Universidad Politécnica de Madrid

**Drone Autónomo**

Rodrigo Díaz Arroyo

Francisco Algaba de la Vieja

Sistemas Inteligentes

Ángel Arroyo Castillo

17/01/2017

**ÍNDICE**

[1. INTRODUCCIÓN 3](#_Toc472455603)

[1.1. DRONE 3](#_Toc472455604)

[1.2. CONTROLADOR BORROSO 4](#_Toc472455605)

[2. EXPLICACIÓN DEL CÓDIGO 6](#_Toc472455606)

[2.1. DRONE – Scripts en C# 6](#_Toc472455607)

[2.2. CONTROLADOR BORROSO – Clases en JAVA 8](#_Toc472455608)

[3. RESULTADOS 10](#_Toc472455609)

[4. CONCLUSIONES 12](#_Toc472455610)

[5. MEJORAS 12](#_Toc472455611)

[6. ANEXO I: Como ejecutar la simulación 13](#_Toc472455612)

1. **INTRODUCCIÓN**

El desarrollo de esta práctica tiene como objetivo el desarrollo de un drone autónomo. Dado un punto de destino el drone debe de ser capaz de alcanzar dicho punto evitando los obstáculos presentes entre el drone y el destino.

El escenario donde se simulará el drone se desarrolla en Unity. Por tanto, hacemos uso de las físicas proporcionadas por éste para conseguir un resultado lo más realista posible.

* 1. **DRONE**

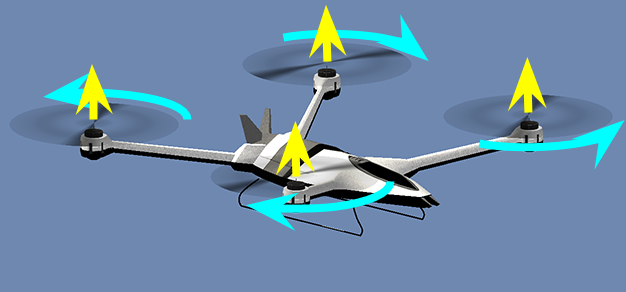
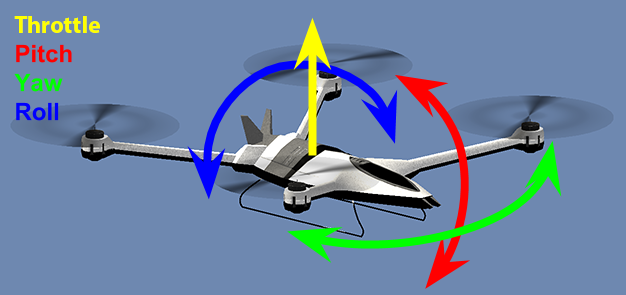
El drone es un cuadricóptero y tiene las siguientes características.

En primer lugar, el drone tiene asignada una masa y dispone de cuatro motores que aplican fuerzas independientes para simular el comportamiento realista de las hélices del drone. Dependiendo de las fuerzas aplicadas a cada motor el drone realizará un movimiento diferente.

Debido a que se hace una simulación realista del funcionamiento del drone fue necesario implementar un control de estabilización para que el drone fuese capaz de mantenerse en el aire a una determinada altura. Para el desarrollo de este sistema de estabilización se implementó un PID para los cuatro motores que es capaz de mantener estable el drone cuando está en estado “idle”. Además, también se ha implementado un PID para cada movimiento del drone para que estos movimientos sean constantes. Los distintos tipos de movimientos del drone son los siguientes:

* Throttle: movimiento del drone que le permite aumentar o disminuir la altura.
* Yaw: movimiento del drone que le permite rotar sobre sí mismo.
* Pitch: movimiento del drone que le permite inclinarse hacia delante y hacia detrás
* Roll: movimiento del drone que le permite inclinarse hacia la izquierda o la derecha.

La representación de la estructura del drone es la siguiente.



* 1. **CONTROLADOR BORROSO**

El desarrollo de la inteligencia artificial se hace a través de un controlador borroso implementado haciendo uso de la librería jFuzzyLogic.

Para el desarrollo de este sistema se ha establecido que el drone avance sólo hacia delante y que únicamente sea capaz de girar respecto de su propio eje (Yaw), es decir, no puede hacer uso de las funciones de inclinación hacia los lados.

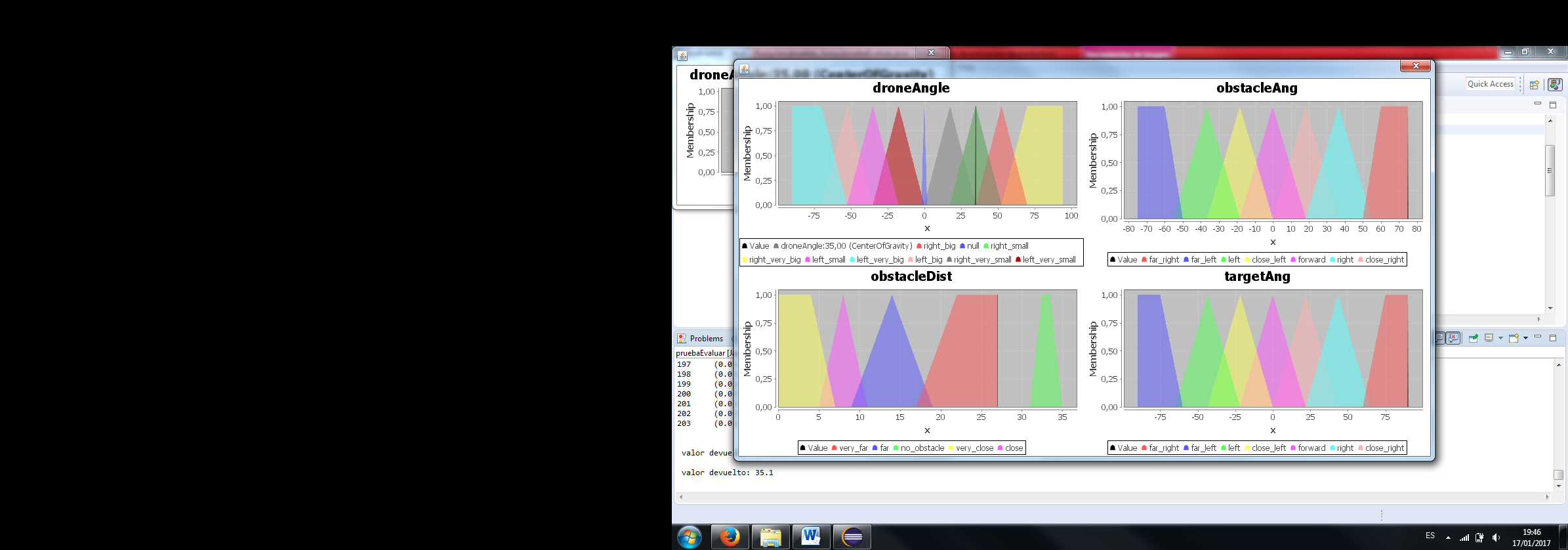
El desarrollo del sistema basado en reglas Fuzzy se puede separar en dos partes.

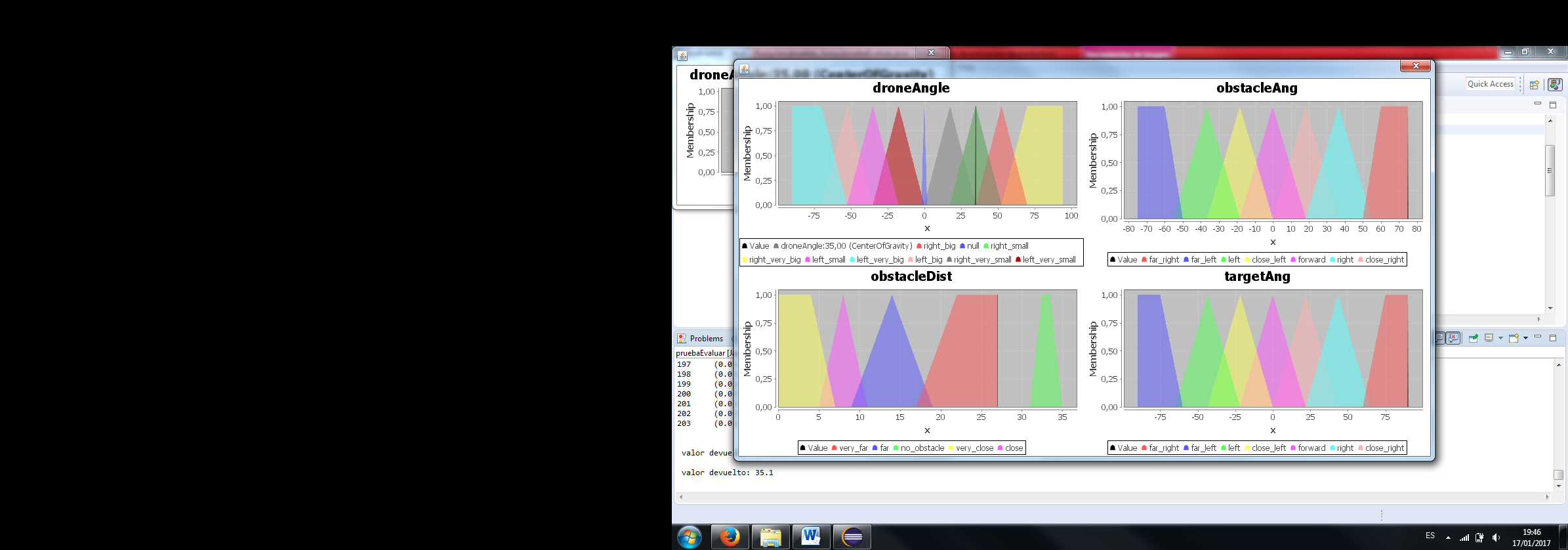
En primer lugar, la capacidad de evitar obstáculos que está basada en la recreación de una fuerza repulsiva del drone respecto de los obstáculos. Para la creación de esta parte se toman como entradas la distancia del obstáculo más cercano, el ángulo al que se encuentra el obstáculo respecto del drone y el ángulo de la posición destino respecto de la dirección del drone. Para la creación de este sistema de detección de obstáculos el drone cuenta con una esfera de un radio de 30 metros que lo recubre.

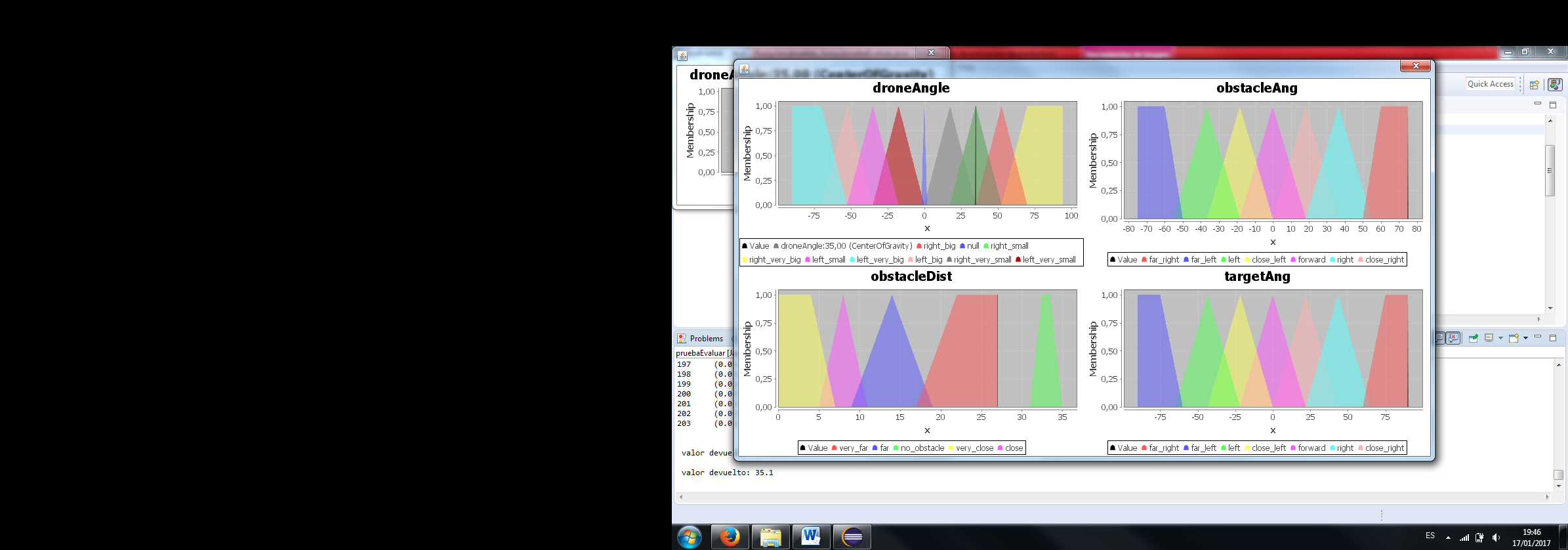
En segundo lugar, se recrea una fuerza de atracción del drone hacia la posición destino. Para su realización tomamos como entrada el ángulo de la posición destino respecto de la dirección del drone y si hay presencia de obstáculos cercanos.

Gracias a las entradas del controlador borroso generamos una salida que indica cuanto debe girar el drone para cumplir sus objetivos.

La representación de las entradas del controlador borroso es la siguiente.







1. **EXPLICACIÓN DEL CÓDIGO**

Este apartado se va a dedicar exclusivamente a la explicación de cada script involucrado en el funcionamiento del drone y el controlador borroso.

* 1. **DRONE – Scripts en C#**

A continuación, se va a explicar la función de cada script necesario para el funcionamiento y recogida de información del entorno del drone.

* ***Controller.cs***

Script diseñado inicialmente para el control del drone usando el teclado. Posteriormente se adaptó para su uso con la inteligencia artificial.

* ***BasicControl.cs***

Script encargado de actualizar la velocidad de cada motor del drone dependiendo del movimiento que esté realizando.

* ***ComputerModule.cs***

Script encargado de actualizar la velocidad de un motor del drone.

* ***BasicGyro.cs***

Script que implementa un giroscopio simple para actualizar la rotación del drone dependiendo del movimiento que esté realizando.

* ***PID.cs***

Implementación de un PID que se encarga de controlar los valores asignados para cada movimiento del drone.

* ***POV.cs***

Script encargado de mantener la cámara encima del drone.

* ***Motor.cs***

Script encargado de aplicar las fuerzas a cada motor.

* ***Move.cs***

Se encarga de mantener los ejes X y Z sin ninguna rotación cuando el drone se inclina. Con esto evitamos que las entradas del controlador Fuzzy sean las correctas y no tengan ningún tipo de rotación aplicada.

* ***Fitness.cs***

Script encargado de comprobar cuando el drone ha alcanzado el destino y finaliza llegado el caso. Además, también lleva la cuenta del número de colisiones sufridas por el drone y llama al script writeResultados para escribir los resultados en un fichero.

* ***Sensing.cs***

Este script se encarga de obtener todas las entradas necesarias para el controlador Fuzzy. En primer lugar, creamos nuestra esfera que recubre el drone y se encarga detectar los obstáculos dentro de su volumen y obtenemos la distancia a la que se encuentran y su ángulo respecto del drone. También obtenemos la distancia a la que está el destino y su ángulo respecto el drone. Llama a la función clienteFuzzy para conectar con el servidor que enviará los datos recogidos por el drone y recibirá el ángulo de giro.

* ***Tcp.cs***

Este script es un cliente TCP usado para comprobar el rendimiento del protocolo.

* ***clienteFuzzy.cs***

Script empleado para realizar la conexión con el servidorJAVA mediante el protocolo TCP. Crea una conexión la cual es usada para enviar y recibir los parámetros de entrada y salida del controlador borroso. Implementa el método evaluar que es llamado desde el script Sensing.

* ***writeResultados.cs***

Script encargado de crear un fichero en el que se escribirán los resultados de la simulación. Posee dos métodos que son llamados desde el script Sensing y se utilizan para escribir el número de colisiones y el tiempo tardado en completar el circuito.

A continuación, se muestran los scripts no terminados que van a ser usados para la creación del algoritmo evolutivo.

* ***DroneManager.cs***

Script no finalizado que se encargaría de gestionar los datos de la población de drones.

* ***EvolutionManager.cs***

Script no finalizado que se encargaría de la creación de un bucle en la escena para permitir la evolución de generaciones del algoritmo evolutivo.

* 1. **CONTROLADOR BORROSO – Clases en JAVA**

El lenguaje de programación Java se ha utilizado para programar todo el código correspondiente al controlador borroso. Esto es debido a que se ha empleado la librería jFuzzyLogic, la cual usa el lenguaje Java para implementar el lenguaje de control Fuzzy.

Dado que Unity no soporta el lenguaje Java se ha creado un cliente en Unity que conecta con un servidor en Java a través de Sockets. En primer lugar, esta conexión se realizó mediante el protocolo TCP, pero tras realizar algunas pruebas de rendimiento se optó por usar el protocolo UDP (tanto cliente como servidor se ejecutan en local y no habrá problemas de perdida de paquetes). Mientras que TCP baja el rendimiento a 17-22 frames por segundo, UDP es capaz de doblar el rendimiento a 40-45, lo cual mejora el rendimiento grafico del sistema.

**Clases para implementar el controlador borroso**

* ***obstacleAvoidance.java***

Clase encargada de gestionar el controlador borroso. Gracias a su constructor se carga el fichero “fcl” con el cual se crea un objeto “Fuzzy Inference System”. Esta clase también se compone del método “evaluar”, el cual gracias al objeto creado en el constructor evalúa las entradas del controlador borroso (distancia al obstáculo, ángulo al obstáculo y ángulo al punto de destino) para dar una salida (ángulo de giro del drone).

* ***servidorFuzzy.java***

Clase encargada de realizar la conexión mediante sockets con el cliente en Unity. Emplea el protocolo UDP para el intercambio de información. Recibe los tres parámetros de entrada del controlador borroso, realiza transformaciones entre tipos de datos, comprueba los datos recibidos y si todo es correcto hace uso de la clase obstacleAvoidance.java para evaluar los datos recibidos. Una vez que el método evaluar devuelve el valor del ángulo de giro del drone, se realiza una comprobación y se envía el dato al cliente en Unity.

***Fichero FCL***

Fichero que define el sistema de inferencia borroso (Fuzzy Inference System). El controlador borroso del drone se compone de tres variables de entrada: la distancia al obstáculo más cercano, el ángulo a dicho obstáculo y el ángulo al punto de destino.

Dado que es un sistema en tiempo real, para “fuzzificar” las variables se ha optado por emplear funciones trapezoidales y triangulares las cuales permiten una rápida computación.

La variable distancia al obstáculo se define mediante 5 etiquetas lingüísticas. La etiqueta ‘no\_obstacle’ tiene una función de pertenencia la cual no se solapa con la de ninguna otra etiqueta puesto que es un valor que se codifica en la clase servidorFuzzy.java (cuando el valor del ángulo al objeto recibido es mayor que 29 se modifica a 33 para que pertenezca a ‘no\_obstacle’).

La variable ángulo al punto de destino tiene 7 etiquetas lingüísticas para poder discernir con mayor claridad la posición del obstáculo con respecto de la cabeza del drone. Las funciones de pertenencia se definen en el rango de -75 grados a 75 grados.

La variable ángulo al punto de destino al igual que la variable anterior tiene 7 etiquetas lingüísticas. Pero esta vez las funciones de pertenencia están definidas en el rango de -90 grados a 90 grados.

Para “defuzzificar” la variable de salida ángulo del drone se utiliza el método “Centro de Gravedad” y se toma como valor por defecto el valor 0, el cual hace que el drone continúe en la misma dirección sin ejecutar ningún giro en caso de que ninguna regla se active. En esta ocasión el rango de la variable también está definida entre -90 y 90 grados, con la diferencia que se reparten entre 9 etiquetas lingüísticas para dar una mayor precisión a la hora de realizar las reglas.

En cuanto a las reglas se utiliza “min” como método de conexión y “max” como método de acumulación. Hay dos tipos de reglas. Por un lado, están las se definen con tres antecedentes (las tres variables de entrada) y un consecuente (la variable de salida), en las cuales se le da más importancia al hecho de evitar el obstáculo que al de intentar llegar al objetivo. Y por otro lado están las que se definen con dos antecedentes (distancia al obstáculo igual a no hay obstáculo y ángulo al punto de destino), las cuales se centran en llegar al obstáculo.

Dado el elevado número de reglas se optó por no darle peso a las reglas, por defecto todas las reglas tienen peso 1.

**Clases empleadas para realizar pruebas**

* ***servidorEchoTcp.java***

Primera clase implementada para realizar la conexión con Unity. Se trata de un servidor el cual recibe un mensaje y lo envía de vuelta al cliente. Con esta clase se comprobó el rendimiento del protocolo TCP y se decidió descartar el uso del mismo.

* ***servidorFuzzyComentado.java***

Es una copia de la clase servidorFuzzy.java en la cual se añaden comentarios para poder seguir y depurar lo recibido y enviado por el servidor.

* ***obstacleAvoidanceComentado.java***

Es una copia de la clase obstacleAvoidance.java esta clase implementa el controlador borroso gracias al fichero fcl. A diferencia de la clase sin comentarios, en esta se pueden ver por consola las variables evaluadas, un gráfico con el valor que toman las variables de entrada en el rango de cada una de ellas, un gráfico con el valor de salida del controlador y todas las reglas que se han activado con el grado de soporte.

* ***pruebaEvaluar.java***

Gracias a esta clase y a la clase anterior se puede evaluar caso a caso que reglas se activan dadas unas ciertas variables de entrada (muy útil cuando se detectan comportamientos anómalos en la ejecución).

Con el uso de esta clase pudimos evaluar la configuración del fichero obstacleAvoidance.fcl y adaptarlo a un comportamiento más realista.

1. **RESULTADOS**

Una vez que todo el código estaba desarrollado se pasó a evaluar el controlador borroso. Para ello se evaluó el archivo “obstacleAvoidance.fcl” el cual contiene todas las reglas y funciones de pertenencia. A la hora de evaluarlo se realizaban dos comprobaciones.

En primer lugar, se probaba que el resultado del controlador borroso fuera el esperado para unas determinadas entradas del sistema. Para ello se hizo uso de la clase “pruebaEvaluar.java”. Si los resultados del controlador no eran los esperados se comprobaban que reglas eran las que fallaban y en base a esto se modificaba el archivo .fcl.

En segundo lugar, una vez pasadas las pruebas de código, se procedería a probar el funcionamiento del sistema simulando el vuelo del drone en Unity. Gracias a esta simulación pudimos darnos cuentas de que reglas eran las que más se activaban y las que más influencia tenían a la hora de girar el drone. También ayudaba a ver que reglas entraban en acción en momentos críticos de la simulación, por ejemplo, cuando el drone chocaba contra un obstáculo.

Fue sobre todo gracias a la simulación cuando más se pudo mejorar el controlador borroso. No siempre que se intentaba hacer una mejora los resultados eran los esperados en simulación (como reflejan en los puntos 6 y 7 los gráficos inferiores).

Para ayudarnos a saber el número de colisiones y el tiempo tardado en cada simulación se hizo uso del script “writeResultados.cs”. Gracias al cual pudimos darnos cuenta que en ocasiones, a pesar de que el tiempo hubiera descendido, el número de colisiones se mantenía o incluso aumentaba con respecto a pruebas anteriores (como refleja el punto 4 en los gráficos inferiores).

Debido al uso de las físicas y a la diferente velocidad de conexión y computo en cada simulación, para cada versión del archivo .fcl se ha simulado un mínimo de cinco veces. Es por ello que ambas gráficas muestra una media del tiempo y el número de colisiones durante la simulación de cada versión del archivo .fcl.

No obstante, el sistema a veces tiene comportamientos anómalos y el drone no es capaz de llegar al objetivo. Esto es debido a que en el diseño actual el drone no frena y con la inercia que tiene cuando se produce una colisión demasiado ladeada con un obstáculo el drone rota demasiados grados y ya no es capaz de llegar al objetivo. Es por esta razón que los resultados de estas simulaciones se han descartado para realizar las gráficas.

1. **CONCLUSIONES**

Podemos concluir que el desarrollo de un drone autónomo ha sido exitoso. El drone es capaz de evitar obstáculos y llegar al destino de manera autónoma.

Como se comprueba en los resultados el rendimiento del drone es muy dependiente a las fuerzas externas como puede ser la inercia. Debido a que en el diseño actual no hay ninguna implementación para evitar esta fuerza, el drone sufre un pequeño número de colisiones. Dado que el drone no frena en ningún momento, es decir, sólo se encarga de girar las colisiones producidas por la inercia no son solucionables actualmente.

Por último, el desarrollo del sistema de inferencia borrroso implementado en este trabajo también es aplicable a otros entornos como puede ser un vehículo terrestre como pudiera ser un robot aspirador y gracias a que la inercia es inferior y que un sistema de frenado es menos complejo de implementar en el medio terrestre, podemos concluir que obtendría mejores resultados.

1. **MEJORAS**

Las posibles mejoras que se pueden implementar en el drone son las siguientes.

* **Sistema de control de inercia**: en el diseño actual el drone no es capaz de manejar la inercia por lo que sería una buena mejora para evitar las colisiones producidas por ésta.
* **Mejora del controlador borroso**: se podría mejorar añadiendo nuevas entradas para que el drone fuese capaz de evaluar varios obstáculos a la vez.
* **Finalización del controlador evolutivo**: terminar la implementación a medio hacer en el diseño actual para que el sistema sea capaz de adaptar el peso de las reglas o cambiar el consecuente de las reglas automáticamente (se pueden ver los scripts en la carpeta DroneAI>Assets>EvolutionSystem).

1. **ANEXO I: Como ejecutar la simulación**

En primer lugar, se utilizará la aplicación de escritorio Eclipse para abrir todos los archivos “.java”. Para ello se abrirá Eclipse y se seleccionará como directorio la siguiente ruta dentro de la carpeta descargada: “JAVA\workspace”. Una vez hecho esto, en el menú superior de eclipse se seleccionará File>Import>Existing Projects into Workspace. Se abrirá una ventana en la cual seleccionaremos la ruta de nuestro directorio clickando en “browse”, donde buscaremos la ruta de la carpeta descargada y dentro de ella seleccionaremos JAVA\workspace\Jfuzzy. Con esto ya tenemos nuestro proyecto importado.

En el caso de que en package explorer (a la izquierda de la pantalla) nos salga nuestro proyecto con un aspa roja, daremos click derecho al proyecto Jfuzzy y seleccionaremos Properties. En la ventana que se abre seleccionaremos “java build path”, después “libraries”. En este punto borraremos jFuzzyLogic.jar seleccionándolo y dándole a remove. Despues añadiremos nuestro JAR dando al botón “add external jars” y seleccionaremos el JAR el cual está en la carpeta descargada en la ruta “JAVA\workspace\Jfuzzy.”

El siguiente paso es desplegar nuestro proyecto y meternos en el paquete fuzzy, después seleccionaremos la clase servidorFuzzy y daremos al icono run (verde con un “play” blanco en su interior) para ejecutar la clase. (en caso de querer ver que se envía y se recibe en el servidor se recomienda ejecutar la clase servidorFuzzyComentado del paquete “pruebas\_fuzzy” en lugar de la clase servidorFuzzy).

En este punto ya tenemos arrancado el servidor (puerto 9900, protocolo UDP).

En segundo lugar, tenemos que abrir el proyecto de Unity. Para ello se abrirá la aplicación de escritorio Unity 5.4.1f1 y se abrirá el proyecto seleccionando la carpeta descargada. Una vez abierto, se seleccionará la escena “\_Scene” y se pulsará el botón play. ¡Ha comenzado la simulación del sistema!