Proyecto Fin de Carrera
Ingeniería Electrónica, Robótica y Mecatrónica

Navegación de UAVs con Deep Learning

Autor: Jorge Hidalgo Martín
Tutor: Begoña C. Arrue Ullés

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Carrera
Ingeniería Electrónica, Robótica y Mecatrónica

Navegación de UAVs con Deep Learning

Autor:
Jorge Hidalgo Martín

Tutor:
Begoña C. Arrue Ullés
Profesor titular

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Carrera: Navegación de UAVs con Deep Learning

Autor: Jorge Hidalgo Martín

Tutor: Begoña C. Arrue Ullés

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Este proyecto es el que cierra una etapa de mi vida, en estos últimos años he vivido múltiples experiencias y me gustaría agradecer a todas aquellas personas que han estado conmigo este tiempo.

Me gustaría comenzar agradeciendo a toda mi familia que me ha estado apoyando estos años, y a lo largo de todos mis estudios. Aunque para algunos soy 'el que hace los muñecos', sé que siempre se interesan y se preocupan por mí.

También agradecer a todos mis amigos tanto de fuera como dentro de la universidad. En especial a los amigos de la universidad agradecerles haber estado juntos estos años, sin ellos no habría podido estar todo este tiempo. Con amigos como vosotros quedarse hasta las 24h haciendo un proyecto sea entretenido. Con vosotros he compartido de los mejores años siempre acompañados de buenos momentos y grandes anécdotas.

Me gustaría también agradecer a todo el profesorado de la Universidad de Sevilla por el trabajo y la ayuda en esto este trayecto. Por abrirme un mundo nuevo y mantener mis ganas de seguir aprendiendo. Agradecer especialmente a Begoña por acompañarme en este último escalón en la universidad.

Finalmente me gustaría agradecer a todas aquellas personas que contribuyen de alguna forma a difundir el conocimiento y hacerlo accesible a muchas personas, desde tutoriales, proyectos, participación en foros entre otras actividades, su esfuerzo hace que otras personas tengamos más fácil poder seguir aprendiendo y desarrollar nuevo contenido.

Jorge Hidalgo Martín

Sevilla, 2019

Resumen

La Navegación es una de las tareas más complejas de la robótica. Esta involucra problemas de localización, para el punto de destino y del UAV, creación de trayectorias y movimiento para llegar al punto final.

El objetivo de este proyecto es crear una red neuronal capaz de proporcionar referencias en velocidad para hacer que el UAV llegue a la posición de destino. Para realizar esta tarea, se necesita cierta información, como las posiciones o imágenes, para así estrenar la red y hacer que el robot se mueva a través de entornos desconocidos. El UAV no solo tendrá que llegar a la posición final, también deberá esquivar cualquier obstáculo que se encuentre en el camino.

Para obtener los resultados, se ha hecho uso de ROS y Gazebo, los que nos permiten interaccionar con entornos de simulación realistas. Con ellos se han podido hacer experimentos para entrenar al agente y probar el funcionamiento del mimo en un entorno.

Abstract

Navigation is one of the most difficult tasks in robotics. It involves a location problem, for the target point and the UAV itself, path creation and movement in order to reach the final point.

The aim of this project is to create a neural network able to provide velocity references to make the UAV reach the final point. To perform this task, we need to feed it with some data, such as positions or images from the surroundings, so that the network can process all this information and move the robot through an unknown environment. Not only does it need to reach the final point, but it must avoid any collisions within the journey.

In order to get the results, it has been used ROS and Gazebo which enables us to interact with a realistic environment. With them we can make some experiments that we will use to train the agent, and later, test how was its performance.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xiv**

Índice de Tablas **xvi**

Índice de Figuras **xviii**

Notación **xx**

1 Introducción **1**
1.1 Motivación y objetivos *1*
1.2 Estructura del trabajo *4*

2 Descripción del problema **5**
2.1 Redes Neuronales *6*
2.1.1 Principios básicos 6
2.1.2 Aprendizaje 8
2.1.3 Redes neuronales convolucionales 9
2.2 Sensores de visión y profundidad *10*

3 Software **13**
3.1 ROS *13*
3.1.1 ¿Por qué usar ROS? 13
3.1.2 ¿Cómo funciona ROS? 13
3.2 Gazebo *14*
3.3 TensorFlow & Keras *15*
3.3.1 ¿Por qué Tensor Flow? 15
3.4 Otras librerías *15*
3.4.1 NumPy 15
3.4.2 OpenCV 16
3.4.3 Pygame 16
3.4.4 Sys – OS 16

4 Métodos Implementados **15**
4.1 Interfaz gráfica *15*
4.1.1 Control por teclado 16
4.1.2 Panel de mando 17
4.1.3 Comandos por terminal 19
4.2 Red neuronal *19*
4.2.1 Entradas 20
4.2.2 Salidas 23
4.2.3 Estructura de la red 24
4.2.4 Creación de entornos 25
4.2.5 Experimentos 27
4.3 Navegación del UAV *28*

4.3.1 Navegación por teclado 28
4.3.2 Navegación autónoma 29

5 Resultados 31
5.1 Entrenamiento 31
5.2 Movimiento con giro 33
5.3 Movimiento con obstáculos 36
5.4 Movimientos con múltiples obstáculos 38

6 Conclusiones y trabajo futuro 45
6.1 Posibles mejoras 45
6.2 Reflexión final y trabajo futuro 46

Referencias 47

Apéndice 49
Apéndice I: Esquema red completa 49

ÍNDICE DE TABLAS

Tabla 4-1 Significado de teclas 17
Tabla 4-2 Codificación ángulos 22
Tabla 4-3 Codificación velocidades 24
Tabla 5-1 Tipos de muestra 31
Tabla 5-2 Velocidades x 32
Tabla 5-3 Velocidades y 32
Tabla 5-4 Velocidades z 32
Tabla 5-5 Velocidades ψ 32

ÍNDICE DE FIGURAS

Figura 1-1 Vehículo autónomo 3
Figura 1-2 Delivery Drone 3
Figura 2-1 Entorno 6
Figura 2-2 Esquema neurona artificial 7
Figura 2-3 Esquema ANN 7
Figura 2-4 Ejemplo CNN X's 9
Figura 2-5 Características 'X' 10
Figura 2-6 Resultado convolución 10
Figura 2-7 Resultado pooling 10
Figura 3-1 Funcionamiento ROS 14
Figura 3-2 Ejemplo de entorno en Gazebo 15
Figura 4-1 Interfaz antigua 15
Figura 4-2 Interfaz nueva 16
Figura 4-3 Teclado 16
Figura 4-4 Diagrama de estados del funcionamiento 18
Figura 4-5 Diagrama estados de bloques 18
Figura 4-6 Imagen de profundidad 20
Figura 4-7 Posiciones propia y objetivo 21
Figura 4-8 Cambio de referencia 21
Figura 4-9 Estructura de la red simplificada 25
Figura 4-10 Building editor Gazebo 25
Figura 4-11 Columna 26
Figura 4-12 Pared 26
Figura 4-13 Esquina 27
Figura 5-1 Entorno movimiento con giro 33
Figura 5-2 Variación posición experimento con giro 34
Figura 5-3 Posición experimento con giro 34
Figura 5-4 Variación posición experimento con giro 2 35
Figura 5-5 Posición experimento con giro 2 36
Figura 5-6 Variación posición experimento con obstáculo 37
Figura 5-7 Posición experimento con obstáculo 37
Figura 5-8 Variación posición experimentos con múltiples obstáculos 38
Figura 5-9 Posición experimentos con múltiples obstáculos 39
Figura 5-10 Entorno con múltiples objetos 1 39

Figura 5-11 Variación posición con múltiples objetos, referencia dinámica 1 40
Figura 5-12 Posición con múltiples objetos, referencia dinámica 1 40
Figura 5-13 Entorno con múltiples objetos 2 41
Figura 5-14 Variación posición con múltiples objetos, referencia dinámica 2 42
Figura 5-15 Posición con múltiples objetos, referencia dinámica 2 42

Notación

UAV	Unmanned Aerial Vehicle
ROS	Robot Operating System
SLAM	Simultanious Location and Mapping
ReLU	Rectified Linear Unit
Ψ	Ángulo yaw
0>>	Valores muy negativos
0>	Valores algo negativos
~0	Valor casi nulo
0<	Valores algo positivos
0<<	Valores muy positivos

1 INTRODUCCIÓN

En los últimos años la inteligencia artificial ha avanzado de forma significativa. A pesar de que mucho de los conceptos matemáticos ya estaban desarrollados a mitad del siglo XX, la poca capacidad de cálculo de la época no ha permitido hasta ahora que sea viable este tipo de opciones. Con la capacidad de cálculo actual, es posible usar estás técnicas para solucionar problemas desde el punto de vista de la inteligencia artificial.

1.1 Motivación y objetivos

La navegación es uno de los problemas más apasionantes de la robótica, la idea en principio es sencilla, llegar de un punto A hacia un punto B, pero en ella intervienen múltiples factores que influyen, como es la interpretación del entorno, la generación de trayectorias y el movimiento del vehículo. Para poder interpretar el entorno, es preciso el uso de varios sensores, como los ultrasonidos, sensores visuales o sensores LASER. Los cuales son esenciales para captar lo que ocurre alrededor. También influye la tipología del vehículo según dónde se desplace, diferenciando los vehículos terrestres, aéreos y marinos; y dentro de ellos existen también distintas subcategorías, como un robot diferencial o un cheetah.

Respecto los robots aéreos como los terrestres, según el tipo de entorno se podrán usar o no una serie de sensores u otros. [1] Estos entornos se podrán dividir en dos grupos, de interiores y exteriores; cada uno tendrá unas limitaciones, que cambiarán la forma de enfrentar el problema. Estas limitaciones, son generalmente relacionados con la localización del robot. Por ejemplo, un robot en interiores no podrá usar el GPS debido a que existen interferencias haciendo que no pueda funcionar correctamente, siendo necesario el uso de sistemas alternativos como ultra wide-band, sin embargo, en exterior el GPS puede ser una mejor opción.

También se podrán diferenciar según la información sobre el entorno que se tenga, basado en mapas, el cual depende de un mapa del entorno. Basado en la construcción de mapas, en la que se usan sensores para construir un mapa y de este realiza la navegación. Navegación sin mapas, no usa ningún tipo de representación del entorno, sino que usa los sensores para detectar objetos generando así movimientos.

En entornos en los que se conoce el mapa, únicamente será necesario saber la localización del robot. Para ello, se podría usar técnicas de localización como las nombradas anteriormente. En los entornos en los que no se tenga información sobre el mapa, será necesario el uso de técnicas más avanzadas como SLAM [2] (Simultaneous Location and Mapping). Esta técnica es muy útil, ya que permite localizar el robot a la vez que crea un mapa del entorno. Respecto de la navegación sin mapas, no será necesario la creación de ninguno, sino que se interactuará en función de lo observado en el entorno.

La generación de trayectorias [3] es otra parte importante de la navegación ya que es el que se encarga de decidir el camino a tomar entre el punto A hacia el punto B. Esta generación de trayectoria debe tener en cuenta múltiples elementos que existan entre ambos puntos, de acuerdo con ciertas restricciones externas o propias del robot. Estos tendrán como objetivo encontrar el camino con el menor coste posible, entendiendo coste como el elemento que se quiere minimizar, cómo podría ser el tiempo o energía. Algunos de estos son los diagramas de Voronoi o descomposición de celdas exactas.

Todo lo descrito anteriormente, es una pequeña introducción de lo que es la navegación en robótica. Con los recientes avances en inteligencia artificial se han conseguido mejorar estos sistemas haciéndolos menos costosos y más robustos. A continuación, se pasará a explicar algunos ejemplos de como puede ser de utilidad la AI en el campo de la navegación, ya sea para complementar ciertas técnicas como para sustituirlas.

Mediante el uso de redes neuronales se pueden mejorar las técnicas de SLAM tradicionales [4], como Kinetic Fusión o dense SLAM que se basan en cámaras de profundidad. El uso de estas cámaras tiene ciertos inconvenientes, principalmente relacionados con el rango de acción o por el exceso de luz. Esto implica que sea preferible recrear un mapa de profundidad a partir de imágenes RGB, lo cual con ayuda de redes neuronales convolucionales se pueden construir mejores mapas, lo que permite tener una mejor interpretación del entorno y por tanto mejorar la navegación.

La inteligencia artificial no solo permite mejorar el funcionamiento de algunas técnicas, sino que en entornos en los que su uso no es posible o está muy limitado pueden ser sustituidos por otros [5]. Este el caso de entornos con muchos elementos en movimiento, como personas. Esto puede generar mucho ruido en las mediciones y detección de landmarks haciendo que el SLAM no sea viable. Por tanto, la creación de mapas o generación de trayectorias puede ser bastante costoso e impreciso. Sin embargo, como el acto de esquivar objetos es un acto reactivo del que solo depende los elementos cercanos. Con el uso de redes neuronales es posible pasar a través de este tipo de entorno sin colisionar, usando únicamente los datos directos de los sensores. Y gracias a la capacidad de generalización es posible enfrentarlo a distintos entornos sin necesidad de variar sus parámetros

Recientemente, se ha introducido también el uso de aprendizaje reforzado, el cual se utiliza cada vez más para resolver problemas de navegación [6]. El concepto general sobre el que se basa el aprendizaje reforzado es que el entrenamiento se realiza a través de un sistema de prueba y error, en el que se deja un agente en un entorno que realice una serie de movimientos y a partir de estos generar una serie de recompensas. El objetivo del agente será el de maximizar esta recompensa. Estas técnicas han tenido grandes éxitos en juegos de Atari [7] y a partir de ellas se pueden extrapolar para usarlas en la navegación. Con esta implementación se puede sustituir el procesado de la información de los sensores, la creación de mapas y la creación de trayectorias por este sistema de recompensas. Aunque todavía queda mucho trabajo por hacer en esta materia parece bastante prometedor.

Por lo general, el uso de aprendizaje reforzado es deseable, ya que permite reducir la complejidad del problema. Sin embargo, el tiempo necesario para entrenar este tipo de agentes para que resuelvan este tipo de tareas es elevado. En esta línea, se han desarrollado métodos por los cuales se puede reducir significativamente este tiempo [8]. Se trata de sistemas en los que reúne lo que sería las dos formas fundamentales de entrenarse, mediante imitación a un operario, lo cual es muy eficiente en número de muestras, pero sin ser demasiado robusto a cambios en el entorno, y por aprendizaje reforzado entrenando mediante prueba y error, lo que es ineficiente en número de muestras, pero muy robustos a cambios de entorno. Haciendo un entrenamiento escalonado, se puede reducir el tiempo de entrenamiento.

Todo lo mencionado anteriormente puede tener gran cantidad de aplicaciones. Por ejemplo, en el mundo del transporte de personas [9], dónde se calcula que el 90% de accidentes de tráfico son causados por errores humanos, el uso de inteligencia artificial en los vehículos para ayudas a la conducción o la conducción autónoma podría ayudar a decrementar ese valor.

Figura 1-1 Vehículo autónomo

También, es de gran utilidad para la entrega de mercancías, ya sea por propósito comercial como Amazon [10] o en lugares donde sea necesario enviar suministros como en operaciones de rescate en lugares de difícil acceso.

Figura 1-2 Delivery Drone

Hay que destacar, que gran parte del trabajo mencionado anteriormente está realizado para robots terrestres, debido a que los robots aéreos son más complejos ya que también pueden moverse también en el eje z. Esto permite reducir en cierta medida la dificultad del problema.

Este proyecto está motivado por los grandes avances que unen la inteligencia artificial con la navegación. Esto unido con el potencial de los UAVs y sus múltiples aplicaciones, lo convierten en un campo con muchas posibilidades. Además, el hecho de partir con poca base sobre el uso de redes neuronales es un reto añadido que se tiene en este proyecto. El objetivo del proyecto será crear una red neuronal, en la que un UAV al que se le da una posición objetivo, navegue hasta dicha posición esquivando cualquier obstáculo que se encuentre. Para ello no se tendrá ningún mapa del entorno ni será necesario crearlo.

Este proyecto también está motivado en parte, en uno realizado anteriormente [11], en este se pretende hacer navegar a un UAV en un entorno sin que colisione. Para el desplazamiento se usaban trayectorias predefinidas las cuales solo permitían el avance mientras se gira a un lado, por tanto, el espacio de acciones es muy limitado además de que no se podrá mover en el eje z. Para mejorar esto, se ha separado cada una de las cuatro componentes de velocidades para que sean independientes. Esto da mayor movilidad al UAV, que podrá realizar desplazamientos hacia los lados, moverse hacia arriba y abajo además de poder moverse hacia atrás.

1.2 Estructura del trabajo

El proyecto se divide en 6 capítulos, se comenzará haciendo una breve introducción al proyecto donde se expondrá como ha sido el desarrollo de este y se introducirán conceptos básicos de elementos usados en la elaboración del proyecto, en nuestro caso las redes neuronales. Posteriormente, se expondrá cual es el software usado junto alguno de sus rasgos fundamentales.

Una vez realizada la parte introductoria, se pasará a explicar los elementos que han sido necesario construir para llegar a la solución final. Entre ellas destaca una interfaz gráfica, la estructura de la red y el movimiento del UAV.

A continuación, se pasará a presentar los resultados, los cuales se han dividido en 4 bloques. El primero corresponderá con los datos de la red después de ser entrenada. Los siguientes bloques serán los relacionados con las pruebas, en los cuales, se ha enfrentado la red a distintos escenarios con características distintas.

Por último, se ha expuesto las posibles mejoras en cuanto a rendimiento o funcionalidad, así como la forma en la que se podría realizar. Además, se ha añadido una reflexión final a partir de la que se introducirá a un posible trabajo futuro.

2 DESCRIPCIÓN DEL PROBLEMA

El objetivo de este proyecto es crear una red neuronal profunda capaz de hacer navegar un UAV por un entorno evitando los posibles obstáculos que pueda haber en el camino, sin la necesidad de que exista un operario controlando el aparato.

En este proceso se pueden diferenciar 3 fases distintas:

1. Toma de muestras
2. Entrenamiento de la red
3. Testeo de la red en entorno simulado.

En la primera fase se ha realizado una serie de experimentos, en los cuales de forma manual se ha navegado el dron por un entorno simulado en Gazebo. De estos experimentos se han sacado una serie de variables de interés para poder entrenar posteriormente la red neuronal. Estas variables corresponden a las velocidades que se pretenden controlar, posición del UAV y las imágenes tomadas por la cámara. Todos estos datos son almacenados en unos archivos que se usarán para entrenar la red posteriormente.

Para la segunda fase se ha creado una red neuronal capaz de obtener las velocidades de referencia las cuales se usarán para mover el UAV. Para entrenarla, se han usado los datos obtenidos de los distintos experimentos, los cuales tras ser procesados se han usado para entrenar la red.

Finalmente, se ha probado en el entorno de Gazebo cómo se comporta esta red en distintos entornos, pretendiendo llegar a un objetivo propuesto. Estas pruebas comprenden escenarios parecidos a los que se usaron para entrenar la red y nuevos escenarios para comprobar si se ha aprendido a generalizar.

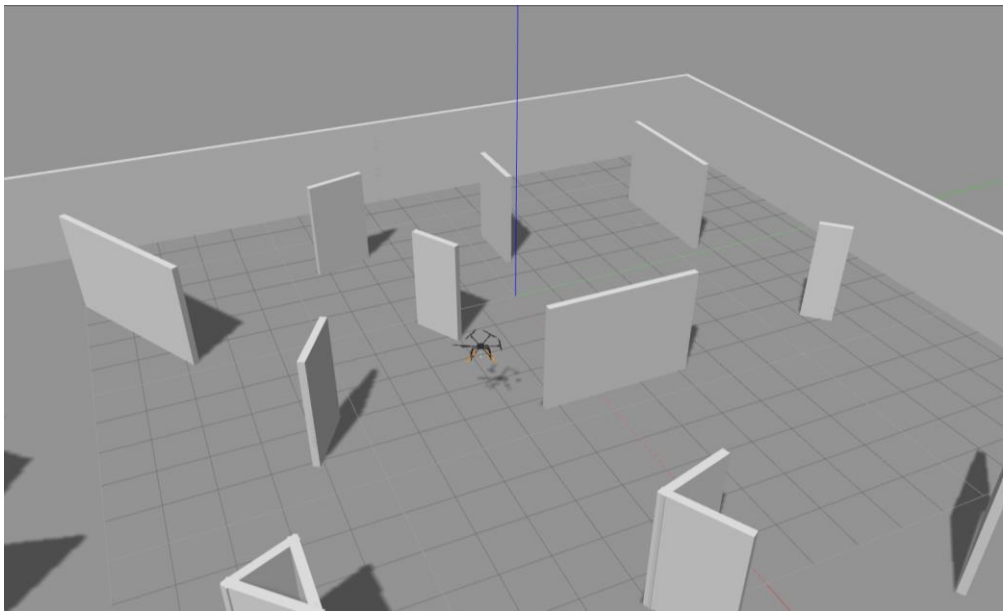

Figura 2-1 Entorno

2.1 Redes Neuronales

2.1.1 Principios básicos

Para poder hablar de redes neuronales [12] se debe de definir primero que es una neurona en el campo de la inteligencia artificial. Una neurona artificial es un sistema matemático que se basa en el modelo de las neuronas biológicas. Esta neurona recibe varias entradas que procesa para obtener una salida.

Cada neurona tiene una serie de partes que son necesarias entender.

- Entradas a_i, son los valores que se introducen y pueden ser salida de otras neuronas.
- Pesos w_{ij}, es un valor que multiplica la a cada una de las entradas, y determina el efecto que tiene esa entrada sobre el resultado final.
- Umbral o bias θ_j, es el valor que se suma a la entrada.
- Función de entrada, es la correspondiente a todas las entradas multiplicadas por sus respectivos pesos y sumándole el umbral.
- Función de activación, es la que relaciona la entrada y la salida, según cual sea la entrada se determinará cuán importante es la neurona para la salida. Las funciones más típicas son las sigmoides y las rectificadoras o ReLU.
- Salida, es el resultante de la función de activación.

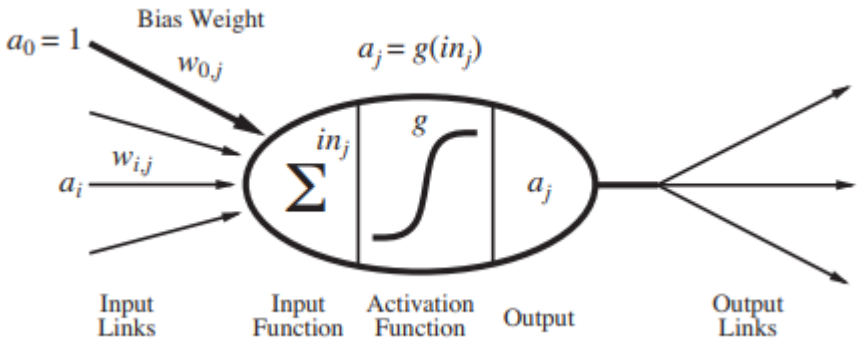

Figura 2-2 Esquema neurona artificial

Matemáticamente una neurona se podría escribir como

$$a_j^{(L)} = g\left(\left(\sum_{i=0}^{N} a_i^{(L-1)} * w_{jk}\right) + b_j\right)$$

Cuando se tienen múltiples neuronas se suelen colocar en capas. Según el lugar que ocupen en la red existen 3 tipos de capas

- Capa de entrada, corresponderá a las entradas del sistema.
- Capas ocultas, son las capas intermedias entre entrada y salida.
- Capa de salida, son las salidas del sistema.

El modelo de una capa (L) sería matemáticamente el siguiente

$$a^{(L)} = g\left(\begin{bmatrix} w_{0,0} & \cdots & w_{0,n} \\ \vdots & \ddots & \vdots \\ w_{k,0} & \cdots & w_{k,n} \end{bmatrix} * \begin{bmatrix} a_0^{(L-1)} \\ \vdots \\ a_n^{(L-1)} \end{bmatrix} + \begin{bmatrix} b_0^{(L)} \\ \vdots \\ b_k^{(L)} \end{bmatrix}\right)$$

En este k serán las neuronas en la capa actual y n las de la capa anterior. Así se tiene que w_{kn} es el peso de la entrada de la neurona n de la capa anterior en la neurona k.

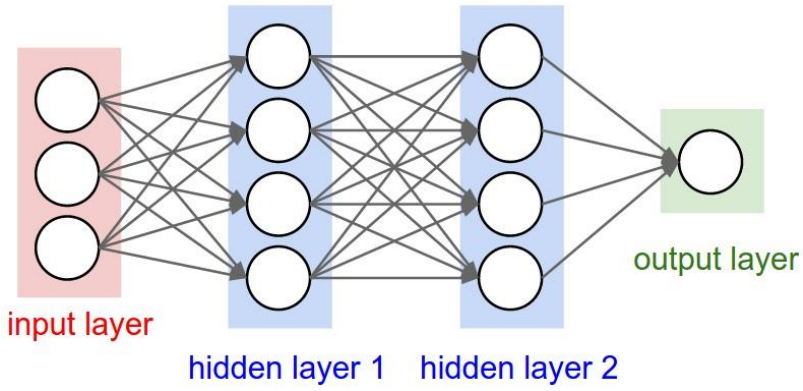

Figura 2-3 Esquema ANN

La finalidad de la red es que a través de estos valores de pesos y umbrales se pueda determinar una determinada solución. Con esta solución se irán cambiando progresivamente los valores de los pesos y los umbrales para que según unas entradas se generen unas determinadas salidas.

Existen distintas formas de entrenar a las redes [13], estas son:

- Supervisado, se tiene en todo momento cuales son las entradas y las salidas durante entrenamiento. Alguno de los problemas típicos son los de clasificación.
- Sin supervisar, en ocasiones no es posible tener una salida claramente definida, por lo que se tienen solo entradas, pero no se tienen las salidas. Este permite encontrar características similares, que permite por ejemplo buscar anomalías o agrupar elementos del mismo tipo.
- Semi-supervisado, es una mezcla del supervisado y el sin supervisar, especialmente útil cuando extraer características consume mucho tiempo.
- Aprendizaje reforzado, en este el agente encuentra la mejor manera para realizar cierta tarea. Dado un entorno el agente irá tomando acciones aleatorias y según como se vaya acercando al objetivo irá recibiendo una serie de recompensas. El objetivo será conseguir la máxima recompensa posible.

2.1.2 Aprendizaje

[14] En el aprendizaje supervisado, la finalidad es tener unas salidas concretas a partir de una serie de entradas. Al inicio se debe inicializar estos valores pesos y umbrales aleatoriamente y se irán modificando para conseguir la salida deseada. Matemáticamente, el cómo de cerca se está de la solución, se puede hacer de distintas formas, una de las más comunes es hacer mínimos cuadrados. Así teniendo una red de n_L salidas (a_j) y una serie de valores a los que se quiere aproximar y_j se tendría que la función de coste sería

$$C_j = \sum_{j=0}^{n_L-1} \left(a_j^{(L)} - y_j\right)^2$$

Los valores de la función de coste serán pequeños cuando los valores de las salidas sean parecidos a los valores que se quieren. Y será mayor cuando se diferencien mucho. Con esta función se puede conocer como de buena es una predicción.

Una vez obtenida la función de coste se deberá usar para modificar todos los parámetros (pesos y umbrales). Es preciso decir, que estas salidas dependen matemáticamente de estos valores como se ha visto anteriormente. Por tanto, esta función de costes se podría escribir como función de los parámetros que depende $C(w, b)$. Así, se deberán encontrar los parámetros que hagan que la función de coste llegue al mínimo posible. Una forma para minimizar este valor sería realizando derivadas parciales para cada uno de los parámetros. De esta manera, se persigue modificar los parámetros de forma que se llegue al mínimo local de la función de coste de la forma más directa. Así nuestro vector correspondiente a los valores de los distintos pesos y umbrales, v, se irá modificando para conseguir llegar a este mínimo.

$$v = v - \nabla C(w, b)$$

Una forma de entender como de importante es el efecto de cada parámetro en la solución, es mirar en esta derivada parcial y si el valor es muy elevado, significará que tiene un efecto importante en la red.

Para modificar cada uno de estos, ello primero se definirán ciertos parámetros que relacionarán todos los valores de los que depende una neurona.

$$z_j^{(L)} = \sum_{k=0}^{n_{(L-1)}} \left(w_{jk}^{(L)} * a_k^{(L-1)}\right) + b_j^{(L)} \quad a_j^{(L)} = g(z_j^{(L)})$$

Empezando por las salidas se conoce cuáles son los errores cometidos para cada una de las salidas. Estas dependerán de las salidas de la capa anterior, de los pesos que conectan las capas y el umbral de cada neurona. Cada uno de estos parámetros tendrá una influencia distinta en el resultado, y por tanto, analizando cada una de las salidas se tendrá una serie de cambios que debe realizar a cada uno de los pesos y umbrales para que se ajuste a la salida. Así, se le hará la media de esas variaciones para cambiar los valores, como los valores de las últimas capas dependen de los valores de las neuronas anteriores y estas a su vez de otros pesos, umbrales y capas anteriores. Así se llegará desde la capa de salida a la de entrada, esto se llama backpropagation o retro propagación. Las derivadas que miden lo anteriormente descrito serán las siguientes,

$$\frac{\partial C_j}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} * \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} * \frac{\partial C}{\partial a_j^{(L)}} \qquad \frac{\partial C_j}{\partial b_j^{(L)}} = \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} * \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} * \frac{\partial C}{\partial a_j^{(L)}} \qquad \frac{\partial C_j}{\partial a_k^{(L-1)}} = \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} * \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} * \frac{\partial C}{\partial a_j^{(L)}}$$

Haciendo uso de este proceso, es interesante hacer grupos de muestras a la hora de entrenar y que se haga una media de lo que debería cambiar cada uno de los valores. Si, por ejemplo, se quiere clasificar números, y únicamente se introduce un '2' la forma en la que estos parámetros cambiarán será de forma que se aproxime más a clasificar un '2'. Sin embargo, si se hace la media con los valores de por ejemplo un '1', '4', '7' y '2' esta aproximación se acercará más al comportamiento que se querría para la red (clasificar números), por tanto, será siempre interesante mezclar los datos para que se entrene de una forma más eficiente.

2.1.3 Redes neuronales convolucionales

Las Redes Neuronales Convolucionales (CNN) [15], consisten en múltiples capas de neuronas donde hay una de entrada y otra de salida, junto a las capas ocultas. Para entender cómo funcionan [16], hay que comprender que realizan las capas convolucionales y de pooling. El ejemplo consiste en clasificar una imagen, en la que se tiene una 'X' o una 'O'.

El principal problema a la hora de comparar 2 imágenes para saber si representan lo mismo con datos diferentes, por tanto, se tendrán 2 imágenes distintas, pero con rasgos similares que son los que se deben descubrir para saber si ambas imágenes representan lo mismo.

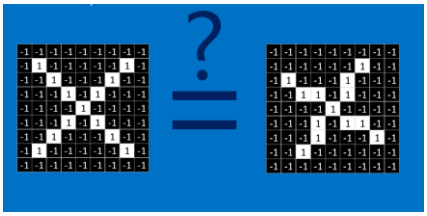

Figura 2-4 Ejemplo CNN X's

2.1.3.1 Características

Las CNNs van comparando las imágenes por partes, las cuales son llamadas características. Estas tratan de encontrar que características similares entre las 2 imágenes. Cada característica es como una pequeña imagen (por ejemplo, de 3x3), que une aspectos comunes de las imágenes. Para una 'X' sería interesante tener información sobre las líneas diagonales o sobre el cruce de líneas del centro, las cuales son las características fundamentales de una 'X'.

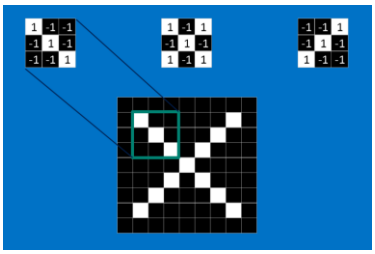

Figura 2-5 Características ‘X’

2.1.3.2 Convolución

A priori no se puede saber dónde se producirán estas similitudes, por lo que lo intentará por toda la imagen. Esta plantilla con la que se compara se llamará filtro, y para comprobar la similitud que tiene con el trozo de imagen, se usará la convolución.

Al pasar este filtro por toda la imagen y hacer la media para cada punto se obtiene donde en esa imagen existe la mayor similitud. Esto se deberá hacer para todos los filtros que se tengan.

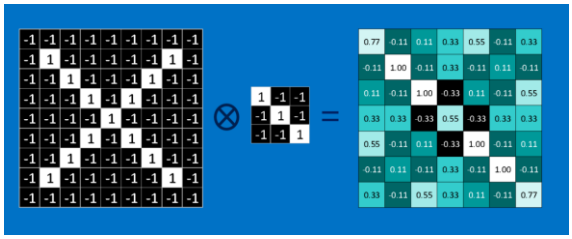

Figura 2-6 Resultado convolución

2.1.3.3 Pooling

Pooling es la forma de comprimir grandes imágenes, preservando la información más importante. Este se realiza pasando una pequeña ventana por la imagen y quedarse con el valor máximo de esa ventana cada vez.

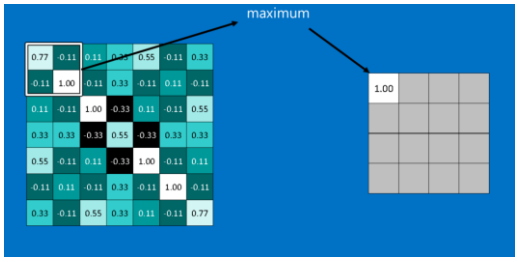

Figura 2-7 Resultado pooling

2.2 Sensores de visión y profundidad

En el entorno pueden existir gran cantidad de obstáculos, fijos o móviles, los cuales podrían provocar colisiones. Toda información que pueda extraer del entorno será útil cuando se quiere interaccionar con en él, ya sea para identificar objetos, interactuar con ellos, para evitarlos o sacar nueva información. Las cámaras RGB permiten obtener toda esta información para después procesarla y obtener la información acerca del entorno.

A parte de las cámaras RGB, existen también cámaras de profundidad, RGB-D, las cuales añaden información acerca de la distancia a la que se encuentra un punto en concreto. Este nuevo canal de información será especialmente útil en aplicaciones como [17],

- Interpretación de imágenes, permite obtener resultados más robustos en comparación con las RGB, principalmente sobre los objetos que se encuentran en primer plano.
- Mapeado de interiores, ahora se puede obtener una información detallada del entorno.
- Escenas con poca variación geométrica, como con paredes planas.
- Escaneo 3D, se pueden escanear objetos y crear un modelo en 3D.

3 SOFTWARE

En este capítulo se centra en los programas utilizado para realizar este proyecto. Este proyecto abarca el uso de redes neuronales, navegación y simulación. Por tanto, la elección de un buen software que abarque todos los requisitos de una forma eficiente es de vital importancia.

3.1 ROS

ROS [18] responde a las siglas de Robot Operating System y es una plataforma para software de robótica. Esta contiene una colección de herramientas, librerías y convenciones cuyo objetivo es crear un software de propósito general suficientemente robusto para distintas plataformas.

Como es sabido la interacción en distintos entornos es bastante compleja y es muy difícil que un solo individuo, laboratorio o institución pueda hacer todo de forma independiente. Así surge ROS, para animar una plataforma colaborativa, en la que expertos en distintas áreas puedan colaborar con algoritmos en el estado del arte. Y que esto sea accesible por otras entidades a lo largo del mundo.

3.1.1 ¿Por qué usar ROS?

ROS es una plataforma usada a nivel mundial y en su página oficial ofrece gran cantidad de ayudas para todos los niveles. Entre las principales ventajas destaca

- Gran cantidad de tutoriales, de nivel básico, nivel intermedio y avanzado. Con ellos puedes comenzar a familiarizarte con ROS y cómo funcionan cada uno de los elementos. También existen tutoriales adicionales que cubren gran parte de las herramientas de ROS.
- Comunidad, con foros en donde hay discusiones de cómo hacer ciertas tareas. Además, es muy útil a la hora de solucionar problemas que pueden ocurrir ya que existen referencias para casi cualquier tipo de error.

Otro de los motivos para usar ROS, es la facilidad para integrar paquetes de otras personas, al ser una plataforma modular basada en nodos, es muy sencillo interconectar distintos programas creados por entes distintos.

3.1.2 ¿Cómo funciona ROS?

Se tiene un workspace en el que se encuentran los siguientes archivos.

- Packages (paquetes), es la unidad fundamental de código en ROS. Cada paquete contiene librerías, ejecutables, código, u otros elementos.
- Manifests (manifiestos), es la descripción del paquete, este sirva para definir las dependencias entre paquetes y para capturar los metadatos del paquete, como son la versión, mantenedor, licencia, etc.
- Nodes (nodos), es un ejecutable que usa ROS para comunicarse entre ellos.
- Messages (mensajes), en una estructura que se usa cuando se subscribe o publica a un topic.
- Topics (temas), es el lugar donde los nodos pueden publicar sus mensajes. De la misma manera se

puede subscribir para leer estos mensajes.

- Master, es el encargado de gestionar la conexión entre los distintos elementos.
- Services (servicios), es una forma alternativa de comunicarse entre nodos, estos permiten a los nodos enviar solicitudes y recibir respuestas.

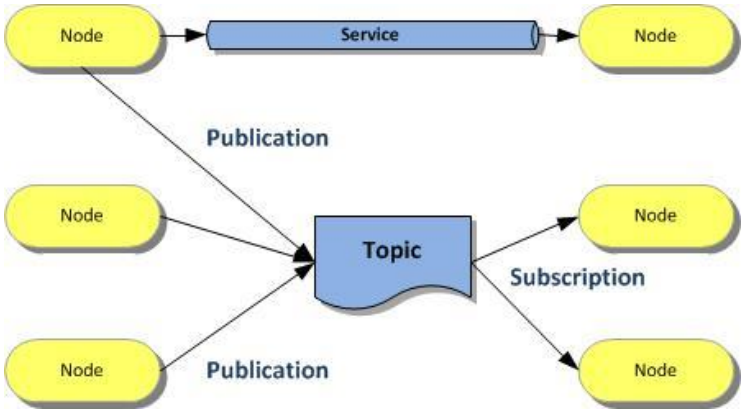

Figura 3-1 Funcionamiento ROS

De esta forma, se puede subdividir el problema como distintos nodos, y mediante el sistema de topics o servicios, comunicarse con otros para realizar una determinada acción.

3.2 Gazebo

Gazebo [19] es un simulador de dinámica en 3D con la capacidad de simular de forma precisa y eficiente poblaciones de robots en entornos de exterior e interior. De forma parecida a algunos motores de juego, Gazebo proporciona simulación de las físicas con mayor grado de fidelidad, una cantidad de sensores, e interfaces para los usuarios y programas.

Entre los usos típicos de Gazebo se encuentra:

- Testeo de algoritmos para la robótica.
- Diseño de robots.
- Testeo de funcionamientos en escenarios realistas.

Además, cuenta con multitud de modelos de elementos, para poder hacer un entorno lo más realista posible. Al igual que en ROS también puedes usar modelos creados por otra gente fácilmente. Asimismo, podrás crear tus propios modelos y con la herramienta building editor podrás crear edificaciones, útil sobre todo para interiores.

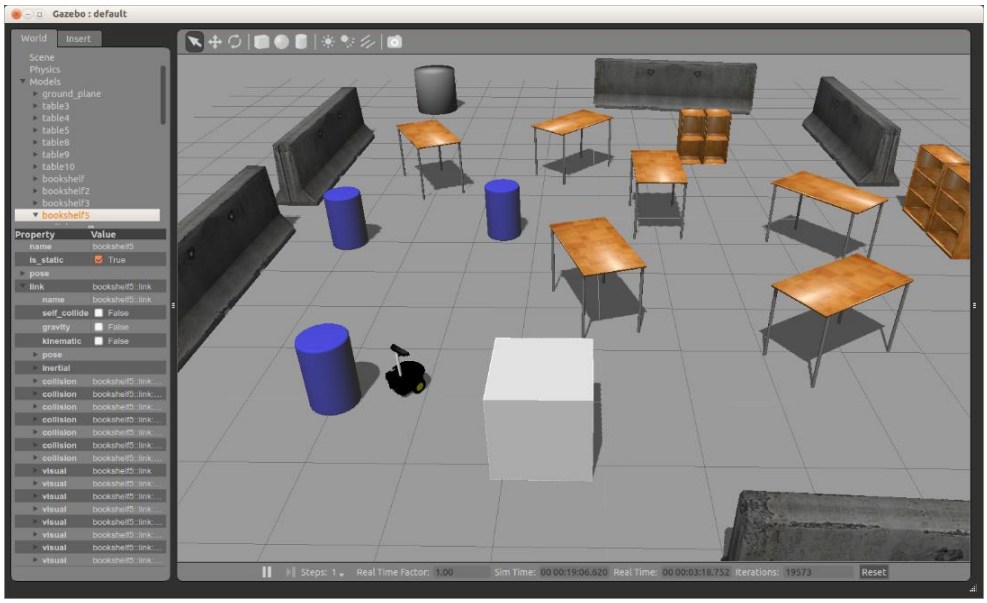

Figura 3-2 Ejemplo de entorno en Gazebo

3.3 TensorFlow & Keras

TensorFlow [20] es una plataforma de código abierto para machine learning. Este proporciona una gran cantidad de herramientas, librerías y recursos para permitir a los investigadores utilizar herramientas en el estado del arte de machine learning, fáciles de construir y usar para distintas aplicaciones.

3.3.1 ¿Por qué Tensor Flow?

Al ser TensorFlow de código abierto y tan usado a nivel global tiene muchas ventajas:

- Existe una gran comunidad de personas que trabajan con esta plataforma, por tanto, tiene una gran cantidad de discusiones sobre cómo realizar distintas tareas. También ayuda a solucionar ciertos errores pudiendo encontrar el origen de este.
- Una documentación extensa, en la que se encuentran todas las funciones, que significa cada uno de sus parámetros y el uso que tiene. Igualmente, algunas contienen algunos ejemplos para ver cómo se escribirían.
- Tutoriales, para ayudar a comprender como funciona la plataforma y algunos conceptos básicos.
- Además, permite múltiples niveles de abstracción para poder elegir el que más se ajuste a tus necesidades. En nuestro caso, usaremos Keras, el cual permite abstraernos de elementos de más bajo nivel. Y nos permite crear estructuras suficientemente complejas como la usada en el proyecto.

3.4 Otras librerías

Las librerías mencionadas anteriormente, tienen soporte para distintos lenguajes de programación como son C++ o Python. Sin embargo, al ser mucho más sencillo utilizar TensorFlow con Python, se ha decidido usar este frente a C++, lo que nos permitirá ahorrar tiempo de desarrollo. Para realizar el proyecto se han hecho uso de otras librerías.

3.4.1 NumPy

NumPy [21] es un paquete para computación con Python. Que contiene:

- Vectores N-dimensionales.

- Funciones de predicción.
- Herramientas para integrar C/C++

Este es bastante más potente que si usáramos las listas nativas de Python. Además, tiene integradas operaciones para realizar con matrices las cuales ahorraran tiempo de computo.

3.4.2 OpenCV

OpenCV [22] es una librería de código abierto el cual permite realizar operaciones con imágenes. Esta librería tiene interfaces para C++, Python y Java y soporta distintos sistemas operativos.

Esta librería permite trabajar fácilmente con imágenes, y realizar operaciones de procesado que haciéndolas manualmente serían más lentas en tiempo de cómputo. A pesar de tener muchas más funcionalidades, en este proyecto se ha usado principalmente para gestionar imágenes y cambiar el tamaño.

3.4.3 Pygame

Pygame [23] es una librería de código abierto para hacer aplicaciones multimedia. Pygame puede correr en casi cualquier plataforma y sistema operativo. Este nos facilita el poder tener acceso a elementos como el teclado o el ratón de una forma fácil y simple.

Con ella se pueden crear una aplicación y hacerla sensible al ratón para cambiar algunas opciones o el teclado para mandar alguna consigna.

3.4.4 Sys – OS

Estas dos librerías [24] [25] son nativas de Python y permite acceder a elementos del sistema operativo. La principal utilidad es tener acceso a los distintos archivos desde el programa. También permite leer los parámetros introducidos en la línea de comandos.

4 MÉTODOS IMPLEMENTADOS

En este capítulo vamos a describir todos los puntos necesarios para para poder llegar a obtener resultados. Para ello es necesario crear una interfaz gráfica la cual se usará para manipular el UAV, la creación de la red neuronal para crear un agente capaz de realizar la tarea asignada y finalmente la comunicación con el control de vuelo para que se pueda volar.

4.1 Interfaz gráfica

Para la realización de los distintos experimentos y comprobar los resultados es necesario poder interactuar con el entorno en Gazebo. Esto se realizará mediante una interfaz que permita; controlar el UAV usando el teclado, visualizar los parámetros que se envían y conectar la red neuronal para que sea la que controle el UAV. Si se tuviera una interfaz que permitiera lo mencionado anteriormente, se podría exprimir al máximo la funcionalidad sin necesidad de cambiar líneas de código ni imprimir por pantalla los resultados.

Desde el Departamento de Ingeniería de Sistemas y Automática de la Universidad de Sevilla se ha proporcionado un modelo de UAV [26] el cual permitía una serie de interfaces, con teclado, mando de la XBOX y gamePAD. Entre estas, la que más interés es la que hace uso del teclado para el movimiento.

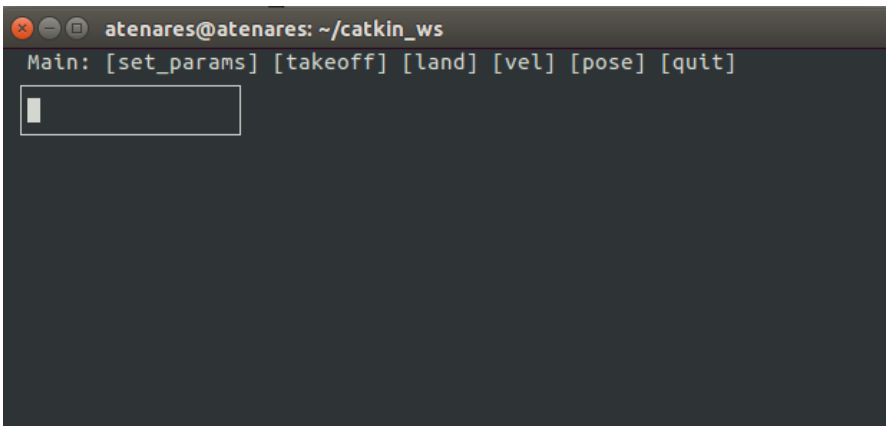

Figura 4-1 Interfaz antigua

Esta interfaz tenía una serie de inconvenientes:

- Complejidad de interfaz, para navegar por ella debes introducir comandos desde el teclado.
- Control de operario reducido, solo podía pulsarse un botón a la vez. Esto impedía que se pudiera avanzar en diagonal, ya que habría que ir hacia delante y hacia el lado simultáneamente, lo que reduce el dinamismo de los experimentos.
- No existe la posibilidad de conectar una red neuronal para que realice el movimiento.
- No se pueden introducir referencias de posición para introducirlas en la red.

Para salvar estos inconvenientes, se ha cambiado la interfaz y funcionalidad de forma casi completa. Para poder realizarlo se ha hecho uso de pygame, la cual sería la siguiente.

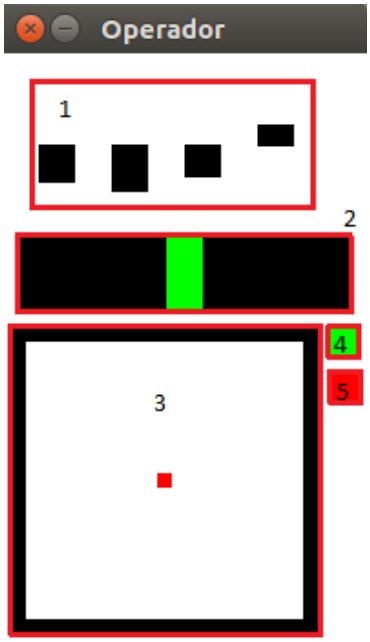

Figura 4-2 Interfaz nueva

1. Indicadores de velocidad a la cual va el UAV. El orden es velocidad en x, y, z y yaw.
2. Indicador de alineación con objetivo.
3. Pantalla de referencias, permite en un rango de 20x20 dar una referencia de posición al dron para que la siga (solo para cuando esté conectado la red neuronal).
4. Botón para iniciar el despegue. Permite despegar a una determinada altura (1.5 metros por defecto).
5. Conectar red neuronal. Hace que el control del dron sea por la acción de la red. Por defecto se controlará desde el teclado.

4.1.1 Control por teclado

Como se tienen 4 ejes de movimiento para el UAV, es necesario de 4 parejas de teclas para cubrir todos los movimientos posibles. Las teclas utilizadas serán las mismas que las que se tenía por defecto en la interfaz que se tenía al principio y corresponderán a la misma función,

Figura 4-3 Teclado

Cada una de estas teclas tiene una función que es la siguiente:

Teclas A-D	'A' giro en yaw hacia izquierda	'D' giro en yaw hacia derecha
Teclas S-W	'S' movimiento hacia abajo	'W' movimiento hacia arriba

Flecha Abajo-Arriba	Flecha abajo, movimiento hacia atrás	Flecha arriba, movimiento hacia delante
Flecha Izquierda-Derecha	Flecha izquierda, movimiento hacia izquierda	Flecha derecha, movimiento hacia derecha

Tabla 4-1 Significado de teclas

Cada botón conforma una clase en la cual se puede imponer un valor correspondiente a la velocidad en esa dirección. Cuando se impone un valor también se debe acotar entre un valor máximo y mínimo (valor absoluto) y también informa del estado; si esta pulsado o no.

Para poder cubrir el rango de velocidades en una dirección en ambos sentidos, es necesario agrupar estas teclas por parejas. Estas parejas se han unido formando una clase que da información sobre el valor de la velocidad en una determinada dirección. Estas clases permiten que cuando no se pulse ninguna tecla la perdida de velocidad sea progresiva o sea inmediata, hasta un valor nulo. Además, se deberán crear prioridades en los casos en que ambas teclas estén activas simultáneamente. Los casos serían los siguientes, (tener en cuenta que 'arriba' significa tomar valores positivos y 'abajo' tomar valores negativos):

- Si arriba está activo y abajo desactivado, se incrementa progresivamente hacia valores positivos, clase de 'arriba'. Mientras se mantiene a 0 la clase de 'abajo'.
- Si arriba está desactivado y abajo activo, se decrementa progresivamente hacia valores negativos, clase de 'abajo'. Mientras se mantiene a 0 la clase de 'arriba'.
- Si ambos están desactivados, según si el valor era positivo o negativo, se decrementará o incrementará su valor respectivamente.
- Si ambos están activados, se deja como estaba, es decir, si estaba aumentando seguirá aumentando y viceversa, si estaba disminuyendo seguirá haciéndolo.

Cabe destacar la posibilidad de que cuando se quiera aumentar el valor, por ejemplo, esté en un valor negativo y que por tanto haya que empezar a aumentar desde ahí. Esto puede hacerse de 2 formas, las cuales están implementadas mediante una bandera. La prima forma sería llevarlo directamente hasta un valor nulo y a partir de ahí aumentar la velocidad, o aumentar el valor hasta que sea nulo y después seguir aumentando.

4.1.2 Panel de mando

El panel de mando es el encargado de gestionar las interacciones del usuario con la interfaz. Estas comprenden la representación de datos, gestión de botones, cambio de funcionalidades y movimiento dinámico de la referencia.

Para crear los gráficos se ha hecho uso de la librería pygame, con el que se ha creado una pantalla de 200x350. Pygame funciona por ciclos, en los que con un reloj interno que marca los periodos de la aplicación. En cada ciclo se rellena la pantalla de un color de fondo, para poder sobrescribir la información, después se interactúa con ella y se dibujan los elementos que se quieren mostrar, y finalmente se actualiza la pantalla para mostrar estos cambios.

4.1.2.1 Funcionamiento

El diagrama de estados correspondiente a la funcionalidad del panel de mando será el siguiente,

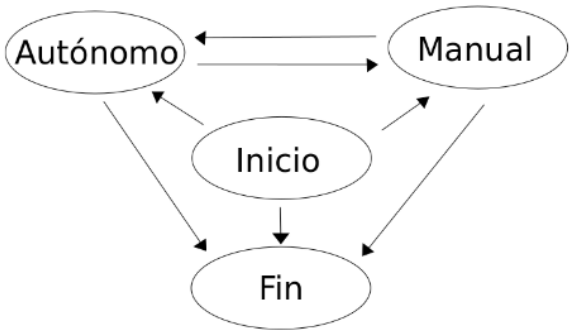

Figura 4-4 Diagrama de estados del funcionamiento

La funcionalidad por tanto sería la siguiente, en primera instancia el UAV se encuentra en el estado de 'inicio' parado sin hacer nada. Para poder manejarlo haciendo del teclado, se debe pulsar el botón (4) de la interfaz, el cual permite despegar a una altura predeterminada. Una vez en el aire, se puede hacer uso del teclado para moverlo por el entorno. Durante el vuelo se mostrará el valor de velocidad a través de unas barras en el indicador de la interfaz (1).

Para poder utilizar la red neuronal, se debe activar pulsando el botón (5), esto se puede hacer en cualquier momento. Si por ejemplo todavía no se ha empezado a moverse cuando se pulse el botón, habrá que esperar a que haya despegado (habría que pulsar botón (4)) para que este empiece a funcionar. Hay que destacar que pulsando el botón (5) se puede volver a cambiar entre el funcionamiento autónomo y manual.

En el caso en el que esté en modo autónomo, se necesitará una referencia para que el dron vaya hacia ella. Esta referencia será dada a través de (3) el cual permite seleccionar una posición en rango que va desde -10 a 10 en el eje x e y. Cabe destacar que en este caso la posición z final estará marcada por defecto.

4.1.2.2 Conexión con resto del sistema

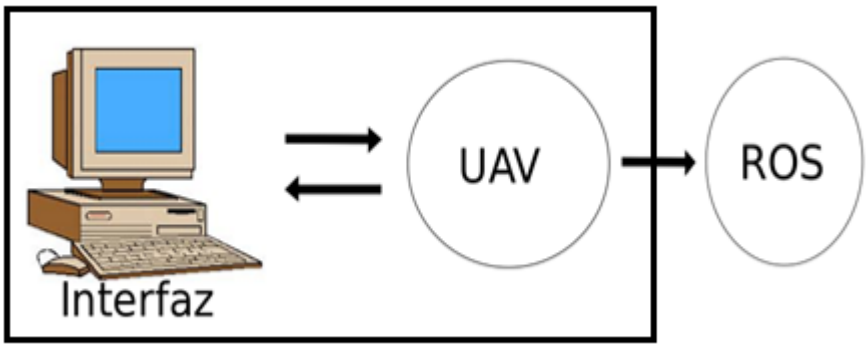

Figura 4-5 Diagrama estados de bloques

La interfaz y el controlador del UAV corresponden a clases independientes, pero implementadas en el mismo nodo, por lo que la interfaz se encarga de comunicar los cambios en los estados de funcionamientos y el otro se encarga de ejecutar las funcionalidades. El manejador de referencia es el único que se comunica con el resto del sistema para enviarle la referencia de las velocidades al nodo del control de vuelo y para recibir información del sistema (como datos de posición).

Este intercambio de información entre las 2 clases se hace a través de una serie variables que son:

- Velocidades obtenidas por teclado, será un vector de cuatro componentes para cada una de las velocidades.
- Bandera de inicio de despegue, permite pasar del estado de inicio a alguno de los otros 2.

- Bandera de conexión al control autónomo, permite pasar entre estado de control manual o autónomo.
- Posición de la referencia dinámica.

De la misma forma que se envían variables en una dirección, también es necesario poder mostrar información acerca del UAV. Las variables que recibe la interfaz son:

- Una matriz con información relacionada con el ángulo del UAV y del objetivo. En caso de control manual, el objetivo del ángulo será 0 grados.
- Valor de las velocidades, en caso de que se esté en control autónomo es necesario enviar las velocidades para mostrarlas.

4.1.3 Comandos por terminal

La interfaz anterior deja la funcionalidad en ocasiones limitada, ya que no es posible realizar determinadas acciones sin modificar el código como:

- Seleccionar la altura en la que se despega o cambiar la altura a la que se mueve de forma dinámica.
- Determinar la posición final exacta, que no sea exclusivamente pulsando en una matriz y que solo permite usar números enteros.

Para poder solventar estos inconvenientes, sin modificar el código, se han introducido una serie de terminaciones que permiten variar ciertos parámetros, y que permiten solucionar los problemas anteriormente mencionados.

La altura a la que se despega por defecto es de 1.5 metros. Para poder modificarlo se debe añadir a la hora de ejecutar el programa ‘h *num*’, siendo ‘num’ el valor numérico al cual quieres que se despegue.

Para cambiar la referencia a donde se quiere que llegue el dron se debe añadir ‘x *num*’ para modificar la posición x final, ‘y *num*’ para la y, y ‘z *num*’ para la z final. Por defecto estos valores serán nulos para las tres componentes. Modificar cualquiera de estos 3 valores implica comenzar el control autónomo, por tanto, se despegará e irá a la posición de destino. Si se utiliza esta opción, no se podrá ni cambiar la referencia final ni cambiar al modo de manejo manual. De manera independiente se seguirá mostrando la información a través de la aplicación del punto anterior.

Finalmente, si se escribe ‘takeoff’ se efectuará directamente el despegue sin necesidad de pulsar el botón correspondiente de la interfaz, pasando directamente al modo de control manual.

Notar que si solo se usan los comandos de h y/o ‘takeoff’ se puede cambiar de opciones entre control manual y automático. Además, todos los comandos se pueden escribir en cualquier orden incluso mezclando de distinto tipo,

4.2 Red neuronal

Para poder navegar por el entorno sin colisionar con ninguno de los objetos se ha usado una red neuronal capaz de proporcionar las velocidades lineales en los ejes (x, y, z) y la velocidad angular en yaw.

4.2.1 Entradas

En los distintos experimentos se han obtenido datos sobre el UAV y su entorno los cuales tras ser procesados se han introducido en la red. Este procesado tiene como objetivo disminuir el número de componentes que forman la red, haciéndola menos pesada y disminuyendo el tiempo de entrenamiento, y aportar la mayor información acerca del entorno con la mínima cantidad de datos.

4.2.1.1 Imágenes

Para ser capaces de esquivar cualquier elemento que se encuentre en la trayectoria entre el UAV y la posición objetivo, es necesario poder percibir el entorno a través de algún sensor. Para ello se ha hecho uso de una cámara de profundidad, la cual proporciona información acerca de la distancia a la que se encuentran los objetos a nuestro alrededor.

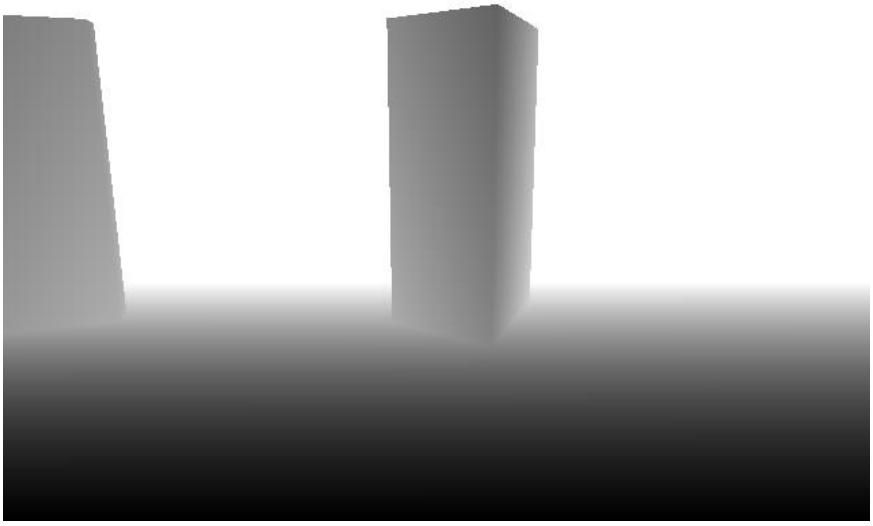

Figura 4-6 Imagen de profundidad

La imagen obtenida de los experimentos es una imagen en blanco y negro, la cual indica la distancia en metros hacia el punto en cuestión. Como el rango de valores es entre 0 y la longitud máxima de percepción de la cámara, se ha preferido cambiar los límites para que coincidan con los de una imagen estándar, con valores entre 0 y 255. Además, los puntos que no es capaz de observar porque se encuentran muy lejos o muy cerca, se dan por defecto con un valor nulo, por lo que es necesario modificarlos ya que no es posible introducírselo a la red.

Finalmente, la imagen tomada es de un tamaño de 640x480, una imagen tan grande no aporta información relevante y por tanto se ha re-escalado para disminuir el tamaño de la red. Así se ha reducido la imagen a una de tamaño 120x100, además, se ha dividido entre 255 para que los valores a introducir en la red estén entre 0 y 1.

4.2.1.2 Posiciones

Para poder llegar a una posición determinada se hace indispensable conocer, dónde está esta posición final y donde nos encontramos en cada instante. Así, se irán aproximando ambas posiciones hasta que la diferencia sea nula.

Para evitar introducir los 6 parámetros correspondientes a las coordenadas (x, y, z) de cada una de las posiciones, se ha obtenido el vector que relaciona la posición del objetivo respecto la posición del UAV. Así se disminuiría a la mitad el número de componentes introducidas en la red.

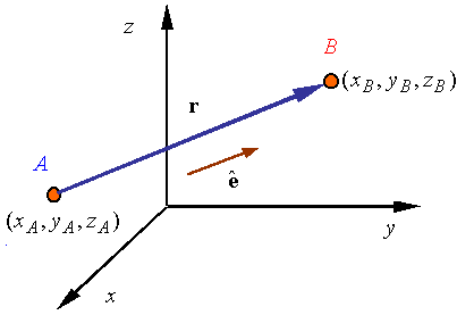

Figura 4-7 Posiciones propia y objetivo

En este dibujo el punto A correspondería a nuestra posición y el punto B a la posición objetivo. Las coordenadas del vector que une ambos puntos.

$$\begin{bmatrix} x_{so} \\ y_{so} \\ z_{so} \end{bmatrix} = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} - \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix}$$

Aun así, estas coordenadas no nos darían información suficientemente buena, ya que, esta diferencia tiene como referencia las coordenadas globales y, por tanto, no tiene en cuenta la dirección a la que apunta el dron. Haciendo un giro a este vector de dirección respecto del dron se puede obtener información de mayor utilidad, ya que tiene en cuenta la dirección a la que apunta el dron.

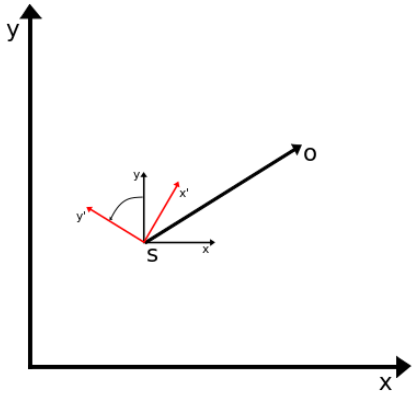

Figura 4-8 Cambio de referencia

$$\begin{bmatrix} x'_{so} \\ y'_{so} \\ z'_{so} \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_{so} \\ y_{so} \\ z_{so} \end{bmatrix}$$

La posición z no se verá alterada, ya que no depende del giro del UAV.

4.2.1.3 Ángulo

A pesar de que las componentes de posición proporcionadas en el subapartado anterior contengan cierta información sobre el ángulo, es útil introducir componentes respecto este valor. Se podrían destacar 2 ángulos distintos, el del objetivo respecto del UAV y el del propio UAV.

Introducir los ángulos de forma independiente no da una información verdaderamente útil. Ya que serían valores entre -π y π, los cuales pueden tener dificultades a la hora de interpretar la diferencia entre ambos,

debido a que existen grandes saltos (de π a - π). Para evitar esto se ha obtenido la diferencia entre los ángulos, siendo negativa si está hacia la derecha y positiva si está para la izquierda. Esto nos da una información más precisa que la anteriormente mencionada. Debido a que no se necesita una precisión demasiado elevada, se ha codificado en un vector de 5 componentes que indicará si está muy a la izquierda, algo a la izquierda, centrado, algo a la derecha o muy a la derecha.

Esta codificación se ha hecho con valores introducidos manualmente, los cuales son,

Tipo	*Rango de ángulos (º)*	*Codificación*
Muy a la izquierda	-180, 90	1, 0, 0, 0, 0
Algo a la izquierda	-90, -20	0, 1, 0, 0, 0
Centrado	-20, 20	0, 0, 1, 0, 0
Algo a la Derecha	20, 90	0, 0, 0, 1, 0
Muy a la derecha	90, 180	0, 0, 0, 0, 1

Tabla 4-2 Codificación ángulos

4.2.1.4 Distancia

Una vez se aproxima el UAV lo suficiente a la posición objetivo, deja de tener tanta relevancia algunos de los parámetros descritos. Ya que por ejemplo si nos encontramos una posición cercana al objetivo interesará hacer movimientos más suaves, puede no ser necesario esquivar ciertos objetos y no es necesario girar para situarse en la posición final.

Este valor se calcula realizando el módulo al vector que une los puntos inicial y final.

$$\mathrm{D} = \sqrt[2]{x_{so}{}^2 + y_{so}{}^2}$$

Ejemplo 4–1. *Imagínese que el UAV se encuentra en la posición (2, 3, 1.5) y el objetivo es (10, -4, 3). Además, el ángulo del dron es de 2.300 radianes.*

En primer lugar, habría que calcular el vector que une a ambos vectores usando la ecuación, se obtendría el siguiente vector.

$$v_{so} = \begin{bmatrix} 10 \\ -4 \\ 3 \end{bmatrix} - \begin{bmatrix} 2 \\ 3 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 8 \\ -7 \\ 1.5 \end{bmatrix}$$

Después, habría que girar este vector para hacerlo coincidir con la dirección que mantiene el dron en ese momento. Con ψ = 2.300 radianes.

$$v'_{so} = \begin{bmatrix} cos\,\psi & -sin\,\psi & 0 \\ sin\,\psi & cos\,\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 8 \\ -7 \\ 1.5 \end{bmatrix} = \begin{bmatrix} -0.11 \\ 10{,}63 \\ 1.5 \end{bmatrix}$$

Para calcular la distancia hacia el objetivo se debe hacer el módulo al vector anterior (únicamente x-y). Por tanto,

$$D = \sqrt[2]{8^2 + (-7)^2}$$

Finalmente se calcula el ángulo hacia el objetivo. Para ello hay que calcular antes el ángulo objetivo el cual se calcula como

$$ang = atan2(y_{so}, x_{so})$$

Este ángulo se comparará con el yaw del UAV.

4.2.2 Salidas

Las salidas corresponderán a los valores de las velocidades lineales deseadas en (x, y, z) y la velocidad angular en yaw. Cada una de estas 4 salidas está compuesta de 5 componentes, las cuales indicarán si el valor es muy negativo, algo negativo, nulo, algo positivo o muy positivo.

Cada una de las salidas es independiente al resto y su salida es resultado de usar a la salida una función softmax. Por tanto, se tendrán cinco valores que indican la probabilidad de que tome cierto valor. Así se podrá conseguir un valor continuo en el rango desde muy negativo a muy positivo, viendo las salidas como una campana de probabilidad. Esto nos da un mayor rango con el que trabajar, debido a que, si se utiliza el argumento máximo para las velocidades únicamente se tendrían 5 niveles de velocidades, de esta forma se tendrá el rango completo.

Cuando se refiere a valores algo positivos, muy negativos, …, se refiere a valores que se dan posteriormente mediante una tabla con la cual se hace una suma ponderada de los valores de velocidad correspondientes a cada una de las etiquetas. Entonces ya se podrá hablar de valores exactos de velocidad.

Ejemplo 4-2. *Pretendiendo reconstruir una velocidad a partir de las probabilidades siguientes*

P = [0.123, 0.423, 0.364, 0.072, 0.018] y las velocidades a las que corresponden serán VT = [-1.5, -0.75, 0.0, 0.75, 1.5].

La velocidad final se calcularía haciendo la suma ponderada de estas velocidades, (suponemos que se trata de la componente x de la velocidad).

$$V_x = 0.123 * -1.5 + 0.423 * \ -0.75 + 0.072 * 0.75 + 0.018 * 1.5 = 0.421\ m/s$$

Si en lugar de este método se utilizara el argumento máximo la velocidad sería de 0.75m/s.

Sin embargo, para entrenar a la red se ha usado la codificación one hot para las velocidades. Como las velocidades que se obtienen de los experimentos están en valores reales, por tanto, y como en el caso de los ángulos, es necesario codificarlo para convertirlo en un vector de 5 componentes. La conversión seguirá la lógica siguiente:

Tipo	*Rango de velocidades (m/s)*	*Codificación*
Valor muy negativo	-1.5, -0.75	1, 0, 0, 0, 0

Algo negativo	-0.75, -0.1	0, 1, 0, 0, 0
Nulo	-0.1, 0.1	0, 0, 1, 0, 0
Algo positivo	0.1, 0.75	0, 0, 0, 1, 0
Valor muy positivo	0.75, 1.5	0, 0, 0, 0, 1

Tabla 4-3 Codificación velocidades

4.2.3 Estructura de la red

La red se ha estructurado con 5 entradas distintas, estas son

- Imágenes
- Posición x-y
- Posición z
- Ángulo
- Distancia.

Además, se tiene 4 salidas independientes,

- Velocidad en x
- Velocidad en y
- Velocidad en z
- Velocidad angular en ψ

En la estructura se ve como primero se unen todas las entradas son procesadas por alguna capa de neuronas y/o convolucional. Después, se han unido para procesar toda esta información junta y finalmente se han separado para cada una de las salidas.

Para favorecer cierta información interesante para determinadas salidas, se ha recuperado cierta información sobre los parámetros antes de que se junten para ser procesados. Esto se ha realizado porque existen entradas con un valor mayor en ciertas salidas. Así, para las velocidades x-y serán importante datos, como la posición, la cámara o la distancia al objetivo. Para la velocidad en z, no interesa la cámara ya que no existirán obstáculos en ese eje (además de que no se podrían captar), por tanto, se usará la posición z y la distancia hacia el objetivo. Finalmente, para la velocidad angular importan parámetros como la diferencia en el ángulo hasta el objetivo, la cámara y la distancia al objetivo.

La estructura de la red es la siguiente,

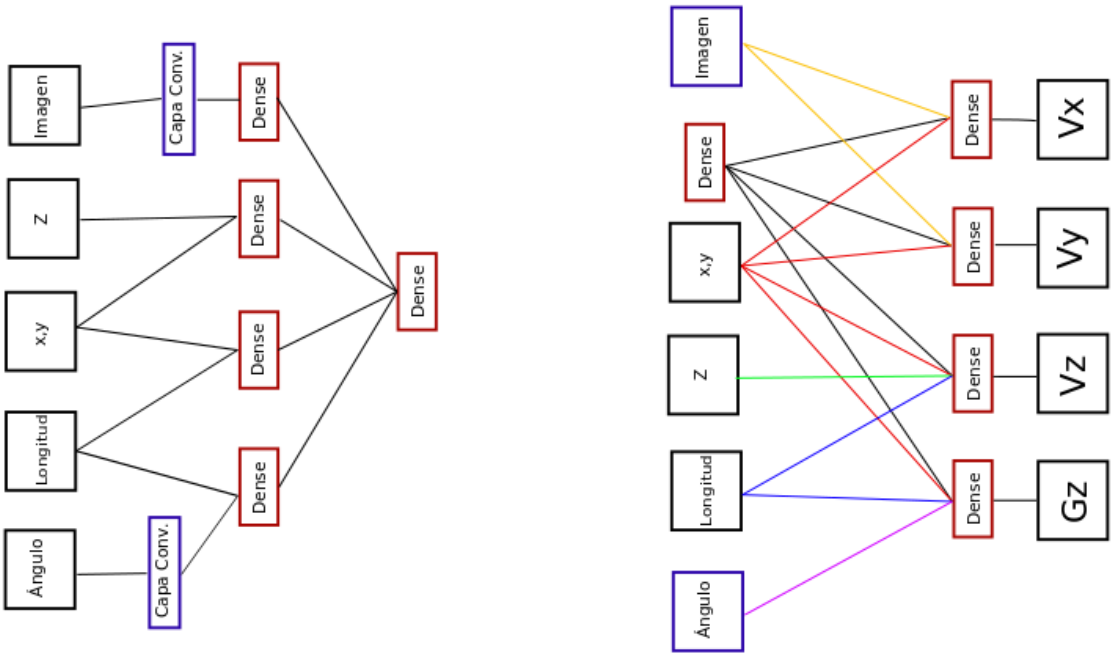

Figura 4-9 Estructura de la red simplificada

Este esquema muestra lo comentado anteriormente, existe una fusión de todos los datos para ser procesados. Posteriormente cada uno de ellos se vuelve a influir por información menos procesada en caso de necesitar hacer algún cambio. Keras también tiene una herramienta para mostrar cual sería la estructura de la red, pero al tener tantos bloques es complicado de comprender, este esquema más detallado se encuentra en el anexo del presente proyecto.

4.2.4 Creación de entornos

Para poder crear entornos con los que el UAV pueda interaccionar, se ha hecho uso de una de la herramienta de Gazebo, building editor, el cual permite construir paredes. A diferencia de los objetos como cubos o cilindros que se pueden introducir normalmente y se puede variar su tamaño, los modelos construidos con el building editor son totalmente estáticos. Esta característica será bastante práctica, ya que con los cubos y otros elementos se podrían mover de posición en caso de que se colisione con ellos.

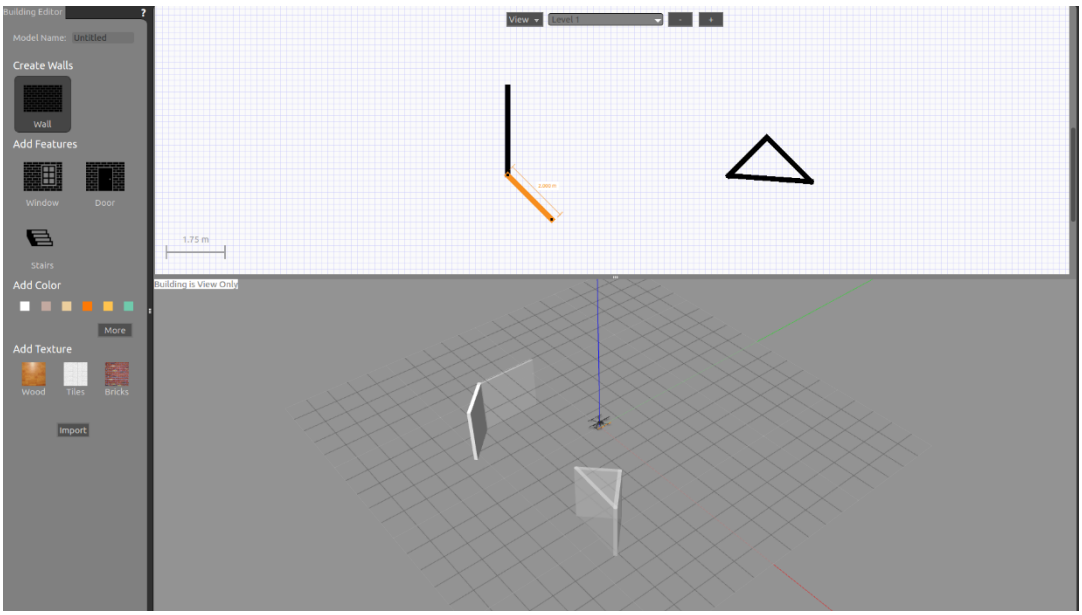

Figura 4-10 Building editor Gazebo

Además, permite introducir varios obstáculos simultáneamente, modificando la altura, ancho y grosor de las paredes. También permite construir nuevos pisos en las estructuras y por tanto techos para los estos.

Otra característica es que se pueden guardar estos modelos que se creen. Esto será de utilidad, sobre todo cuando se quieren entornos con gran cantidad de objetos. Por tanto, estos escenarios se podrían usar para ver cómo funcionan distintas estructuras frente el mismo problema.

Los objetos básicos con los que se ha trabajado son:

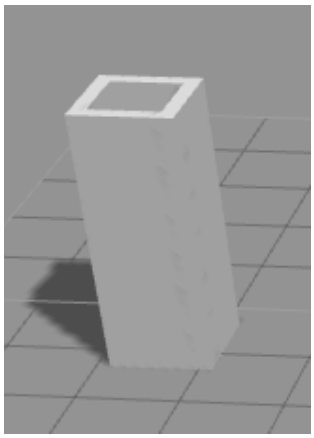

Figura 4-11 Columna

Una columna cuadrada de 1 m y 2.5 metros de alto.

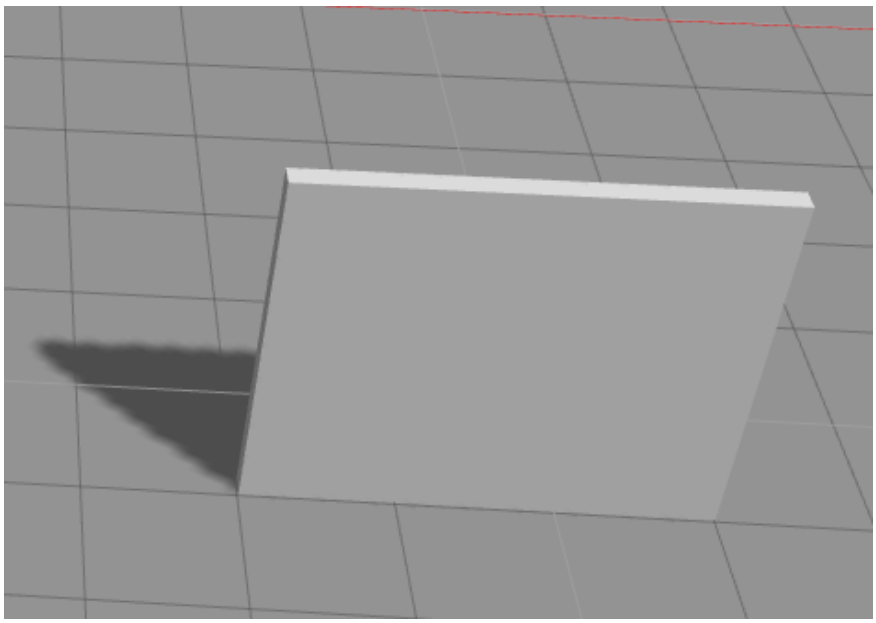

Figura 4-12 Pared

Una pared de 3 metros de largo y 2.5 metros de alto.

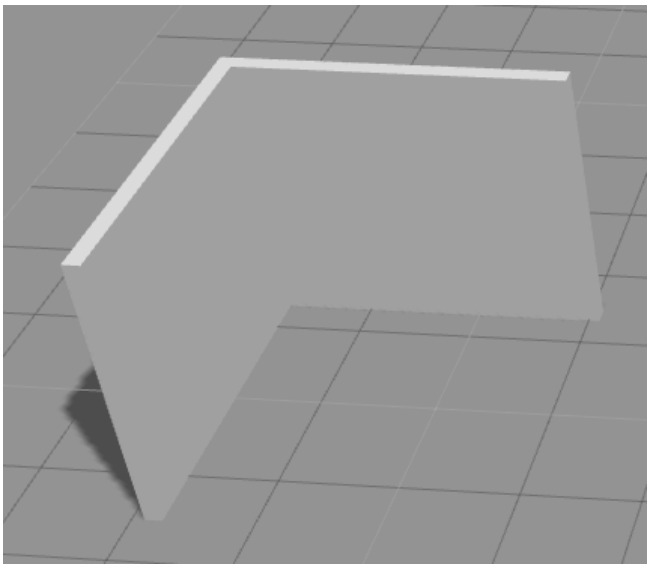

Figura 4-13 Esquina

Una esquina de 3 metros de largo por cada cara, formando un ángulo de 90º y con 2.5 metros de alto.

4.2.5 Experimentos

Para entrenar la red neuronal es necesario hacer una serie de experimentos para obtener las entradas y salidas con la que se entrenará la red. Para ello se ha usado el modo manual para navegar por un entorno, guardando los datos recogidos en cada momento.

La posición (x, y, z, ψ) se ha guardado en un archivo de texto llamado 'pose.out' en el que se guardan en cada instante los 4 parámetros en líneas separadas por espacio. Estos valores están en metros y se han guardado con 3 decimales.

Las velocidades lineales (x, y, z) y la angular de yaw se han guardado en un archivo de texto llamado 'velocity.out', en el que se guardan las velocidades en cada uno de los ejes separados por espacios en líneas separadas. Estos corresponden a los valores dados por el teclado. Un ejemplo sería

Las imágenes se obtienen de la cámara de profundidad, como se explicó anteriormente es necesario realizar una serie de transformaciones a estas imágenes. Sin embargo, para guardarlas únicamente se ha re-escalado cada uno de los valores para que esté entre 0 y 255. Finalmente se han guardado todas las imágenes en una carpeta llamada 'depth_images' la cual contiene las imágenes guardándose cada una en formato .png y cuyo nombre es un número que se asigna cronológicamente.

Todos estos archivos generados se han ido guardando en carpetas diferentes para poder acceder a todos los datos de forma más simple. Quedando pues, una carpeta donde se incluyen los valores de velocidad y posición, y una carpeta con las imágenes.

Los experimentos se han dividido en distintas categorías para que sea más fácil localizar y reemplazar algunos sets de experimentos. Así se dividirán en distintas categorías en las que su nombre será EXXX_Y, donde XXX corresponde al número de experimentos y la Y la categoría (caracteres). Los experimentos consistirán en lo siguiente.

A. En un entorno en el que no hay obstáculos se gira hacia izquierda o derecha, para mirar hacia el objetivo y posteriormente avanzar recto hacia el objetivo. Una vez en la posición (x, y) objetivo se

eleva o desciende hacia la posición z deseada.

B. En un entorno en el que se ha colocado una columna se gira a izquierda o derecha hasta posicionarse mirando al objetivo. Posteriormente se avanza, y cuando localiza la columna la esquiva a derecha o izquierda. Una vez sobrepasada, se gira para reposicionar el objetivo en frente y avanza hasta él. Finalmente se eleva o desciende hasta la posición final.

C. Se realiza lo mismo que en los experimentos de B, pero en lugar de una columna se ha colocado una pared.

D. En línea recta se colocan distintos obstáculos algo hacia los lados. Así se desplazará a izquierda o derecha para evitar dichos obstáculos, estos se sobrepasarán por el lado que tenga que recorrer menor distancia. Una vez sobrepasados gira para posicionar el objetivo en frente y avanza hacia él. Finalmente se eleva o desciende hasta la posición final.

E. En un entorno sin nada se eleva o desciende y se intenta mantener el dron en una posición.

F. En un entorno con múltiples obstáculos se gira el dron para que mire a la posición objetivo y avanza esquivando los posibles objetos que se encuentre en el camino.

G. Se realiza lo mismo que en F, pero con menor cantidad de obstáculos para que se pueda mover más libremente.

H. Se realiza lo mismo que en E, pero con distintos obstáculos a la vista.

Como ya se sabe, es necesario tener en todo momento un objetivo ya que este valor se utiliza a la hora de calcular las entradas de la red. Al hacer los experimentos manualmente, es difícil saber de antemano cual será la posición final exacta. Para aproximar ese valor objetivo se ha usado el último valor del archivo de posición generado. Realmente este valor tampoco es necesario que sea excesivamente preciso, por tanto, se ha redondeado al medio metro más cercano.

4.3 Navegación del UAV

Como se ha explicado anteriormente existen 2 funcionalidades diferenciadas, la primera el control por teclado y la segunda el control mediante una red neuronal. A continuación, se pasará a explicar el funcionamiento desde el punto de vista del UAV.

4.3.1 Navegación por teclado

Para poder comenzar la navegación por teclado se puede bien hacer uso del botón de despegue o añadiendo ‘takeoff’ a la hora de ejecutar el programa desde la terminal.

Cuando se empieza a despegar se inicia el servicio de despegue que automáticamente eleva al dron hasta una cierta posición (modificable a través de parámetros por la terminal). Cuando se haya terminado este servicio el color del botón de despegue cambiará de color a verde.

La forma en la que se mueve el UAV consiste en lo siguiente, la interfaz es la que gestiona los botones y por ende la envía estas velocidades al UAV. Este será el que las envía al controlador de vuelo para que las tome como referencia para adaptar la velocidad. Para poder enviar estas velocidades es necesario contactar con el nodo que gestiona el control de las velocidades. Los parámetros que se envíen serán los de velocidad.

Para hacer esta conexión, será necesario publicar mensajes en el topic ‘ual/set_velocity’. Estos mensajes serán del tipo TwistStamped y en ellos se publican las velocidades en x, y, z, y yaw. Hay que destacar que estas velocidades que se envían están en coordenadas globales. Como las velocidades están dadas respecto del

UAV, será necesario girar estas velocidades.

4.3.2 Navegación autónoma

Para poder comenzar la navegación autónoma se puede hacer manualmente conectando la red con el botón asignado para ello y luego pulsando el botón para despegar el UAV. Otra forma de hacer sería introducir las coordenadas desde la terminal. De esta última forma no sería necesario despegar el UAV, sería automático.

El UAV debe tener en todo momento una posición objetivo para ir hacia ella. Esta referencia será por defecto la posición (0, 0, 0), y esta puede ser modificada. Si se elige la forma manual, la referencia se podrá cambiar de forma dinámica en un rango limitado, eso sí, sin poder cambiar la altura final. Por el contrario, si se introduce por la terminal se podrá introducir cualquier posición, pero no se podrá cambiar en ningún momento.

Con estas entradas, entre las que se encuentra la referencia, se irá hacia el punto final. Para moverse hasta el punto final se girará hasta alinearse con este punto, después se moverá hacia delante en línea recta hasta la posición objetivo. En caso de encontrarse con algún obstáculo en mitad del camino, se desplazará hacia alguno de los lados, generalmente hacia el lado que esté más cerca. Por lo que, si ve que el obstáculo esta ligeramente a la izquierda, se desplazará a la derecha para sobrepasarlo y viceversa. Una vez sobrepasado el obstáculo, girará para volver a alinearse con el objetivo y avanzar hasta llegar a este. Finalmente, cuando llegue a la posición objetivo se elevará o descenderá para colocarse a la altura final. Una vez aquí, se moverá para ajustarse a la posición final.

De la misma forma que en el apartado anterior, para que el dron se pueda mover será necesario enviarle las velocidades al UAV para que las tome como referencia. Por tanto, se usará la clase 'TwistStamped' para enviar estas velocidades. También será necesario hacer el cambio de referencia de estas velocidades para que se traduzcan al marco global. Hay que recordar que la red únicamente da probabilidades de la velocidad que se debe seguir, por tanto, será necesario convertir estas probabilidades en velocidades a través de una tabla de conversión.

Finalmente, cabe destacar que la red necesita de cierta información que introduce en esta para que sacar las probabilidades de velocidad. Esta información será, la imagen de profundidad, posición hacia objetivo, longitud y diferencia en el ángulo. Todas estas informaciones han sido procesadas para poder introducirlas a la red y para obtenerla se ha tenido que subscribir a distintos topics.

Para obtener el valor de la posición se ha subscrito a un topic llamado 'ual/pose' del cual se leen mensaje del tipo 'PoseStamped'. Con este se obtiene la posición del UAV en cada momento.

Respecto la información de la cámara, se ha subscrito al topic 'mbzirc_1/camera_0/depth/image_raw', del cual se lee una imagen la cual mediante una función se convertirá para poder usarla con openCV.

Igual que en el control manual, las velocidades se publicarán en el topic 'ual/set_velocity'. Las velocidades que devuelve la red también tienen como referencia la dirección del UAV, y deberán de girarse antes de ser enviadas.

5 RESULTADOS

5.1 Entrenamiento

Para entrenar a la red se ha hecho uso de los datos obtenidos a través de los experimentos. Realmente para algunos casos no es necesario usar todos los datos adquiridos, pero los resultados mostrados a continuación responden a la utilización de todos al mismo tiempo.

En total se han adquirido alrededor de 15000 pasos con los cual se entrenará la red, cada uno de estos están compuestos por la imagen de profundidad, posición y velocidad. Estos pasos corresponden a unos 25 minutos de vuelo del UAV. De todos estos pasos se han sacado todas las entradas y salidas como bien se explicó en el capítulo anterior. En todo este tiempo de entrenamiento, existen distintos valores de las velocidades que se pretende alcanzar, es casi imposible poder equilibrar cada uno de estos valores y, por consiguiente, es comprensible que algunos valores tengan pocas muestras. En la siguiente tabla se muestran las muestras que hay de cada tipo.

	0 >>	*0 >*	*~ 0*	*0 <*	*0 <<*
Velocidad x	14	439	9458	1970	3075
Velocidad y	335	962	12978	428	253
Velocidad z	576	200	13578	223	370
Velocidad ψ	265	495	13395	476	325

Tabla 5-1 Tipos de muestra

Parece lógico que, si el objetivo es llegar a un punto determinado, la mayoría de las velocidades en su eje x correspondan a valores positivos (avance), ya que los negativos se reservarán para cuando ajuste su posición. Por otro lado, si se avanza de frente la velocidad 'y' es normal que se mantenga en valores próximos a 0, algo parecido pasa con la velocidad en z y la velocidad en yaw.

Estos antes de ser introducidos en la red para entrenarlos se han mezclado de forma aleatoria para mejorar el resultado. Los principales parámetros con los que se ha entrenado la red son la tasa de aprendizaje el cual tiene un valor de 0.001, cada batch contiene 16 pasos y se ha entrenado en 5 etapas. Tras ser entrenado estos han sido los resultados.

Real \ Res	*0 >>*	*0 >*	*~ 0*	*0 <*	*0 <<*
0 >>	0.286	0.714	0	0	0
0 >	0.011	0.223	0.765	0	0

~ 0	0	0.004	0.94	0.043	0.013
0 <	0	0	0.304	0.45	0.456
0 <<	0	0	0.02	0.063	0.917

Tabla 5-2 Velocidades x

Real \ Res	*0 >>*	*0 >*	*~ 0*	*0 <*	*0 <<*
0 >>	0.776	0.155	0.069	0	0
0 >	0.018	0.314	0.668	0	0
~ 0	0	0.005	0.992	0.003	0
0 <	0	0	0.535	0.374	0.091
0 <<	0	0	0.016	0.13	0.854

Tabla 5-3 Velocidades y

Real \ Res	*0 >>*	*0 >*	*~ 0*	*0 <*	*0 <<*
0 >>	0.738	0	0.262	0	0
0 >	0.34	0	0.66	0	0
~ 0	0.011	0	0.979	0	0.012
0 <	0	0	0.641	0.02	0.34
0 <<	0	0	0.235	0.011	0.584

Tabla 5-4 Velocidades z

Real \ Res	*0 >>*	*0 >*	*~ 0*	*0 <*	*0 <<*
0 >>	0.683	0.128	0.186	0	0
0 >	0.184	0.147	0.101	0	0
~ 0	0.006	0.003	0.987	0	0.004
0 <	0	0	0.905	0	0.095
0 <<	0	0	0.514	0	0.486

Tabla 5-5 Velocidades ψ

A pesar de que los resultados para algunos casos no parezcan suficientemente buenos, para poder asegurar que

funciona o no, es necesario probar y ver cómo se comporta. Las pruebas para ver la precisión de la red se han obtenido utilizando la función argmax, el cual da el argumento máximo. Al hacerlo de esta forma cabe la posibilidad de que, si se duda entre 2 opciones, si una tuviera un 1% más de probabilidad escogería esa. Al sacar la velocidad resultante de la suma ponderada de las probabilidades a la salida, puede que en un momento determinado se quiera avanzar, pero la opción que es más probable es que se encuentre parado. Al hacer la suma ponderada el valor de velocidad podrá ser positivo y avanzar en alguna dirección, por tanto, se podría decir que está funcionando correctamente.

También se debe notar que en algunos casos el número de muestras es muy reducido, por lo que es comprensible que la precisión para ciertos supuestos sea buena.

5.2 Movimiento con giro

La prueba básica para navegar en un entorno sería moverse por los ejes x, y, z hasta llegar a la posición objetivo. Esta aproximación sería la más sencilla, sin embargo, si el objetivo final es ser capaces de detectar objetos para esquivarlo, se hace necesario poder girar el UAV apuntando a la posición a la que se quiere avanzar.

En los siguientes experimentos se tiene el UAV en un entorno sin ningún tipo de obstáculo, allí se despegará a una cierta altura, se girará apuntando hacia la posición objetivo, se avanzará hacia ella y finalmente se ajustará la posición para que coincida con el objetivo.

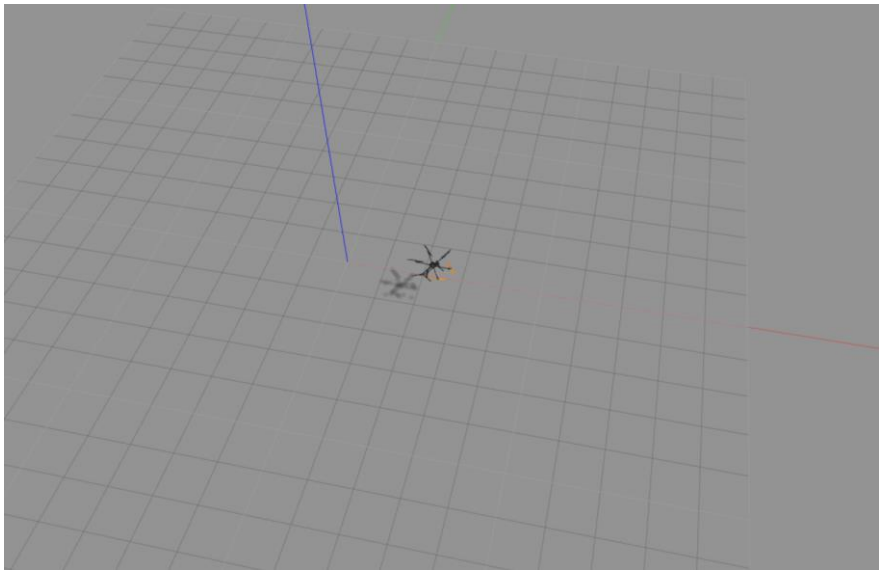

Figura 5-1 Entorno movimiento con giro

Este primer experimento se ha realizado haciendo uso de la terminal para pasar las referencias exactas.

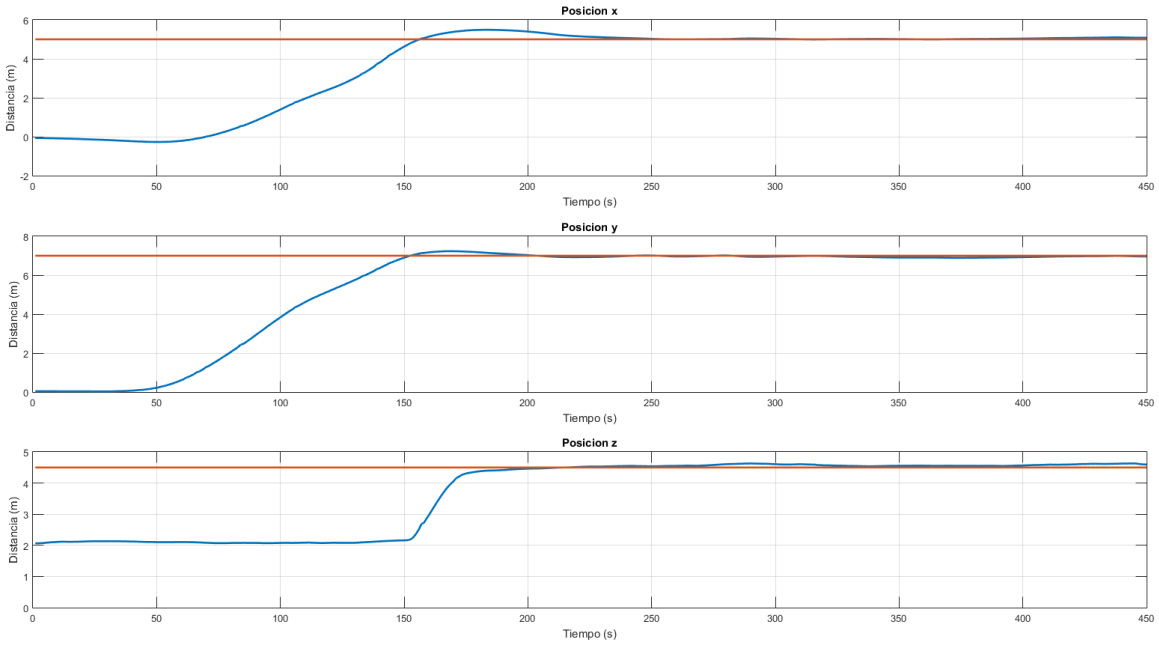

Figura 5-2 Variación posición experimento con giro

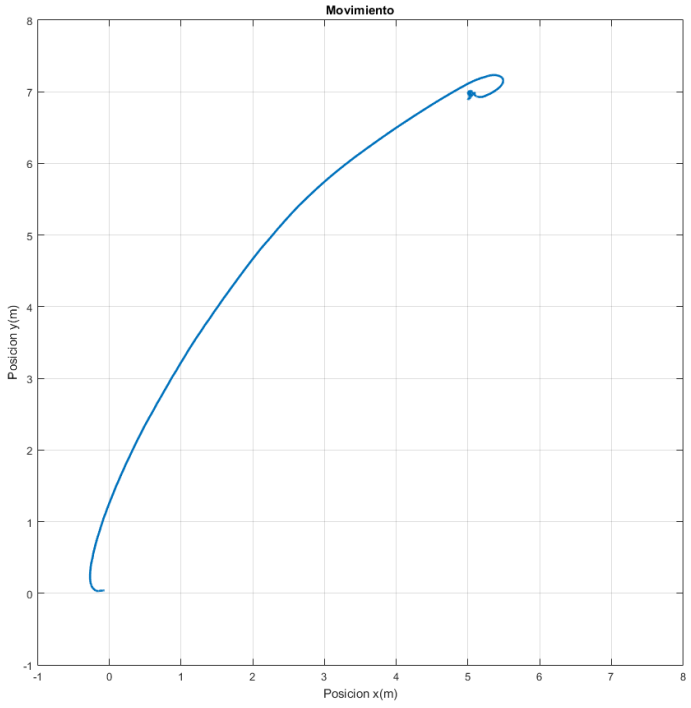

Figura 5-3 Posición experimento con giro

Desde la Figura 5-2 se observa como al principio el UAV se encuentra casi estático y posteriormente comienza a avanzar. En los momentos en los que está casi estático, realmente se está girando para apuntar a la posición objetivo. Cuando se punta a esta posición se avanza a máxima velocidad hasta alcanzar la posición x-y objetivo. Una vez allí se eleva en la posición z y después modifica su posición hasta alcanzar la posición de destino. De la figura también se puede comprobar cuando una vez que aproximadamente llega, la posición x se

corrige hasta ser la posición final.

En la Figura 5-3 se puede observar esta corrección, cuando se pasa del punto final y se mueve hacia atrás para llegar a la posición objetivo. Después se queda estático entorno al punto.

Ahora se comprobará como funcionaría si se usara la herramienta con la que permite cambiar las referencias de forma dinámica. El entorno será el mismo ya que para este supuesto no necesita ningún obstáculo.

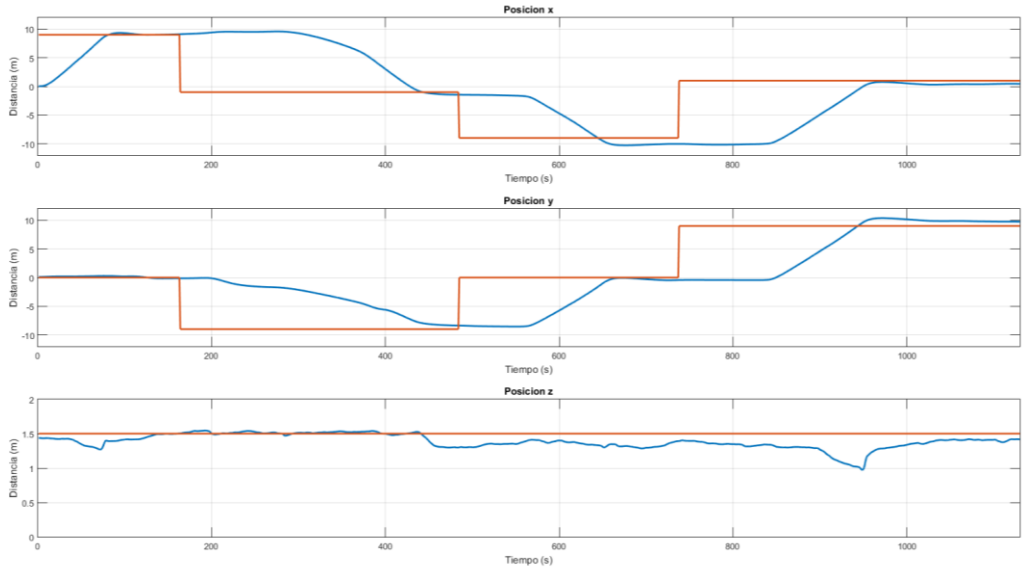

Figura 5-4 Variación posición experimento con giro 2

Aquí se aprecia como al cambiar la referencia el dron lo sigue cambiando primero su orientación y avanzando recto hasta la posición. Vemos como finalmente, aunque con algún pequeño error, se llega a la posición deseada. Además, este experimento es especialmente ilustrativo ya que se observa un fenómeno que puede alterar el comportamiento del UAV. En la Figura 5-4 se observa como la posición en z empieza a deteriorarse, por la forma en la que se han realizado los experimentos si se intenta mover a menor altitud que 1 metro comienza a tener comportamientos extraños haciendo la red más imprecisa. Además, como está entrenada para que cuando llega al punto final cambie de altitud, difícilmente podrá elevarse en mitad del camino.

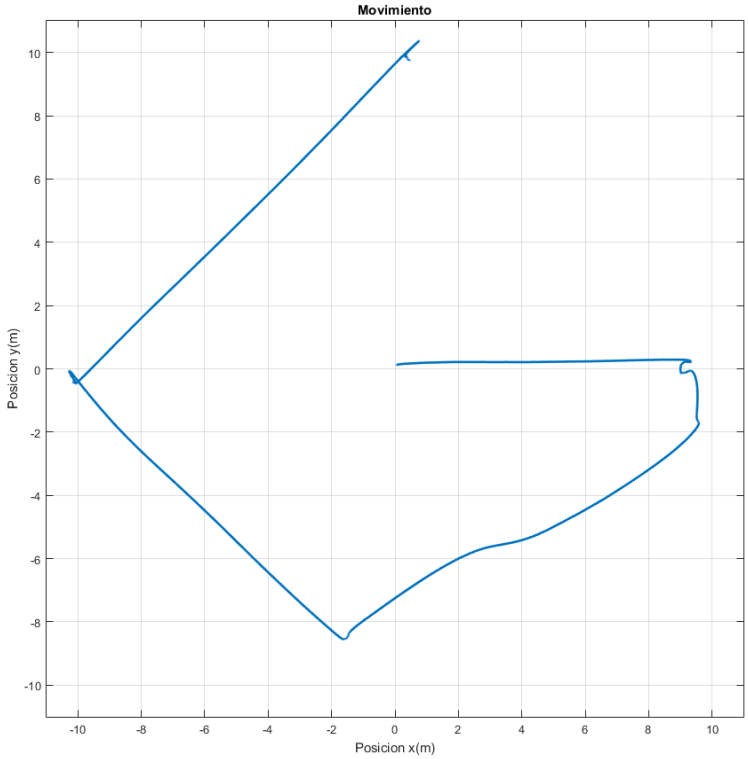

Figura 5-5 Posición experimento con giro 2

5.3 Movimiento con obstáculos

Cuando ya se ha comprobado que se puede ir hacia los objetivos mirando hacia ellos, es posible empezar a esquivar los posibles obstáculos que haya en el camino. En este supuesto se hace imprescindible la utilización de imágenes, ya que es necesario procesarlas para poder detectar obstáculos en el camino.

En la realización de los experimentos relacionados con esquivar objetos, es necesario tener en cuenta que se debe equilibrar, sobrepasarlos por izquierda o derecha. En caso contrario, podría tener tendencia a pasarlo por un único lado sin sentido alguno.

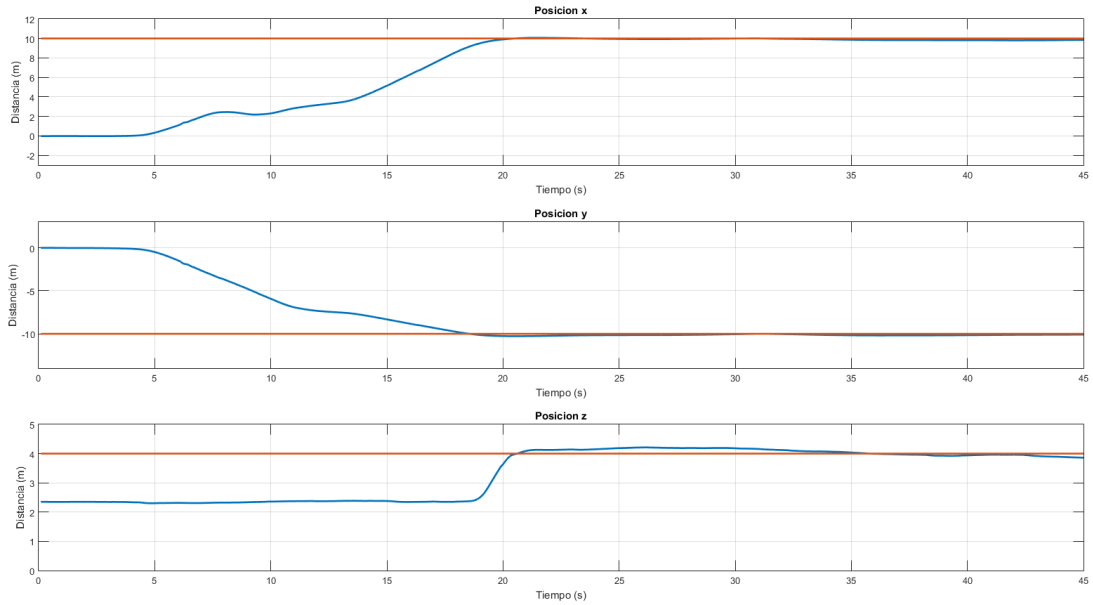

Figura 5-6 Variación posición experimento con obstáculo

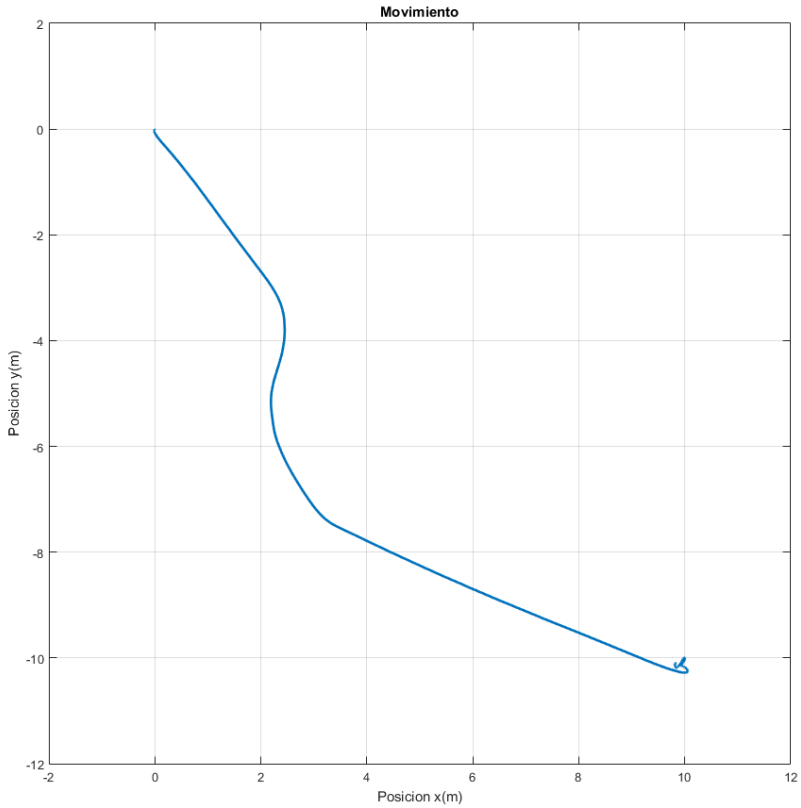

Figura 5-7 Posición experimento con obstáculo

Como en las pruebas anteriores al comienzo se gira hasta apuntar a la posición objetivo, una vez apuntando a esta dirección se avanza hasta encontrarse con un objeto en medio. En la Figura 5-7 se puede observar cómo no sigue una línea recta, sino que gira hacia la derecha y después cuando lo sobrepasa vuelve a girar, avanzando hasta la posición final. Una vez allí, se eleva hasta alcanzar la posición z. Finalmente, intenta aproximarse lo máximo posible a la posición final.

En este apartado no merece tanto la pena el cambio de referencia dinámico, ya que sería igual que

tener múltiples obstáculos lo cual correspondería al siguiente apartado.

5.4 Movimientos con múltiples obstáculos

A pesar de que se pueden esquivar sin problemas objetos individuales, en entornos con mayor cantidad de objetos es posible que las imágenes se interpreten erróneamente y que se intenten esquivar obstáculos que están en un segundo plano. Por ejemplo, si se tiene una columna y de fondo hay una pared, en caso de que tu objetivo esté en medio puede incitar al UAV a sobrepasar a la columna y a la vez intentar esquivar la pared. Por tanto, se daría un rodeo innecesario pudiendo generar colisiones por una mala interpretación.

Para evitar esto es recomendable hacer nuevos experimentos en los que se esquive objetos, pero en los que se introduzcan nuevos, para así poder clasificarlos internamente entre los obstáculos críticos y no críticos.

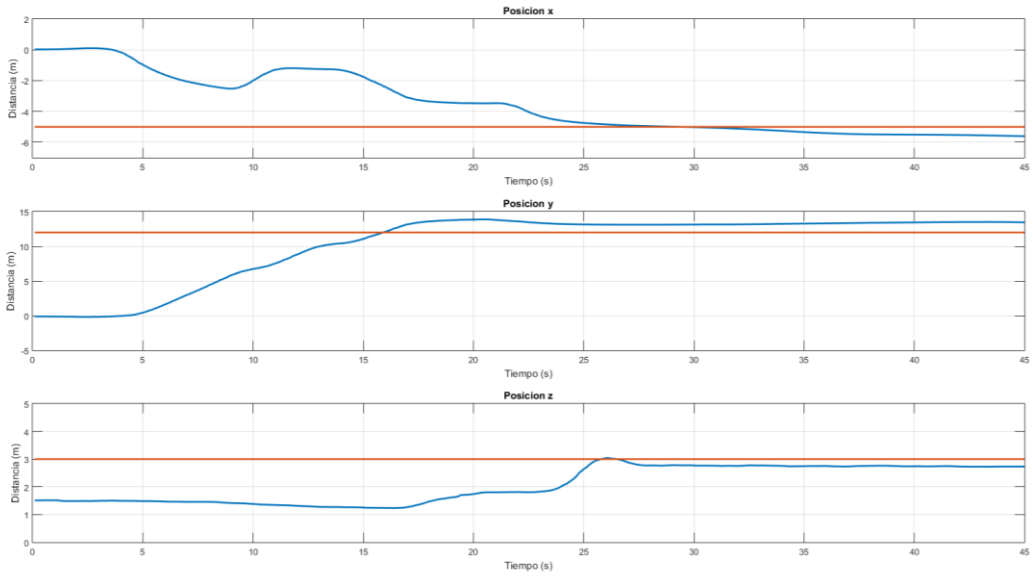

Figura 5-8 Variación posición experimentos con múltiples obstáculos

En este experimento, nuevamente hay que girar al principio para avanzar a la posición objetivo. Una vez en posición, se detecta un obstáculo y se desplaza hacia la izquierda para esquivarlo, después se sobrepasa y vuelve a mirar al objetivo. Una vez recuperada la posición se avanza, pero se encuentra con otro obstáculo el cual sobrepasa por la derecha. Finalmente se llega a la posición objetivo y se eleva el UAV hasta la posición final.

Se puede observar cómo en este caso, las posiciones tienen un error en régimen permanente mayor que en los casos anteriores. Esto es debido a que se tenga algún objeto en frente y la red lo interprete ligeramente como que no debe acercarse o que debe esquivarse y esto hace que no pueda aproximarse con más exactitud.

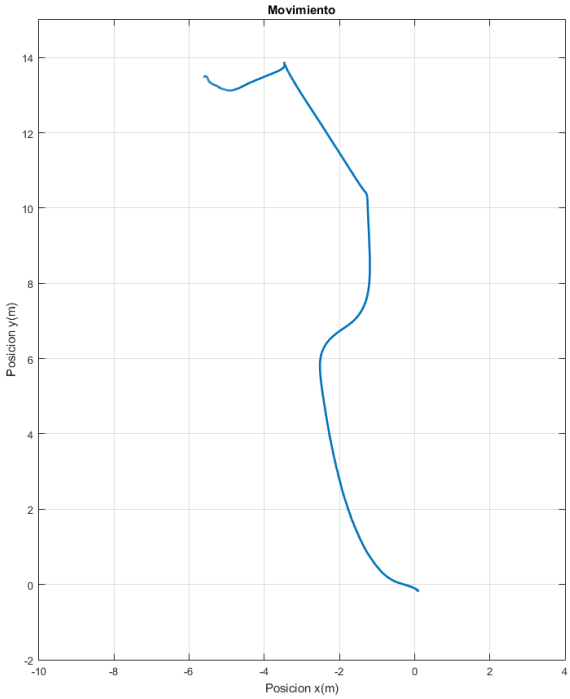

Figura 5-9 Posición experimentos con múltiples obstáculos

Ahora se observará cómo funciona con múltiples coordenadas. Para realizar el escenario donde se moverá se ha hecho uso del building editor de Gazebo, donde se ha delimitado un perímetro con distintos objetos.

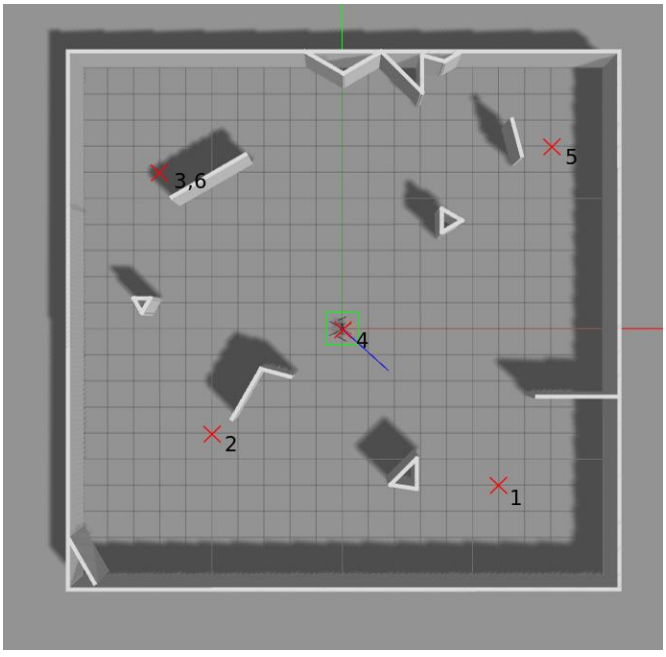

Figura 5-10 Entorno con múltiples objetos 1

Este entorno presenta ciertos obstáculos los cuales se deben sortear, en todo momento la referencia de altura será la misma.

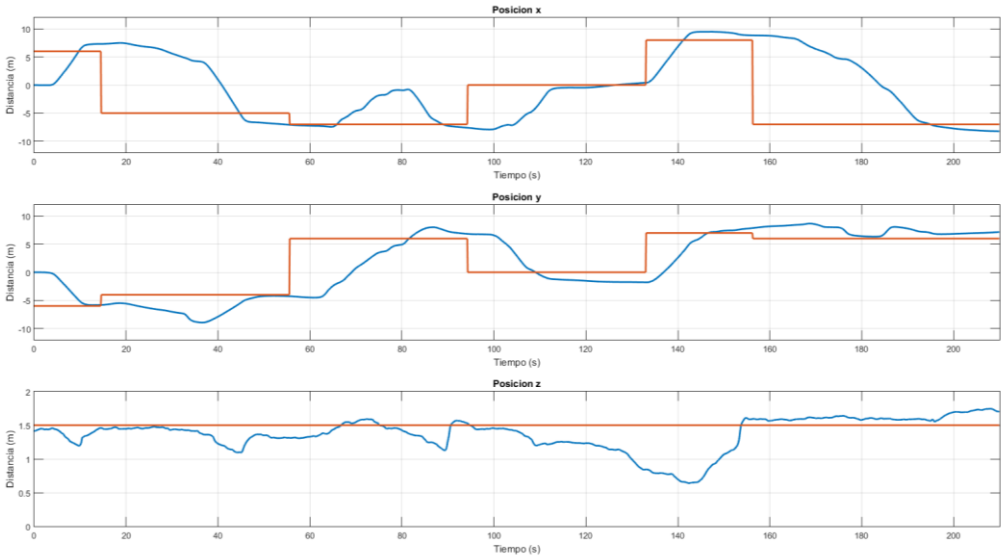

Figura 5-11 Variación posición con múltiples objetos, referencia dinámica 1

Como se puede observar cuando se cambia la referencia el UAV se mueve hacia esa posición sin demasiada dificultad, es verdad, que la precisión no parece excesivamente buena, aunque se desvíe simplemente un metro. Este error es asumible y hay que tener en cuenta que en comparación con otros experimentos la referencia tarda poco tiempo en cambiar. Respecto de la posición vemos que en algún punto ha perdido altura, pero una vez llegada a la posición final ha recuperado la posición.

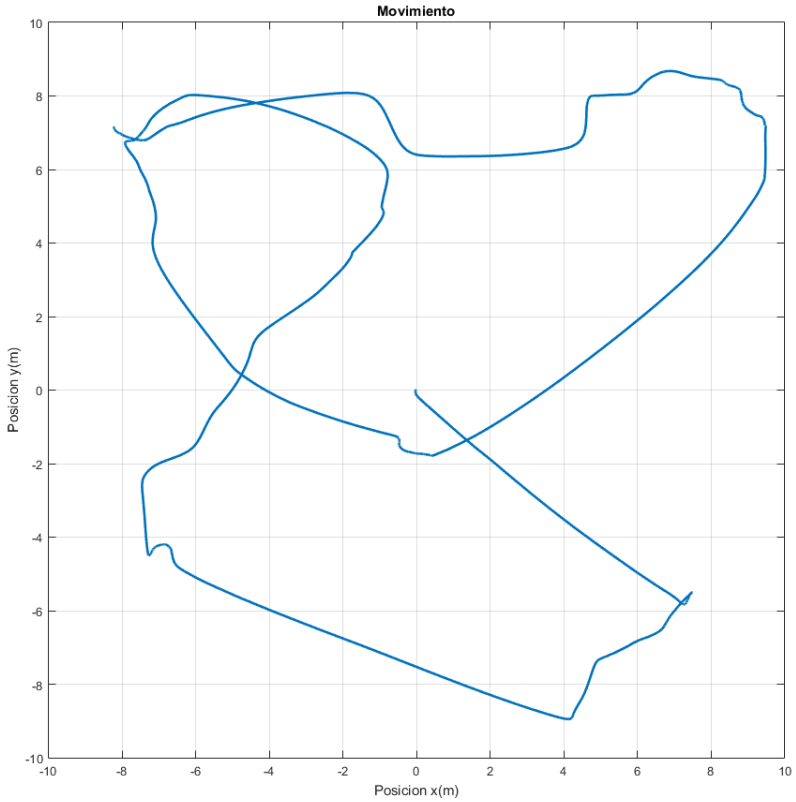

Figura 5-12 Posición con múltiples objetos, referencia dinámica 1

En esta otra imagen se ve como llega a los distintos puntos, como esquiva los objetos que existen en el camino.

A continuación, se ha realizado otro experimento de la misma tipología, pero en un entorno con una distribución de los obstáculos distinta y una mayor cantidad de ellos.

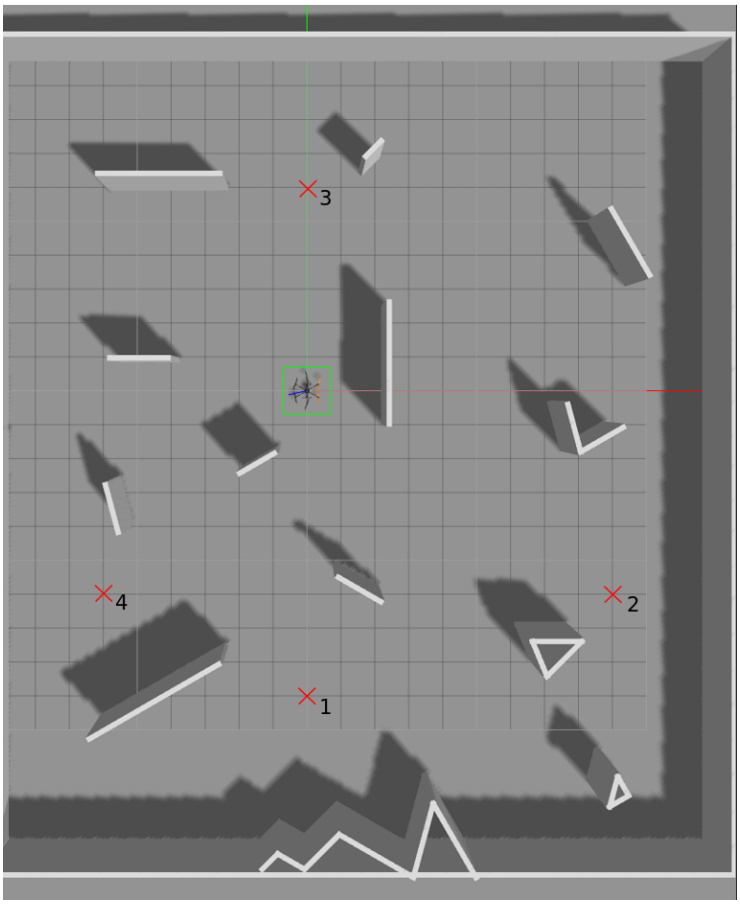

Figura 5-13 Entorno con múltiples objetos 2

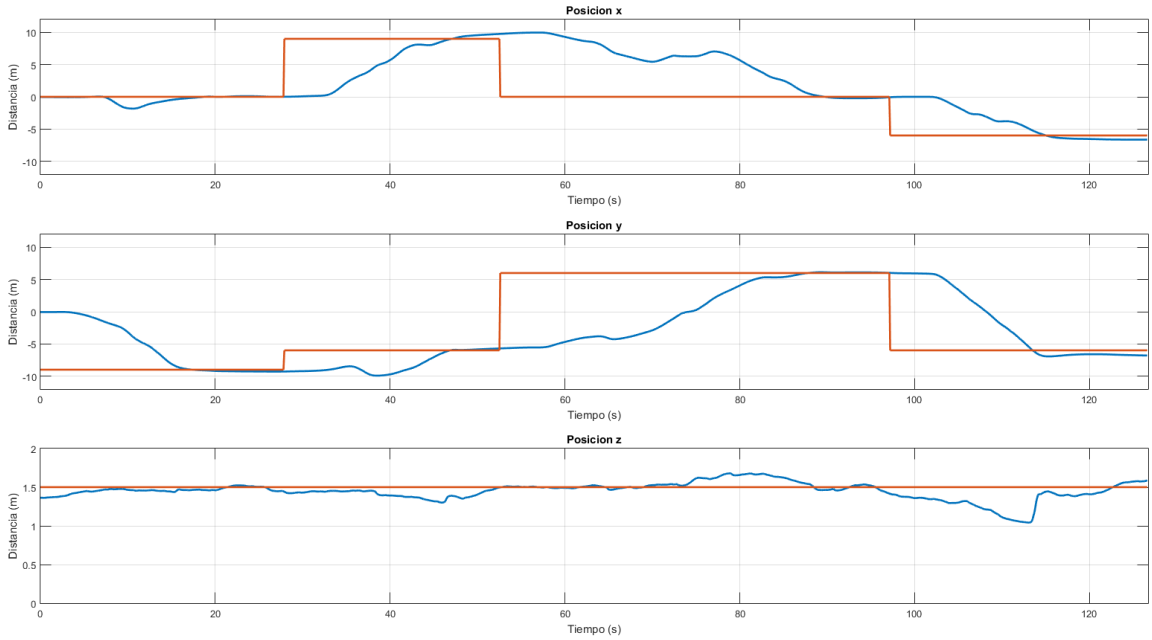

Figura 5-14 Variación posición con múltiples objetos, referencia dinámica 2

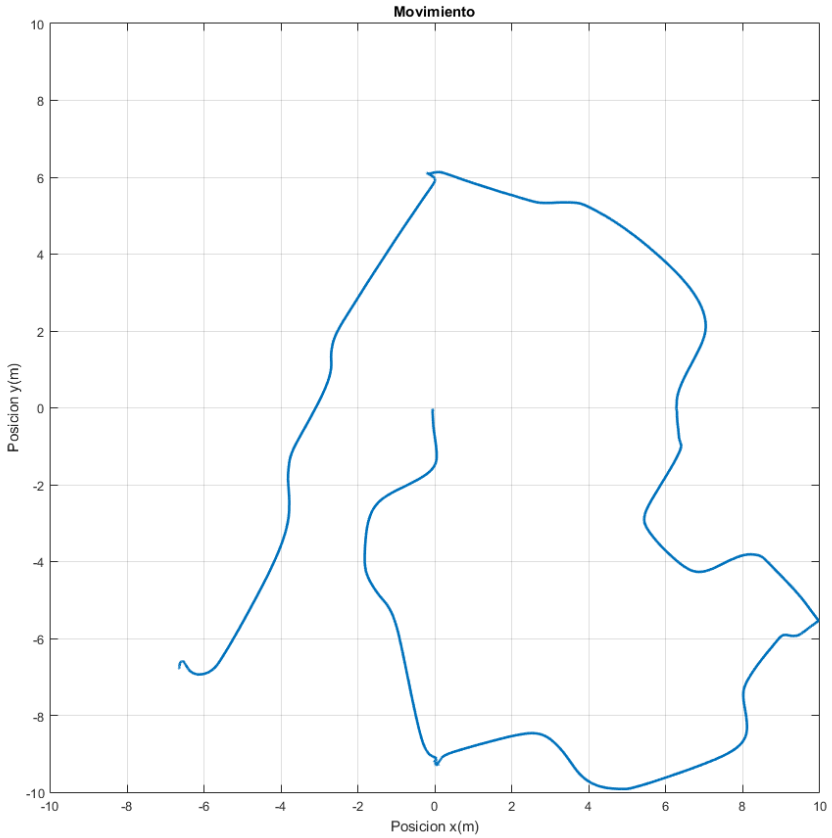

Figura 5-15 Posición con múltiples objetos, referencia dinámica 2

A pesar de tener mayor cantidad de obstáculos, la precisión es bastante buena, a simple vista se puede ver que es mejor que la del supuesto anterior.

6 CONCLUSIONES Y TRABAJO FUTURO

En este proyecto se ha mostrado como usar redes neuronales para la navegación con UAVs. El principal reto de este proyecto ha sido la creación de redes con una estructura compleja. La navegación es una tarea que involucra gran cantidad de variantes que hay que tener en cuenta. Por lo que crear un agente capaz de resolver dicha tarea es un reto.

En conclusión, es posible crear un agente basado en redes neuronales para que navegue en un entorno, pasando las referencias en velocidad. Respecto del entrenamiento, debe destacar que con pocas muestras se puede conseguir entrenar de forma bastante eficiente el UAV. Además, según la complejidad del entorno en el que se quiera interactuar se necesitarán más o menos muestras. Si no existiera ningún obstáculo bastaría con menos de 10 minutos de muestras para entrenarlo, lo que significa que el agente se entrenará rápidamente. Sin embargo, cuando la complejidad del entorno aumenta, la cantidad de distintos experimentos puede hacer replantear el uso de aprendizaje profundo para realizarlo.

El uso de las salidas, que permite obtener un rango continuo de velocidades, es un acierto ya que permite mitigar ciertos errores que cometa el agente a la hora de obtener las referencias. Esto evita que se pueda quedar parado en ciertas circunstancias y favorece a que se llegue al objetivo final con mayor precisión.

En general, los resultados del proyecto son satisfactorios. A continuación, se procederá a comentar ciertas mejoras que se podrían haber aplicado y algunas valoraciones para algún futuro trabajo.

6.1 Posibles mejoras

Este proyecto ha sido una primera aproximación a la navegación mediante el uso de redes neuronales profundas. La amplitud del campo permite añadir diversas mejoras al proyecto.

Al realizar el proyecto usando redes neuronales es necesario realizar distintos experimentos que posteriormente el agente intentará imitar. Por tanto, todo lo que quieras que realice el UAV será necesario que se haga anteriormente y después se pueda generalizar para distintos casos. Todo esto implica que cualquier comportamiento que se quiera que tenga el UAV es necesario introducirlo a la hora de entrenar al agente.

La aproximación realizada en este proyecto no tiene en cuenta lagunas facetas en la navegación como son:

- Saltar obstáculos, usar las velocidades en la coordenada z para sobrepasar obstáculos.
- La precisión es un tema importante según que aplicación. A pesar de tener una precisión aceptable en ocasiones tarda algo más de tiempo para llegar hasta el objetivo, debido a que se ha pasado de velocidad o está demasiado al lado. Se podría mejorar la rapidez con la que llega a estos puntos.
- Durante el proceso todos los experimentos se han hecho a unos 12 metros a la redonda. Esto hace que cuando se quiere mandar a distancias mayores no interprete correctamente los datos y queda en un estado en el que no se mueve.
- Trayectoria, según lo que va percibiendo en cada instante se toman las decisiones que llevan hasta la

posición objetivo. Al no proporcionar ningún mapa no es posible limitar zonas por las que el UAV no pueda pasar.

6.2 Reflexión final y trabajo futuro

Recientemente se han hecho grandes avances en el uso de la inteligencia artificial, en los últimos años se ha visto como DeepMind [27], ha conseguido crear agentes capaces de ganar a los humanos en juegos de una extraordinaria complejidad. El primero fue con AlphaGo en 2015 el cual fue capaz por primera vez a un humano en el juego 'Go'. Posteriormente se desarrolló AlphaZero en 2017, la cual fue capaz de vencer a los motores de ajedrez más potentes como es 'Stockfish'. El último ha sido a finales del año pasado 2018 con AlphaStar [28], el cual fue capaz de ganar a jugadores profesionales en el juego StarCraft 2.

Para ser capaces de dominar estos juegos, es necesario ser capaces de gestionar una estrategia a corto y largo plazo además de intentar predecir lo que hace el adversario. En caso del StarCraft 2 además no se tiene información completa sobre el enemigo, lo cual aumenta la dificultad.

Todas estas nuevas técnicas hacen plantearse la posibilidad de incluirlos en la navegación para robots móviles. En el proyecto, la solución del agente pasaba por el entrenamiento basado en la imitación de un operador. Esto limita bastante la forma en la que el agente se enfrenta a las situaciones y por lo general no es capaz de entender si la forma de proceder que está tomando es la correcta o no. Con el uso de aprendizaje reforzado es posible sobrepasar estos límites debido a que sería el propio UAV el que interactuaría con el entorno para sacar sus propias soluciones. Esto hace que el resultado sea más diverso y robusto con nuevos tipos de entornos.

A partir del paper [6], se puede obtener ideas para iniciar en movimiento con aprendizaje reforzado, sin embargo, como se dice todavía no encuentra el camino más corto posible. El cual puede ser un punto interesante para un futuro proyecto. El hecho de usar aprendizaje reforzado puede también requerir mayor capacidad de cómputo a la hora de entrenar, sin embargo, existen ejemplos en los que se mezcla el aprendizaje por imitación (como en el proyecto) y el aprendizaje reforzado. Haciendo uso de ambos, se puede disminuir el tiempo de entrenamiento. Con este planteamiento se conseguiría superar todas las mejoras planteadas en el apartado anterior.

REFERENCIAS

[1] G.N. Desouza, A.C. Kak, «https://ieeexplore.ieee.org/document/982903,» 2002. [En línea].

[2] «https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping,» [En línea].

[3] P. B. A. L. R. V. Alessandro Gasparetto, «Path Planning and Trajectory Planning Algorithms: A General Overview».

[4] Keisuke Tateno, Federico Tombari, Iro Laina, Nassir Navab, «https://arxiv.org/pdf/1704.03489.pdf».

[5] Tingxiang Fan, Xinjing Cheng, Jia Pan, Dinesh Manocha, Ruigang Yang, «CrowdMove: Autonomous Mapless Navigation in Crowded Scenarios,» [En línea].

[6] Vikas Dhiman, Shurjo Banerjee, Brent Griffin, Jeffrey M. Siskind, Jason J. Corso, «https://arxiv.org/pdf/1802.02274.pdf,» [En línea].

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, «https://arxiv.org/pdf/1312.5602.pdf,» [En línea].

[8] M. Pfeiffer, S. Shukla, M.Turchetta, C. Cadena, A. Krause, R. Siegwart, J. Nieto, «https://arxiv.org/pdf/1805.07095.pdf,» [En línea].

[9] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, Hon-Mark Allen, Vinh-Dieu Lam, Alex Bewley, Amar Shah, «https://arxiv.org/pdf/1807.00412.pdf,» [En línea].

[10] «https://www.theverge.com/2019/6/5/18654044/amazon-prime-air-delivery-drone-new-design-safety-transforming-flight-video,» [En línea].

[11] Shichao Yang, Sandeep Koman, Chen Ma, Stephanie Rosenthal, Manuela Veloso, Sebastian Scherer, «https://arxiv.org/pdf/1704.08759.pdf,» [En línea].

[12] Grosan, Crina; Abraham, Ajith, Intelligent Systems A Modern Approach, vol. 2, 2011.

[13] «https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/,» [En línea].

[14] «https://en.wikipedia.org/wiki/Artificial_neural_network,» [En línea].

[15] «https://en.wikipedia.org/wiki/Convolutional_neural_network,» [En línea].

[16] «https://brohrer.github.io/how_convolutional_neural_networks_work.html,» [En línea].

[17] Achuta Kadambi, Ayush Bhandari, Ramesh Raskar, «https://pdfs.semanticscholar.org/7bcc/aa4c7afb021d8606f523738e97842b90c1ed.pdf,» [En línea].

[18] «https://www.ros.org/,» [En línea].

[19] «http://gazebosim.org/,» [En línea].

[20] «https://www.tensorflow.org/,» [En línea].

[21] «https://www.numpy.org/,» [En línea].

[22] «https://opencv.org/,» [En línea].

[23] «https://www.pygame.org/wiki,» [En línea].

[24] «https://docs.python.org/2/library/sys.html,» [En línea].

[25] «https://docs.python.org/2/library/os.html,» [En línea].

[26] «https://github.com/,» [En línea].

[27] «https://deepmind.com/,» [En línea].

[28] «https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/,» [En línea].

APÉNDICE

Apéndice I: Esquema red completa