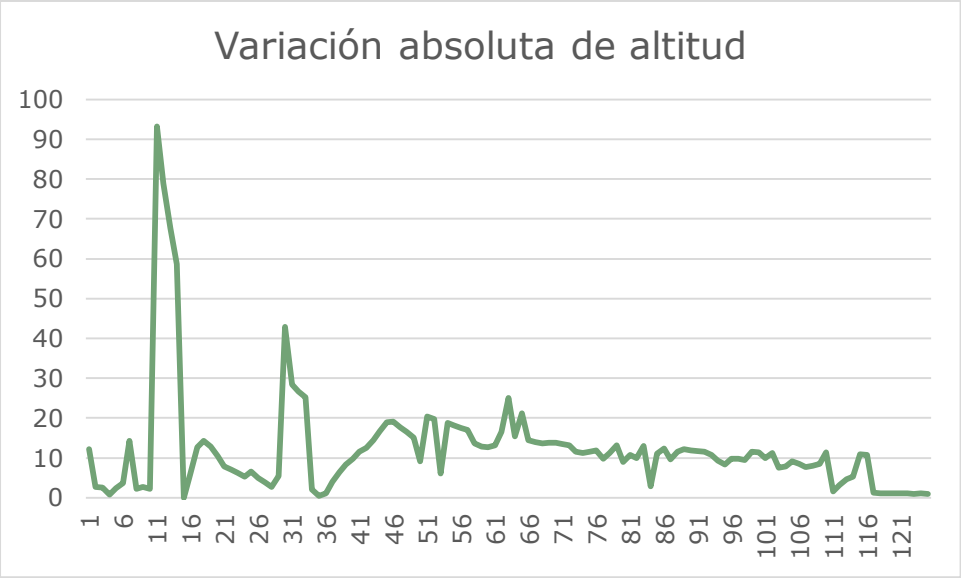


Ilustración 44: Gráfica variación absoluta altitud

Si observamos la variación absoluta de altitud, y descartamos el pico de 90 mm, **observamos que una variación plausible sería 40 mm**.

Ilustración 45: Gráfica variación absoluta posición x

Ilustración 46: Gráfica variación absoluta de posición y

Por último, vemos que la variación de la posición máxima en ambas coordenadas se encuentra entre 350 y 400 mm. Por lo que **usaremos 350 mm como rango**.

7.2 CONTROLADOR

Hubo que hacer varias pruebas utilizando los sistemas de control para poder definir correctamente los valore de salida, así como sus rangos.

7.2.1 RESULTADOS EXPERIMENTOS CONTROLADOR DE POSICIÓN

Primero probamos a utilizar como salidas: [-1,-0.6, 0, 0.6, 1], pero como podemos observar en la ilustración era demasiado fuerte y oscilaba hasta 1000 mm.

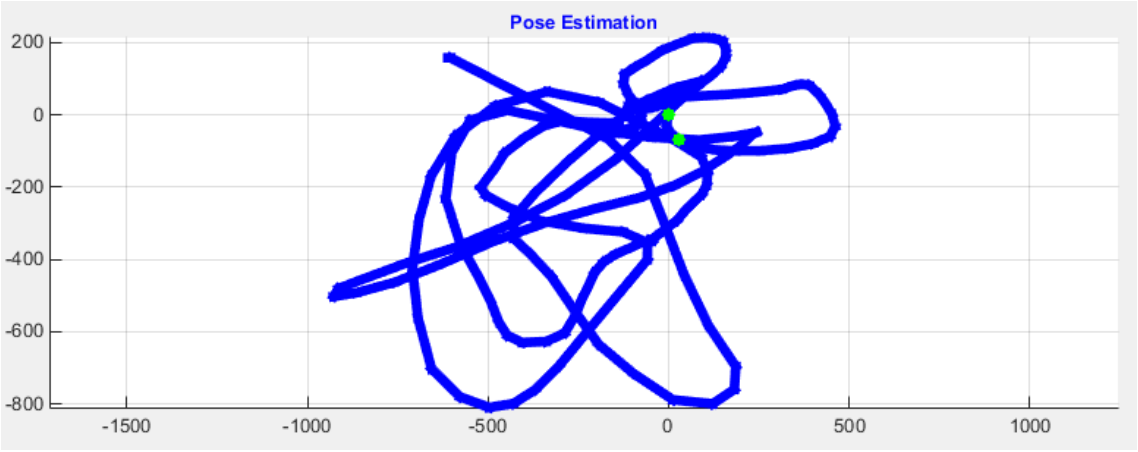


Ilustración 47: Trayectoria dron 1

Tras eso, probamos a reducir la salida máxima de 1 a 0,8 y vemos que el resultado es muy satisfactorio, teniendo en cuenta el tamaño del dron.

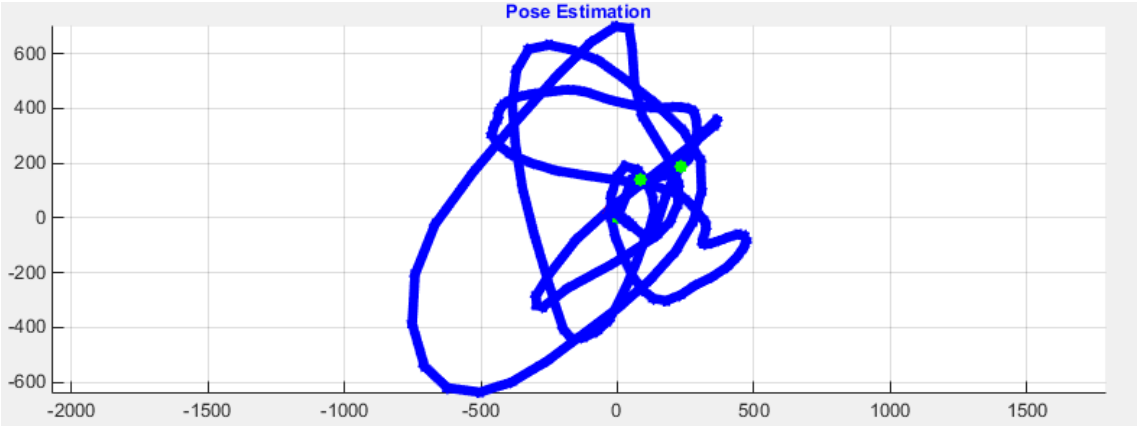


Ilustración 48: Trayectoria dron 2

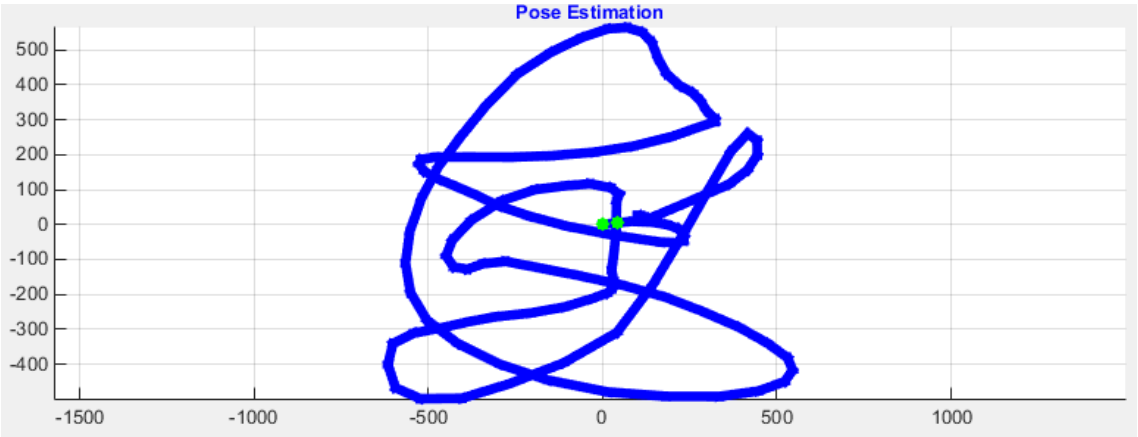


Ilustración 49: Trayectoria dron 3

8 CONCLUSIONES Y TRABAJOS FUTUROS

8.1 CONCLUSIONES

Tras desarrollar este proyecto, podemos llegar a la conclusión de que la lógica borrosa puede ser una herramienta extremádamente útil y potente cuando desconocemos datos matemáticos exactos de los sistemas a controlar o cuando lo que queremos transladar es el funcionamiento de un sistema de una forma lógica con términos lingüísticos a un controlador matemático.

En este caso se ha desarrollado un sistema de control de posición y estabilidad de un dron, del cual se desconoce su función de transferencia. Si es verdad, que es menos preciso que un PID convencional, pero podemos deducir que al no haber necesidad de linealizar el sistema, es más estable en cuanto a variaciones del entorno.

Este tipo de controladores puede resultar muy útil para controlar entornos con muchas variables que no se pueden controlar o muy no lineales.

8.2 TRABAJOS FUTUROS

Tras desarrollar el controlador de posición basado en lógica fuzzy, existen dos campos fundamentales en los que se puede trabajar.

Por un lado, se puede trabajar en el propio controlador. En este trabajo, a base de prueba y error se han obtenido unos valores de "output" para introducirlos en la función "Navigate" entre -1 y 1. Como trabajo futuro en este ámbito se podría desarrollar un algoritmo de optimización que fuese capaz de dependiendo del dron y la situación varíe los valores de salida adaptándose al entorno para mejorar la calidad del control.

Por otro lado, se puede trabajar utilizando el controlador en el dron. Se podría generar otro código que vaya trasladando el punto de referencia según alguna necesidad: perseguir un objetivo, o detectar algo. Eso hará que el dron vaya siguiendo lo que puede ser una aplicación muy útil.

9 PRESUPUESTO

Código	Uds. Medida	Descripción	Cantidad	Precio/Ud. [€]	Importe [€]
9.1 COSTES MATERIALES					
9.1.1	Uds.	ArDrone 2.0 Vehículo aéreo no tripulado, 4 rotores eléctricos, desarrollado y distribuido por Parrot.	1	312,55	312,55
9.1.2	Uds.	Ordenador personal Procesador Intel i5, Memoria RAM 8GB, Disco duro 500GB, Windows 8.1, conexión WiFi. Instalado y funcionando.	1	257,00	257,00
9.1.3	Uds.	Microsoft Visual Studio Software de programación, acepta lenguaje C#, licencia gratuita para uso universitario	1	0,00	0,00
				SUBTOTAL COSTES MATERIAL	**569,55**
9.2 COSTES PERSONAL					
9.2.1	Meses	Ingeniero Industrial Ingeniero Electrónico Industrial, conocimientos de control, visión por computador, programación en C#, lógica Fuzzy	5	1500,00	7500,00
				SUBTOTAL COSTES PERSONAL	**7500,00**
				TOTAL PROYECTO	**8069,55**

Tabla 11: Presupuesto proyecto

ANEXOS

A. ANEXO: CÓDIGO "FOLLOWER FILTER" APLICADO A LOS ÁNGULOS

```
public static float AngleFollowerFilter(List<float> FilteredData, List<float>
NonFilteredData, float LastDataSensor)
        {

//////////////////////////////////////RANGES/////////////////////////////////////
///////

            //Range is a variable which limit the absolute value between
LastDataSensor and a fact of the lists
            int Range = 10;
            //FollowerRange is the addition number you sum or substract from the
last FilteredData fact when LastDataSensor doesn't pass the
            //specifications to be accepted in the system as a valid and real
fact.
            float FollowerRange = (float)0.5 * Range;
            //NumDataMedian set the number of data which is used by the filter,
number between 1 and 10, it has to be less than FilterRange
            //in main program (at first 7)
            int NumData = 5;

////////////////////////////////////////////////////////////////////////////////
////////



//////////////////////////////////////PROGRAM/////////////////////////////////////
////////

            //In case it is the first fact in the list, it is put back directly
            if (FilteredData.Count == 0)
            {
                return (LastDataSensor);
            }


            //In other case, we start the analysis
            else
            {
                //Firstly we generate the absolute value between LastDataSensor
and last fact in FilteredData list
                float abs = LastDataSensor - FilteredData[FilteredData.Count -
1];
                if (abs < 0) abs = abs * -1;
                //Afterwards, we compare absolute value with the Range
established. If it is in the range, we accept the LastDataSensor as
                //a valid value.
                if (abs < Range)
                {
                    return LastDataSensor;
                }
                //In other case, we continue filtering.
                else
                {
                    //Now we are analysing the last dynamic of the data, it helps
if the system has changed its state sharply in a recent
```

```
                    //period of time. We use a variable called count, which value
depends if the last NumDataMedian facts are in the range
                    //but in a different value out of range of the last value
accepted.
                    int count = 0;
                    for (int i = NonFilteredData.Count - 1; i >
NonFilteredData.Count - NumData - 1; i--)
                    {
                        if (i < 0) break;
                        //Now we calculate again the absolute value, but in this
case it is between LastDataSensor and last fact of
                        // NonFilteredData list
                        abs = NonFilteredData[i] - LastDataSensor;
                        if (abs < 0) abs = abs * -1;
                        //And we compare it with the range, if it is in range, we
add 1 to count
                        if (abs < Range) count += 1;
                    }
                    //if count is NumData means that NumData facts in the list
was in range, so we accept the change in the
                    //system
                    if (count == NumData)
                    {
                        return (LastDataSensor);
                    }
                    //If it is not in range, then the fact is a wrong value and
we have to filtered with the median filter
                    else
                    {
                        //Eventually, depending on the number of facts from
MedianList list, we choose the medium value
                        if (LastDataSensor - FilteredData[FilteredData.Count - 1]
> 0)
                        {
                            return (FilteredData[FilteredData.Count - 1] +
FollowerRange);
                        }
                        else
                        {
                            return (FilteredData[FilteredData.Count - 1] -
FollowerRange);
                        }
                    }
                }
            }


////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////

        }
```

B. ANEXO: CÓDIGO CONTROL ESTABILIDAD PITCH FUZZY

```
public static FuzzyOutput PitchStabilityController(FuzzyOutput Fuzzy, float vel,
float angle)
        {

            float NBvel = 0, Nvel = 0, Zvel = 0, Pvel = 0,
            PBvel = 0, NBangle = 0, Nangle = 0, Zangle = 0, Pangle = 0, PBangle =
0;

            // Velocity inputs MF to get probability
            if (vel <= -750)
                NBvel = 1;
            if (vel > -750 && vel <= -400)
                NBvel = 1 - (-750 - vel) / (-750 - (-400));
            if (vel > -750 && vel < -400)
                Nvel = (vel - (-750)) / ((-400) - (-750));
            if (vel > -400 && vel < -200)
                Nvel = 1;
            if (vel > -200 && vel < -70)
                Nvel = 1 - (-200 - vel) / (-200 - (-70));
            if (vel >= -200 && vel < -70)
                Zvel = (vel - (-200)) / (-70 - (-200));
            if (vel >= -70 && vel <= 70)
                Zvel = 1;
            if (vel > 70 && vel < 200)
                Zvel = 1 - (70 - vel) / (70 - 200);
            if (vel > 70 && vel < 200)
                Pvel = (vel - 70) / (200 - 70);
            if (vel > 200 && vel <= 400)
                Pvel = 1;
            if (vel > 400 && vel < 750)
                Pvel = 1 - (400 - vel) / (400 - 750);
            if (vel > 400 && vel < 750)
                PBvel = (vel - 400) / (750 - 400);
            if (vel >= 750)
                PBvel = 1;

            //Angle inputs Gaussian MFs to get the probability
            if (angle <= -45)
                NBangle = 1;
            if (angle > -45 && angle < -25)
                NBangle = 1 - (-45 - angle) / (-45 - (-25));
            if (angle >= -45 && angle < -25)
                Nangle = (angle - (-45)) / (-25 - (-45));
            if (angle >= -25 && angle <= -10)
                Nangle = 1;
            if (angle > -10 && angle <= -2)
                Nangle = 1 - (-10 - angle) / (-10 - (-2));
            if (angle >= -10 && angle < -2)
                Zangle = (angle - (-10)) / (-2 - (-10));
            if (angle >= -2 && angle <= 2)
                Zangle = 1;
            if (angle > 2 && angle <= 10)
                Zangle = 1 - (2 - angle) / (2 - 10);
            if (angle >= 2 && angle < 10)
                Pangle = (angle - 2) / (10 - 2);
            if (angle >= 10 && angle <= 25)
                Pangle = 1;
            if (angle > 25 && angle <= 45)
                Pangle = 1 - (25 - angle) / (25 - 45);
            if (angle >= 25 && angle < 45)
```

```
                PBangle = (angle - 25) / (45 - 25);
            if (angle >= 45)
                PBangle = 1;

            //Now we create an array with the strength of every rule and
afterwards, we create rules.
            int NumberOfRules = 25;
            float[] Strength = new float[NumberOfRules];
            float[] Rules = new float[NumberOfRules];

            if (NBvel != 0 && NBangle != 0)
            {
                Strength[0] = Math.Min(NBvel, NBangle);
                Rules[0] = Fuzzy.Z;
            }
            if (NBvel != 0 && Nangle != 0)
            {
                Strength[1] = Math.Min(NBvel, Nangle);
                Rules[1] = Fuzzy.Z;

            }
            if (NBvel != 0 && Zangle != 0)
            {
                Strength[2] = Math.Min(NBvel, Zangle);
                Rules[2] = Fuzzy.NB;

            }
            if (NBvel != 0 && Pangle != 0)
            {
                Strength[3] = Math.Min(NBvel, Pangle);
                Rules[3] = Fuzzy.NB;

            }
            if (NBvel != 0 && PBangle != 0)
            {
                Strength[4] = Math.Min(NBvel, PBangle);
                Rules[4] = Fuzzy.NB;

            }
            if (Nvel != 0 && NBangle != 0)
            {
                Strength[5] = Math.Min(Nvel, NBangle);
                Rules[5] = Fuzzy.Z;

            }
            if (Nvel != 0 && Nangle != 0)
            {
                Strength[6] = Math.Min(Nvel, Nangle);
                Rules[6] = Fuzzy.Z;

            }
            if (Nvel != 0 && Zangle != 0)
            {
                Strength[7] = Math.Min(Nvel, Zangle);
                Rules[7] = Fuzzy.N;

            }
            if (Nvel != 0 && Pangle != 0)
            {
                Strength[8] = Math.Min(Nvel, Pangle);
                Rules[8] = Fuzzy.N;
```

```
}
if (Nvel != 0 && PBangle != 0)
{
    Strength[9] = Math.Min(Nvel, PBangle);
    Rules[9] = Fuzzy.NB;

}
if (Zvel != 0 && NBangle != 0)
{
    Strength[10] = Math.Min(Zvel, NBangle);
    Rules[10] = Fuzzy.P;

}
if (Zvel != 0 && Nangle != 0)
{
    Strength[11] = Math.Min(Zvel, Nangle);
    Rules[11] = Fuzzy.P;

}
if (Zvel != 0 && Zangle != 0)
{
    Strength[12] = Math.Min(Zvel, Zangle);
    Rules[12] = Fuzzy.Z;

}
if (Zvel != 0 && Pangle != 0)
{
    Strength[13] = Math.Min(Zvel, Pangle);
    Rules[13] = Fuzzy.N;

}
if (Zvel != 0 && PBangle != 0)
{
    Strength[14] = Math.Min(Zvel, PBangle);
    Rules[14] = Fuzzy.N;

}
if (Pvel != 0 && NBangle != 0)
{
    Strength[15] = Math.Min(Pvel, NBangle);
    Rules[15] = Fuzzy.PB;

}
if (Pvel != 0 && Nangle != 0)
{
    Strength[16] = Math.Min(Pvel, Nangle);
    Rules[16] = Fuzzy.P;

}
if (Pvel != 0 && Zangle != 0)
{
    Strength[17] = Math.Min(Pvel, Zangle);
    Rules[17] = Fuzzy.P;

}
if (Pvel != 0 && Pangle != 0)
{
    Strength[18] = Math.Min(Pvel, Pangle);
    Rules[18] = Fuzzy.Z;

}
if (Pvel != 0 && PBangle != 0)
```

```
            {
                Strength[19] = Math.Min(Pvel, PBangle);
                Rules[19] = Fuzzy.Z;

            }
            if (PBvel != 0 && NBangle != 0)
            {
                Strength[20] = Math.Min(PBvel, NBangle);
                Rules[20] = Fuzzy.PB;

            }
            if (PBvel != 0 && Nangle != 0)
            {
                Strength[21] = Math.Min(PBvel, Nangle);
                Rules[21] = Fuzzy.PB;

            }
            if (PBvel != 0 && Zangle != 0)
            {
                Strength[22] = Math.Min(PBvel, Zangle);
                Rules[22] = Fuzzy.PB;

            }
            if (PBvel != 0 && Pangle != 0)
            {
                Strength[23] = Math.Min(PBvel, Pangle);
                Rules[23] = Fuzzy.Z;

            }
            if (PBvel != 0 && PBangle != 0)
            {
                Strength[24] = Math.Min(PBvel, PBangle);
                Rules[24] = Fuzzy.Z;

            }

            //Now we can formulate the output which is the sum of rules
multiplied by strengths splitted by sum of all strengths.
            //to save time, we make a variable with the sum of all strengths.
            float TotalStrength = 0;
            for (int x = 0; x < NumberOfRules; x++)
            {
                TotalStrength += Strength[x];
            }
            float Output = 0;

            for (int x = 0; x < NumberOfRules; x++)
            {
                Output += (Strength[x] * Rules[x]) / TotalStrength;
            }

            Fuzzy.Value = Output;
            return (Fuzzy);
        }

        public static FuzzyOutput RollStabilityController(FuzzyOutput Fuzzy,
float vel, float angle)
        {

            float NBvel = 0, Nvel = 0, Zvel = 0, Pvel = 0,
            PBvel = 0, NBangle = 0, Nangle = 0, Zangle = 0, Pangle = 0, PBangle =
0;
```

```
            // Velocity inputs MF to get probability
            if (vel <= -750)
                NBvel = 1;
            if (vel > -750 && vel <= -400)
                NBvel = 1 - (-750 - vel) / (-750 - (-400));
            if (vel > -750 && vel < -400)
                Nvel = (vel - (-750)) / ((-400) - (-750));
            if (vel > -400 && vel < -200)
                Nvel = 1;
            if (vel > -200 && vel < -70)
                Nvel = 1 - (-200 - vel) / (-200 - (-70));
            if (vel >= -200 && vel < -70)
                Zvel = (vel - (-200)) / (-70 - (-200));
            if (vel >= -70 && vel <= 70)
                Zvel = 1;
            if (vel > 70 && vel < 200)
                Zvel = 1 - (70 - vel) / (70 - 200);
            if (vel > 70 && vel < 200)
                Pvel = (vel - 70) / (200 - 70);
            if (vel > 200 && vel <= 400)
                Pvel = 1;
            if (vel > 400 && vel < 750)
                Pvel = 1 - (400 - vel) / (400 - 750);
            if (vel > 400 && vel < 750)
                PBvel = (vel - 400) / (750 - 400);
            if (vel >= 750)
                PBvel = 1;

            //Angle inputs Gaussian MFs to get the probability
            if (angle <= -45)
                NBangle = 1;
            if (angle > -45 && angle < -25)
                NBangle = 1 - (-45 - angle) / (-45 - (-25));
            if (angle >= -45 && angle < -25)
                Nangle = (angle - (-45)) / (-25 - (-45));
            if (angle >= -25 && angle <= -10)
                Nangle = 1;
            if (angle > -10 && angle <= -2)
                Nangle = 1 - (-10 - angle) / (-10 - (-2));
            if (angle >= -10 && angle < -2)
                Zangle = (angle - (-10)) / (-2 - (-10));
            if (angle >= -2 && angle <= 2)
                Zangle = 1;
            if (angle > 2 && angle <= 10)
                Zangle = 1 - (2 - angle) / (2 - 10);
            if (angle >= 2 && angle < 10)
                Pangle = (angle - 2) / (10 - 2);
            if (angle >= 10 && angle <= 25)
                Pangle = 1;
            if (angle > 25 && angle <= 45)
                Pangle = 1 - (25 - angle) / (25 - 45);
            if (angle >= 25 && angle < 45)
                PBangle = (angle - 25) / (45 - 25);
            if (angle >= 45)
                PBangle = 1;


            //Now we create an array with the strength of every rule and
afterwards, we create rules.
```

```
int NumberOfRules = 25;
float[] Strength = new float[NumberOfRules];
float[] Rules = new float[NumberOfRules];

if (NBvel != 0 && NBangle != 0)
{
    Strength[0] = Math.Min(NBvel, NBangle);
    Rules[0] = Fuzzy.PB;

}
if (NBvel != 0 && Nangle != 0)
{
    Strength[1] = Math.Min(NBvel, Nangle);
    Rules[1] = Fuzzy.PB;

}
if (NBvel != 0 && Zangle != 0)
{
    Strength[2] = Math.Min(NBvel, Zangle);
    Rules[2] = Fuzzy.PB;

}
if (NBvel != 0 && Pangle != 0)
{
    Strength[3] = Math.Min(NBvel, Pangle);
    Rules[3] = Fuzzy.Z;

}
if (NBvel != 0 && PBangle != 0)
{
    Strength[4] = Math.Min(NBvel, PBangle);
    Rules[4] = Fuzzy.Z;

}
if (Nvel != 0 && NBangle != 0)
{
    Strength[5] = Math.Min(Nvel, NBangle);
    Rules[5] = Fuzzy.PB;

}
if (Nvel != 0 && Nangle != 0)
{
    Strength[6] = Math.Min(Nvel, Nangle);
    Rules[6] = Fuzzy.P;

}
if (Nvel != 0 && Zangle != 0)
{
    Strength[7] = Math.Min(Nvel, Zangle);
    Rules[7] = Fuzzy.P;

}
if (Nvel != 0 && Pangle != 0)
{
    Strength[8] = Math.Min(Nvel, Pangle);
    Rules[8] = Fuzzy.Z;

}
if (Nvel != 0 && PBangle != 0)
{
    Strength[9] = Math.Min(Nvel, PBangle);
    Rules[9] = Fuzzy.Z;
```

```
}
if (Zvel != 0 && NBangle != 0)
{
    Strength[10] = Math.Min(Zvel, NBangle);
    Rules[10] = Fuzzy.P;

}
if (Zvel != 0 && Nangle != 0)
{
    Strength[11] = Math.Min(Zvel, Nangle);
    Rules[11] = Fuzzy.P;

}
if (Zvel != 0 && Zangle != 0)
{
    Strength[12] = Math.Min(Zvel, Zangle);
    Rules[12] = Fuzzy.Z;

}
if (Zvel != 0 && Pangle != 0)
{
    Strength[13] = Math.Min(Zvel, Pangle);
    Rules[13] = Fuzzy.N;

}
if (Zvel != 0 && PBangle != 0)
{
    Strength[14] = Math.Min(Zvel, PBangle);
    Rules[14] = Fuzzy.N;

}
if (Pvel != 0 && NBangle != 0)
{
    Strength[15] = Math.Min(Pvel, NBangle);
    Rules[15] = Fuzzy.Z;

}
if (Pvel != 0 && Nangle != 0)
{
    Strength[16] = Math.Min(Pvel, Nangle);
    Rules[16] = Fuzzy.Z;

}
if (Pvel != 0 && Zangle != 0)
{
    Strength[17] = Math.Min(Pvel, Zangle);
    Rules[17] = Fuzzy.N;

}
if (Pvel != 0 && Pangle != 0)
{
    Strength[18] = Math.Min(Pvel, Pangle);
    Rules[18] = Fuzzy.N;

}
if (Pvel != 0 && PBangle != 0)
{
    Strength[19] = Math.Min(Pvel, PBangle);
    Rules[19] = Fuzzy.NB;

}
```

```
            if (PBvel != 0 && NBangle != 0)
            {
                Strength[20] = Math.Min(PBvel, NBangle);
                Rules[20] = Fuzzy.Z;

            }
            if (PBvel != 0 && Nangle != 0)
            {
                Strength[21] = Math.Min(PBvel, Nangle);
                Rules[21] = Fuzzy.Z;

            }
            if (PBvel != 0 && Zangle != 0)
            {
                Strength[22] = Math.Min(PBvel, Zangle);
                Rules[22] = Fuzzy.NB;

            }
            if (PBvel != 0 && Pangle != 0)
            {
                Strength[23] = Math.Min(PBvel, Pangle);
                Rules[23] = Fuzzy.NB;

            }
            if (PBvel != 0 && PBangle != 0)
            {
                Strength[24] = Math.Min(PBvel, PBangle);
                Rules[24] = Fuzzy.NB;

            }

            //Now we can formulate the output which is the sum of rules
multiplied by strengths splitted by sum of all strengths.
            //to save time, we make a variable with the sum of all strengths.
            float TotalStrength = 0;
            for (int x = 0; x < NumberOfRules; x++)
            {
                TotalStrength += Strength[x];
            }
            float Output = 0;

            for (int x = 0; x < NumberOfRules; x++)
            {
                Output += (Strength[x] * Rules[x]) / TotalStrength;
            }

            Fuzzy.Value = Output;
            return (Fuzzy);

        }
```

C. ANEXO: CÓDIGO CONTROL POSICIÓN ROLL FUZZY

```
public static FuzzyOutput Roll5Controller(FuzzyOutput Fuzzy, float vel, float
angle, float DeltaX)
        {

            float NBvel = 0, Nvel = 0, Zvel = 0, Pvel = 0,
            PBvel = 0, NBangle = 0, Nangle = 0, Zangle = 0, Pangle = 0, PBangle =
0
            , PDeltaX = 0, ZDeltaX = 0, NDeltaX = 0;

            // Velocity inputs MF to get probability
            if (vel <= -750)
                NBvel = 1;
            if (vel > -750 && vel <= -400)
                NBvel = 1 - (-750 - vel) / (-750 - (-400));
            if (vel > -750 && vel < -400)
                Nvel = (vel - (-750)) / ((-400) - (-750));
            if (vel > -400 && vel < -200)
                Nvel = 1;
            if (vel > -200 && vel < -70)
                Nvel = 1 - (-200 - vel) / (-200 - (-70));
            if (vel >= -200 && vel < -70)
                Zvel = (vel - (-200)) / (-70 - (-200));
            if (vel >= -70 && vel <= 70)
                Zvel = 1;
            if (vel > 70 && vel < 200)
                Zvel = 1 - (70 - vel) / (70 - 200);
            if (vel > 70 && vel < 200)
                Pvel = (vel - 70) / (200 - 70);
            if (vel > 200 && vel <= 400)
                Pvel = 1;
            if (vel > 400 && vel < 750)
                Pvel = 1 - (400 - vel) / (400 - 750);
            if (vel > 400 && vel < 750)
                PBvel = (vel - 400) / (750 - 400);
            if (vel >= 750)
                PBvel = 1;

            //Angle inputs Gaussian MFs to get the probability
            if (angle <= -45)
                NBangle = 1;
            if (angle > -45 && angle < -25)
                NBangle = 1 - (-45 - angle) / (-45 - (-25));
            if (angle >= -45 && angle < -25)
                Nangle = (angle - (-45)) / (-25 - (-45));
            if (angle >= -25 && angle <= -10)
                Nangle = 1;
            if (angle > -10 && angle <= -2)
                Nangle = 1 - (-10 - angle) / (-10 - (-2));
            if (angle >= -10 && angle < -2)
                Zangle = (angle - (-10)) / (-2 - (-10));
            if (angle >= -2 && angle <= 2)
                Zangle = 1;
            if (angle > 2 && angle <= 10)
                Zangle = 1 - (2 - angle) / (2 - 10);
            if (angle >= 2 && angle < 10)
                Pangle = (angle - 2) / (10 - 2);
            if (angle >= 10 && angle <= 25)
                Pangle = 1;
            if (angle > 25 && angle <= 45)
                Pangle = 1 - (25 - angle) / (25 - 45);
```

```
            if (angle >= 25 && angle < 45)
                PBangle = (angle - 25) / (45 - 25);
            if (angle >= 45)
                PBangle = 1;

            // Deltax inputs MF to get probability
            if (DeltaX <= -500)
                NDeltaX = 1;
            if (DeltaX > -500 && DeltaX <= -300)
                NDeltaX = 1 - (-500 - DeltaX) / (-500 - (-300));
            if (DeltaX >= -500 && DeltaX < -300)
                ZDeltaX = (DeltaX - (-500)) / (-300 - (-500));
            if (DeltaX >= -300 && DeltaX <= 300)
                ZDeltaX = 1;
            if (DeltaX > 300 && DeltaX <= 500)
                ZDeltaX = (DeltaX - 300) / (500 - 300);
            if (DeltaX >= 300 && DeltaX < 500)
                PDeltaX = (DeltaX - 300) / (500 - 300);
            if (DeltaX >= 500)
                PDeltaX = 1;

            //Now we create an array with the strength of every rule and
afterwards, we create rules.
            int NumberOfRules = 75;
            float[] Strength = new float[NumberOfRules];
            float[] Rules = new float[NumberOfRules];

            if (NDeltaX != 0 && NBangle != 0 && NBvel != 0)
            {
                Strength[0] = Math.Min(NBvel, NBangle);
                Strength[0] = Math.Min(Strength[0], NDeltaX);
                Rules[0] = Fuzzy.PB;
                Fuzzy.PBbool = true;
            }
            if (NDeltaX != 0 && NBangle != 0 && Nvel != 0)
            {
                Strength[1] = Math.Min(Nvel, NBangle);
                Strength[1] = Math.Min(Strength[1], NDeltaX);
                Rules[1] = Fuzzy.P;
                Fuzzy.Pbool = true;
            }
            if (NDeltaX != 0 && NBangle != 0 && Zvel != 0)
            {
                Strength[2] = Math.Min(Zvel, NBangle);
                Strength[2] = Math.Min(Strength[2], NDeltaX);
                Rules[2] = Fuzzy.P;
                Fuzzy.Pbool = true;
            }
            if (NDeltaX != 0 && NBangle != 0 && Pvel != 0)
            {
                Strength[3] = Math.Min(Pvel, NBangle);
                Strength[3] = Math.Min(Strength[3], NDeltaX);
                Rules[3] = Fuzzy.P;
                Fuzzy.Pbool = true;
            }
            if (NDeltaX != 0 && NBangle != 0 && PBvel != 0)
            {
                Strength[4] = Math.Min(PBvel, NBangle);
                Strength[4] = Math.Min(Strength[4], NDeltaX);
                Rules[4] = Fuzzy.Z;
                Fuzzy.Zbool = true;
            }
```

```
if (NDeltaX != 0 && Nangle != 0 && NBvel != 0)
{
    Strength[5] = Math.Min(NBvel, Nangle);
    Strength[5] = Math.Min(Strength[5], NDeltaX);
    Rules[5] = Fuzzy.P;
    Fuzzy.Pbool = true;
}
if (NDeltaX != 0 && Nangle != 0 && Nvel != 0)
{
    Strength[6] = Math.Min(Nvel, Nangle);
    Strength[6] = Math.Min(Strength[6], NDeltaX);
    Rules[6] = Fuzzy.P;
    Fuzzy.Pbool = true;
}
if (NDeltaX != 0 && Nangle != 0 && Zvel != 0)
{
    Strength[7] = Math.Min(Zvel, Nangle);
    Strength[7] = Math.Min(Strength[7], NDeltaX);
    Rules[7] = Fuzzy.Z;
    Fuzzy.Zbool = true;
}
if (NDeltaX != 0 && Nangle != 0 && Pvel != 0)
{
    Strength[8] = Math.Min(Pvel, Nangle);
    Strength[8] = Math.Min(Strength[8], NDeltaX);
    Rules[8] = Fuzzy.Z;
    Fuzzy.Zbool = true;
}
if (NDeltaX != 0 && Nangle != 0 && PBvel != 0)
{
    Strength[9] = Math.Min(PBvel, Nangle);
    Strength[9] = Math.Min(Strength[9], NDeltaX);
    Rules[9] = Fuzzy.Z;
    Fuzzy.Zbool = true;
}
if (NDeltaX != 0 && Zangle != 0 && NBvel != 0)
{
    Strength[10] = Math.Min(NBvel, Zangle);
    Strength[10] = Math.Min(Strength[10], NDeltaX);
    Rules[10] = Fuzzy.P;
    Fuzzy.Pbool = true;
}
if (NDeltaX != 0 && Zangle != 0 && Nvel != 0)
{
    Strength[11] = Math.Min(Zangle, Nvel);
    Strength[11] = Math.Min(Strength[11], NDeltaX);
    Rules[11] = Fuzzy.Z;
    Fuzzy.Zbool = true;
}
if (NDeltaX != 0 && Zangle != 0 && Zvel != 0)
{
    Strength[12] = Math.Min(Zvel, Zangle);
    Strength[12] = Math.Min(Strength[12], NDeltaX);
    Rules[12] = Fuzzy.N;
    Fuzzy.Nbool = true;
}
if (NDeltaX != 0 && Zangle != 0 && Pvel != 0)
{
    Strength[13] = Math.Min(Pvel, Zangle);
    Strength[13] = Math.Min(Strength[13], NDeltaX);
    Rules[13] = Fuzzy.N;
    Fuzzy.Nbool = true;
```

```
}
if (NDeltaX != 0 && Zangle != 0 && PBvel != 0)
{
    Strength[14] = Math.Min(PBvel, Zangle);
    Strength[14] = Math.Min(Strength[14], NDeltaX);
    Rules[14] = Fuzzy.NB;
    Fuzzy.NBbool = true;
}
if (NDeltaX != 0 && Pangle != 0 && NBvel != 0)
{
    Strength[15] = Math.Min(NBvel, Pangle);
    Strength[15] = Math.Min(Strength[15], NDeltaX);
    Rules[15] = Fuzzy.Z;
    Fuzzy.Zbool = true;
}
if (NDeltaX != 0 && Pangle != 0 && Nvel != 0)
{
    Strength[16] = Math.Min(Nvel, Pangle);
    Strength[16] = Math.Min(Strength[16], NDeltaX);
    Rules[16] = Fuzzy.N;
    Fuzzy.Nbool = true;
}
if (NDeltaX != 0 && Pangle != 0 && Zvel != 0)
{
    Strength[17] = Math.Min(Zvel, Pangle);
    Strength[17] = Math.Min(Strength[17], NDeltaX);
    Rules[17] = Fuzzy.N;
    Fuzzy.Nbool = true;
}
if (NDeltaX != 0 && Pangle != 0 && Pvel != 0)
{
    Strength[18] = Math.Min(Pvel, Pangle);
    Strength[18] = Math.Min(Strength[18], NDeltaX);
    Rules[18] = Fuzzy.NB;
    Fuzzy.NBbool = true;
}
if (NDeltaX != 0 && Pangle != 0 && PBvel != 0)
{
    Strength[19] = Math.Min(PBvel, Pangle);
    Strength[19] = Math.Min(Strength[19], NDeltaX);
    Rules[19] = Fuzzy.NB;
    Fuzzy.NBbool = true;
}
if (NDeltaX != 0 && PBangle != 0 && NBvel != 0)
{
    Strength[20] = Math.Min(NBvel, PBangle);
    Strength[20] = Math.Min(Strength[20], NDeltaX);
    Rules[20] = Fuzzy.Z;
    Fuzzy.Zbool = true;
}
if (NDeltaX != 0 && PBangle != 0 && Nvel != 0)
{
    Strength[21] = Math.Min(Nvel, PBangle);
    Strength[21] = Math.Min(Strength[21], NDeltaX);
    Rules[21] = Fuzzy.N;
    Fuzzy.Nbool = true;
}
if (NDeltaX != 0 && PBangle != 0 && Zvel != 0)
{
    Strength[22] = Math.Min(Zvel, PBangle);
    Strength[22] = Math.Min(Strength[22], NDeltaX);
    Rules[22] = Fuzzy.N;
```

```
        Fuzzy.Nbool = true;
    }
    if (NDeltaX != 0 && PBangle != 0 && Pvel != 0)
    {
        Strength[23] = Math.Min(Pvel, PBangle);
        Strength[23] = Math.Min(Strength[23], NDeltaX);
        Rules[23] = Fuzzy.NB;
        Fuzzy.NBbool = true;
    }
    if (NDeltaX != 0 && PBangle != 0 && PBvel != 0)
    {
        Strength[24] = Math.Min(PBvel, PBangle);
        Strength[24] = Math.Min(Strength[24], NDeltaX);
        Rules[24] = Fuzzy.NB;
        Fuzzy.NBbool = true;
    }
    if (ZDeltaX != 0 && NBangle != 0 && NBvel != 0)
    {
        Strength[25] = Math.Min(NBvel, NBangle);
        Strength[25] = Math.Min(Strength[25], ZDeltaX);
        Rules[25] = Fuzzy.PB;
        Fuzzy.PBbool = true;
    }
    if (ZDeltaX != 0 && NBangle != 0 && Nvel != 0)
    {
        Strength[26] = Math.Min(Nvel, NBangle);
        Strength[26] = Math.Min(Strength[26], ZDeltaX);
        Rules[26] = Fuzzy.PB;
        Fuzzy.PBbool = true;
    }
    if (ZDeltaX != 0 && NBangle != 0 && Zvel != 0)
    {
        Strength[27] = Math.Min(Zvel, NBangle);
        Strength[27] = Math.Min(Strength[27], ZDeltaX);
        Rules[27] = Fuzzy.P;
        Fuzzy.Pbool = true;
    }
    if (ZDeltaX != 0 && NBangle != 0 && Pvel != 0)
    {
        Strength[28] = Math.Min(Pvel, NBangle);
        Strength[28] = Math.Min(Strength[28], ZDeltaX);
        Rules[25] = Fuzzy.Z;
        Fuzzy.Zbool = true;
    }
    if (ZDeltaX != 0 && NBangle != 0 && PBvel != 0)
    {
        Strength[29] = Math.Min(PBvel, NBangle);
        Strength[29] = Math.Min(Strength[29], ZDeltaX);
        Rules[29] = Fuzzy.Z;
        Fuzzy.Zbool = true;
    }
    if (ZDeltaX != 0 && Nangle != 0 && NBvel != 0)
    {
        Strength[30] = Math.Min(NBvel, Nangle);
        Strength[30] = Math.Min(Strength[30], ZDeltaX);
        Rules[30] = Fuzzy.PB;
        Fuzzy.PBbool = true;
    }
    if (ZDeltaX != 0 && Nangle != 0 && Nvel != 0)
    {
        Strength[31] = Math.Min(Nvel, Nangle);
        Strength[31] = Math.Min(Strength[31], ZDeltaX);
```

```
        Rules[31] = Fuzzy.P;
        Fuzzy.Pbool = true;
    }
    if (ZDeltaX != 0 && Nangle != 0 && Zvel != 0)
    {
        Strength[32] = Math.Min(Zvel, Nangle);
        Strength[32] = Math.Min(Strength[32], ZDeltaX);
        Rules[32] = Fuzzy.P;
        Fuzzy.Pbool = true;
    }
    if (ZDeltaX != 0 && Nangle != 0 && Pvel != 0)
    {
        Strength[33] = Math.Min(Pvel, Nangle);
        Strength[33] = Math.Min(Strength[33], ZDeltaX);
        Rules[33] = Fuzzy.Z;
        Fuzzy.Zbool = true;
    }
    if (ZDeltaX != 0 && Nangle != 0 && PBvel != 0)
    {
        Strength[34] = Math.Min(PBvel, Nangle);
        Strength[34] = Math.Min(Strength[34], ZDeltaX);
        Rules[34] = Fuzzy.Z;
        Fuzzy.Zbool = true;
    }
    if (ZDeltaX != 0 && Zangle != 0 && NBvel != 0)
    {
        Strength[35] = Math.Min(NBvel, Zangle);
        Strength[35] = Math.Min(Strength[35], ZDeltaX);
        Rules[35] = Fuzzy.PB;
        Fuzzy.PBbool = true;
    }
    if (ZDeltaX != 0 && Zangle != 0 && Nvel != 0)
    {
        Strength[36] = Math.Min(Nvel, Zangle);
        Strength[36] = Math.Min(Strength[36], ZDeltaX);
        Rules[35] = Fuzzy.P;
        Fuzzy.Pbool = true;
    }
    if (ZDeltaX != 0 && Zangle != 0 && Zvel != 0)
    {
        Strength[37] = Math.Min(Zvel, Zangle);
        Strength[37] = Math.Min(Strength[37], ZDeltaX);
        Rules[37] = Fuzzy.Z;
        Fuzzy.Zbool = true;
    }
    if (ZDeltaX != 0 && Zangle != 0 && Pvel != 0)
    {
        Strength[38] = Math.Min(Pvel, Zangle);
        Strength[38] = Math.Min(Strength[38], ZDeltaX);
        Rules[38] = Fuzzy.N;
        Fuzzy.Nbool = true;
    }
    if (ZDeltaX != 0 && Zangle != 0 && PBvel != 0)
    {
        Strength[39] = Math.Min(PBvel, Zangle);
        Strength[39] = Math.Min(Strength[39], ZDeltaX);
        Rules[39] = Fuzzy.NB;
        Fuzzy.NBbool = true;
    }
    if (ZDeltaX != 0 && Pangle != 0 && NBvel != 0)
    {
        Strength[40] = Math.Min(NBvel, Pangle);
```

```
        Strength[40] = Math.Min(Strength[40], ZDeltaX);
        Rules[40] = Fuzzy.Z;
        Fuzzy.Zbool = true;
    }
    if (ZDeltaX != 0 && Pangle != 0 && Nvel != 0)
    {
        Strength[41] = Math.Min(Nvel, Pangle);
        Strength[41] = Math.Min(Strength[41], ZDeltaX);
        Rules[41] = Fuzzy.Z;
        Fuzzy.Zbool = true;
    }
    if (ZDeltaX != 0 && Pangle != 0 && Zvel != 0)
    {
        Strength[42] = Math.Min(Zvel, Pangle);
        Strength[42] = Math.Min(Strength[42], ZDeltaX);
        Rules[42] = Fuzzy.N;
        Fuzzy.Nbool = true;
    }
    if (ZDeltaX != 0 && Pangle != 0 && Pvel != 0)
    {
        Strength[43] = Math.Min(Pvel, Pangle);
        Strength[43] = Math.Min(Strength[43], ZDeltaX);
        Rules[43] = Fuzzy.N;
        Fuzzy.Nbool = true;
    }
    if (ZDeltaX != 0 && Pangle != 0 && PBvel != 0)
    {
        Strength[44] = Math.Min(PBvel, Pangle);
        Strength[44] = Math.Min(Strength[44], ZDeltaX);
        Rules[44] = Fuzzy.NB;
        Fuzzy.NBbool = true;
    }
    if (ZDeltaX != 0 && PBangle != 0 && NBvel != 0)
    {
        Strength[45] = Math.Min(NBvel, PBangle);
        Strength[45] = Math.Min(Strength[45], ZDeltaX);
        Rules[45] = Fuzzy.Z;
        Fuzzy.Zbool = true;
    }
    if (ZDeltaX != 0 && PBangle != 0 && Nvel != 0)
    {
        Strength[46] = Math.Min(Nvel, PBangle);
        Strength[46] = Math.Min(Strength[46], ZDeltaX);
        Rules[46] = Fuzzy.Z;
        Fuzzy.Zbool = true;
    }
    if (ZDeltaX != 0 && PBangle != 0 && Zvel != 0)
    {
        Strength[47] = Math.Min(Zvel, PBangle);
        Strength[47] = Math.Min(Strength[47], ZDeltaX);
        Rules[47] = Fuzzy.N;
        Fuzzy.Nbool = true;
    }
    if (ZDeltaX != 0 && PBangle != 0 && Pvel != 0)
    {
        Strength[48] = Math.Min(Pvel, PBangle);
        Strength[48] = Math.Min(Strength[48], ZDeltaX);
        Rules[48] = Fuzzy.NB;
        Fuzzy.NBbool = true;
    }
    if (ZDeltaX != 0 && PBangle != 0 && PBvel != 0)
    {
```

```
        Strength[49] = Math.Min(PBvel, PBangle);
        Strength[49] = Math.Min(Strength[49], ZDeltaX);
        Rules[49] = Fuzzy.NB;
        Fuzzy.NBbool = true;
    }
    if (PDeltaX != 0 && NBangle != 0 && NBvel != 0)
    {
        Strength[50] = Math.Min(NBvel, NBangle);
        Strength[50] = Math.Min(Strength[50], PDeltaX);
        Rules[50] = Fuzzy.PB;
        Fuzzy.PBbool = true;
    }
    if (PDeltaX != 0 && NBangle != 0 && Nvel != 0)
    {
        Strength[51] = Math.Min(Nvel, NBangle);
        Strength[51] = Math.Min(Strength[51], PDeltaX);
        Rules[51] = Fuzzy.PB;
        Fuzzy.PBbool = true;
    }
    if (PDeltaX != 0 && NBangle != 0 && Zvel != 0)
    {
        Strength[52] = Math.Min(Zvel, NBangle);
        Strength[52] = Math.Min(Strength[52], PDeltaX);
        Rules[52] = Fuzzy.P;
        Fuzzy.Pbool = true;
    }
    if (PDeltaX != 0 && NBangle != 0 && Pvel != 0)
    {
        Strength[53] = Math.Min(Pvel, NBangle);
        Strength[53] = Math.Min(Strength[53], PDeltaX);
        Rules[53] = Fuzzy.P;
        Fuzzy.Pbool = true;
    }
    if (PDeltaX != 0 && NBangle != 0 && PBvel != 0)
    {
        Strength[54] = Math.Min(PBvel, NBangle);
        Strength[54] = Math.Min(Strength[54], PDeltaX);
        Rules[54] = Fuzzy.Z;
        Fuzzy.Zbool = true;
    }
    if (PDeltaX != 0 && Nangle != 0 && NBvel != 0)
    {
        Strength[55] = Math.Min(NBvel, Nangle);
        Strength[55] = Math.Min(Strength[55], PDeltaX);
        Rules[55] = Fuzzy.PB;
        Fuzzy.PBbool = true;
    }
    if (PDeltaX != 0 && Nangle != 0 && Nvel != 0)
    {
        Strength[56] = Math.Min(Nvel, Nangle);
        Strength[56] = Math.Min(Strength[56], PDeltaX);
        Rules[56] = Fuzzy.PB;
        Fuzzy.PBbool = true;
    }
    if (PDeltaX != 0 && Nangle != 0 && Zvel != 0)
    {
        Strength[57] = Math.Min(Zvel, Nangle);
        Strength[57] = Math.Min(Strength[57], PDeltaX);
        Rules[57] = Fuzzy.P;
        Fuzzy.Pbool = true;
    }
    if (PDeltaX != 0 && Nangle != 0 && Pvel != 0)
```

```
{
    Strength[58] = Math.Min(Pvel, Nangle);
    Strength[58] = Math.Min(Strength[58], PDeltaX);
    Rules[58] = Fuzzy.P;
    Fuzzy.Pbool = true;
}
if (PDeltaX != 0 && Nangle != 0 && PBvel != 0)
{
    Strength[59] = Math.Min(PBvel, Nangle);
    Strength[59] = Math.Min(Strength[59], PDeltaX);
    Rules[59] = Fuzzy.Z;
    Fuzzy.Zbool = true;
}
if (PDeltaX != 0 && Zangle != 0 && NBvel != 0)
{
    Strength[60] = Math.Min(NBvel, Zangle);
    Strength[60] = Math.Min(Strength[60], PDeltaX);
    Rules[60] = Fuzzy.PB;
    Fuzzy.PBbool = true;
}
if (PDeltaX != 0 && Zangle != 0 && Nvel != 0)
{
    Strength[61] = Math.Min(Nvel, Zangle);
    Strength[61] = Math.Min(Strength[61], PDeltaX);
    Rules[61] = Fuzzy.P;
    Fuzzy.Pbool = true;
}
if (PDeltaX != 0 && Zangle != 0 && Zvel != 0)
{
    Strength[62] = Math.Min(Zvel, Zangle);
    Strength[62] = Math.Min(Strength[62], PDeltaX);
    Rules[62] = Fuzzy.P;
    Fuzzy.Pbool = true;
}
if (PDeltaX != 0 && Zangle != 0 && Pvel != 0)
{
    Strength[63] = Math.Min(Pvel, Zangle);
    Strength[63] = Math.Min(Strength[63], PDeltaX);
    Rules[63] = Fuzzy.Z;
    Fuzzy.Zbool = true;
}
if (PDeltaX != 0 && Zangle != 0 && PBvel != 0)
{
    Strength[64] = Math.Min(PBvel, Zangle);
    Strength[64] = Math.Min(Strength[64], PDeltaX);
    Rules[64] = Fuzzy.N;
    Fuzzy.Nbool = true;
}
if (PDeltaX != 0 && Pangle != 0 && NBvel != 0)
{
    Strength[65] = Math.Min(NBvel, Pangle);
    Strength[65] = Math.Min(Strength[65], PDeltaX);
    Rules[65] = Fuzzy.Z;
    Fuzzy.Zbool = true;
}
if (PDeltaX != 0 && Pangle != 0 && Nvel != 0)
{
    Strength[66] = Math.Min(Nvel, Pangle);
    Strength[66] = Math.Min(Strength[66], PDeltaX);
    Rules[66] = Fuzzy.Z;
    Fuzzy.Zbool = true;
}
```

```
            if (PDeltaX != 0 && Pangle != 0 && Zvel != 0)
            {
                Strength[67] = Math.Min(Zvel, Pangle);
                Strength[67] = Math.Min(Strength[67], PDeltaX);
                Rules[67] = Fuzzy.Z;
                Fuzzy.Zbool = true;
            }
            if (PDeltaX != 0 && Pangle != 0 && Pvel != 0)
            {
                Strength[68] = Math.Min(Pvel, Pangle);
                Strength[68] = Math.Min(Strength[68], PDeltaX);
                Rules[68] = Fuzzy.N;
                Fuzzy.Nbool = true;
            }
            if (PDeltaX != 0 && Pangle != 0 && PBvel != 0)
            {
                Strength[69] = Math.Min(PBvel, Pangle);
                Strength[69] = Math.Min(Strength[69], PDeltaX);
                Rules[69] = Fuzzy.N;
                Fuzzy.Nbool = true;
            }
            if (PDeltaX != 0 && PBangle != 0 && NBvel != 0)
            {
                Strength[70] = Math.Min(NBvel, PBangle);
                Strength[70] = Math.Min(Strength[70], PDeltaX);
                Rules[70] = Fuzzy.Z;
                Fuzzy.Zbool = true;
            }
            if (PDeltaX != 0 && PBangle != 0 && Nvel != 0)
            {
                Strength[71] = Math.Min(Nvel, PBangle);
                Strength[71] = Math.Min(Strength[71], PDeltaX);
                Rules[71] = Fuzzy.N;
                Fuzzy.Nbool = true;
            }
            if (PDeltaX != 0 && PBangle != 0 && Zvel != 0)
            {
                Strength[72] = Math.Min(Zvel, PBangle);
                Strength[72] = Math.Min(Strength[72], PDeltaX);
                Rules[72] = Fuzzy.N;
                Fuzzy.Nbool = true;
            }
            if (PDeltaX != 0 && PBangle != 0 && Pvel != 0)
            {
                Strength[73] = Math.Min(Pvel, PBangle);
                Strength[73] = Math.Min(Strength[73], PDeltaX);
                Rules[73] = Fuzzy.N;
                Fuzzy.Nbool = true;
            }
            if (PDeltaX != 0 && PBangle != 0 && PBvel != 0)
            {
                Strength[74] = Math.Min(PBvel, PBangle);
                Strength[74] = Math.Min(Strength[74], PDeltaX);
                Rules[74] = Fuzzy.NB;
                Fuzzy.NBbool = true;
            }


            //Now we can formulate the output which is the sum of rules
multiplied by strengths splitted by sum of all strengths.
            //to save time, we make a variable with the sum of all strengths.
            float TotalStrength = 0;
```

```
        for (int x = 0; x < NumberOfRules; x++)
        {
            TotalStrength += Strength[x];
        }
        float Output = 0;

        for (int x = 0; x < NumberOfRules; x++)
        {
            Output += (Strength[x] * Rules[x]) / TotalStrength;
        }
        //Now we have to convert degrees in radians
        //float RadianOutput = Output * (float)Math.PI / 180;

        //Fuzzy.Value = RadianOutput;
        Fuzzy.Value = Output;
        return (Fuzzy);

    }
```

D. ANEXO: CÓDIGO CONTROLADOR DE ALTITUD

```
public static float AltitudeController(float altitude)
        {

            float NBaltitude = 0, NMaltitude = 0, Naltitude = 0, Zaltitude = 0,
Paltitude = 0,PMaltitude = 0, PBaltitude = 0;


            //altitude inputs Gaussian MFs to get the probability
            if (altitude <= -1000) NBaltitude = 1;
            if (altitude > -1000 && altitude <= -600)
                NBaltitude = Gaussbell(altitude, -1000, (float)141.6);

            if (altitude > -1000 && altitude <= -200)
                NMaltitude = Gaussbell(altitude, (float)-666.7, (float)141.6);

            if (altitude > -800 && altitude <= 200)
                Naltitude = Gaussbell(altitude, (float)-333.3, (float)141.6);

            if (altitude > -400 && altitude <= 400)
                Zaltitude = Gaussbell(altitude, 0, (float)141.6);

            if (altitude > -100 && altitude <= 800)
                Paltitude = Gaussbell(altitude, (float)333.3, (float)141.6);

            if (altitude > 200 && altitude <= 1000)
                PMaltitude = Gaussbell(altitude, (float)666.7, (float)141.6);

            if (altitude > 600 && altitude < 1000)
                PBaltitude = Gaussbell(altitude, 1000, (float)212.4);

            if (altitude >= 1000) PBaltitude = 1;

            ////Firstly we give values to possible altitude outputs



            float PBoutput = (float) 1;
            float PMoutput = (float)0.66;
            float Poutput = (float)0.33;
            float Zoutput = 0;
            float Noutput = (float)-0.33;
            float NMoutput = (float)-0.66;
            float NBoutput = (float)-1;

            //Now we create an array with the strength of every rule and
afterwards, we create rules.
            int NumberOfRules = 7;
            float[] Strength = new float[NumberOfRules];
            float[] Rules = new float[NumberOfRules];

            if (NBaltitude != 0)
            {
                Strength[0] = NBaltitude;
                Rules[0] = NBoutput;
            }
            if (NMaltitude != 0)
            {
                Strength[1] = NMaltitude;
```

```
                Rules[1] = NMoutput;
            }

            if (Naltitude != 0)
            {
                Strength[2] = Naltitude;
                Rules[2] = Noutput;
            }
            if (Zaltitude != 0)
            {
                Strength[3] = Zaltitude;
                Rules[3] = Zoutput;
            }
            if (Paltitude != 0)
            {
                Strength[4] = Paltitude;
                Rules[4] = Poutput;
            }

            if (PMaltitude != 0)
            {
                Strength[5] = PMaltitude;
                Rules[5] = PMoutput;
            }

            if (PBaltitude != 0)
            {
                Strength[6] = PBaltitude;
                Rules[6] = PBoutput;
            }
            //Now we can formulate the output which is the sum of rules
multiplied by strengths splitted by sum of all strengths.
            //to save time, we make a variable with the sum of all strengths.
            float TotalStrength = 0;
            for (int x = 0; x < NumberOfRules; x++)
            {
                TotalStrength += Strength[x];
            }
            float Output = 0;

            for (int x = 0; x < NumberOfRules; x++)
            {
                Output += (Strength[x] * Rules[x]) / TotalStrength;
            }

            return (Output);

        }
```

BIBLIOGRAFÍA

[1] V. V. D. F. a. J. F. Tomás Krajník, «AR-Drone as a Platform for Robotic Research and Education,» de *Research and Education in Robotics EUROBOT 2011*, Springer, 2011, pp. 172-186.

[2] G. B. A. Alba Coello Romero, *Fotogrametría de UAV de ala fija y,* Madrid: Universidad Politécnica de Madrid (PFC), 2014.

[3] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Parrot_AR.Drone. [Último acceso: mayo 2015].

[4] D. Mery, *Visión por computador,* Universidad Católica de Chile, 2004.

[5] «ARDrone 2.0,» [En línea]. Available: http://ardrone-2.es/especificaciones-ar-drone-2/. [Último acceso: enero 2015].

[6] «Microsoft C#,» Junio 2014. [En línea]. Available: http://msdn.microsoft.com/es-es/library/kx37x362.aspx. [Último acceso: 10 junio 2015].

[7] L. Moreno, *Fundamentos de la Lógica Fuzzy (L1),* Madrid, 2011.

[8] G. Viot, «Fuzzy Logic in C,» February 1993. [En línea]. Available: http://www.chebucto.ns.ca/Science/AIMET/archive/ddj/fuzzy_logic_in_C/. [Último acceso: febrero 2015].

[9] M. G. H. S. M. E. L. d. G. D. y. M. J. H. S. M.I. Alberto Pedro Lorandi Medina, «Revista de la Ingeniería Industrial,» vol. 5, nº 1, 2011.

[10] A. K. Gaurav, «Comparison between Conventional PID and Fuzzy Logic Controller for Liquid Flow Control: Performance Evaluation of Fuzzy Logic and PID Controller by Using MATLAB/Simullink,» *International Journal of Innovative Technology and Exploring Engineering (IJITEE),* vol. 1, nº 1, pp. 84-88, 2012.