



ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE

### **TRABAJO FIN DE GRADO**

Navegación autónoma de un dron  
para localizar un transmisor de radio frecuencia  
basado en aprendizaje por refuerzo

Autor: Cristian Sánchez Rodríguez

Tutor: Dr. Roberto Calvo Palomino

Curso académico 2023/2024

# Agradecimientos

---

Me gustaría agradecer a mi familia que siempre me ha apoyado y me ha dado la oportunidad de estudiar lo que hoy en día me apasiona. También mencionar a todas esas personas que han hecho de este periodo, algo más llevadero.

Además, me gustaría dar las gracias al profesorado, que ha conseguido despertar en mí esas ganas que, a día de hoy, me hacen considerarme parte de la comunidad robótica, y en especial gracias a Roberto por su aguante y paciencia depositados en mí a lo largo de este proyecto.

Madrid, 13 de Octubre de 2023

*Cristian Sánchez Rodríguez*

# Resumen

---

En la actualidad, la ciencia ha avanzado a pasos agigantados con respecto a las soluciones tecnológicas. Especialmente la robótica y la Inteligencia Artificial (IA), gracias también a que abarca una inmensa variedad de campos donde se pueden desarrollar soluciones eficientes y robustas. Uno de los puntos más destacables es la generación de comportamientos autónomos, lo que dota a estos sistemas de una independencia muy favorable a la hora de resolver tareas complejas, vease por ejemplo en labores de patrullaje, para controlar ciertas zonas, o también en agricultura, donde se requiera controlar que zonas deben regarse más y cuales menos, entre otros.

Adicionalmente, nos encontramos el uso de drones, o sistemas aéreos provistos de sensores y actuadores, que amplían el abanico de uso en el entorno tecnológico, permitiendo abordar los problemas desde nuevas perspectivas. En este proyecto, el foco de estudio se centra en los *Unmanned Air Vehicles* (UAV), cuyo uso abarca desde rescates en áreas poco accesibles, hasta inspecciones en la industria, o labores de inventariado, entre muchos otros ejemplos.

En paralelo, tenemos el estudio del espectro de Radio Frecuencia (RF), el cual abarca gran parte del mundo moderno, como puede ser en el uso de comunicaciones móviles, o en la transmisión usando bandas de frecuencia como la radio FM. Por ello, comprender y trabajar sobre la propagación de señales puede introducirnos en nuevas aplicaciones útiles para el mundo.

De este modo, surge la idea de realizar este Trabajo de Fin de Grado (TFG), agrupando cada parte descrita anteriormente, es decir, soluciones autónomas con dispositivos aéreos tremadamente adaptables a las circunstancias de un problema derivado de señales RF. Concretamente, el objetivo de este proyecto es demostrar que, empleando aprendizaje por refuerzo (Q-Learning), se puede lograr rastrear y navegar hacia una el transmisor de una señal RF, utilizando un dron situado en un escenario realista (con obstáculos y degradación de señal), de forma efectiva y robusta.

# Acrónimos

---

**RAE** Real Academia Española

**TFG** Trabajo de Fin de Grado

**UAV** *Unmanned Air Vehicles*

**UAS** *Unmanned Aerial Systems*

**GCS** *Ground Control Station*

**SUAV** *Small Unmanned Air Vehicle*

**LOS** *Line Of Sight*

**IA** Inteligencia Artificial

**RF** Radio Frecuencia

**EM** Electromagnético

**ROS** *Robot Operating System*

**ADC** *Analog to Digital Converter*

**RSSI** *Received Signal Strength Indicator*

**SNR** *Signal to Noise Ratio*

**PLE** *Path-Loss Exponent*

**AMR** *Autonomous Mobile Robot*

**AGV** *Automated Guided Vehicle*

**SAR** *Search and Rescue*

**VFF** *Virtual Force Field*

**RL** *Reinforcement Learning*

**MPSS** Módulo de Propagación de Señal Simulado

**ANAV** Algoritmo de Navegación Autónoma basado en la Información de la Vecindad

**PPO** *Proximal Policy Optimization*

**DDPG** *Deep Deterministic Policy Gradient*

**SAC** *Soft Actor Critic*

# Índice general

---

<b>1. Introducción</b>	<b>1</b>
1.1. Robots . . . . .	2
1.1.1. Drones . . . . .	3
1.2. Inteligencia artificial . . . . .	5
1.2.1. Aprendizaje por refuerzo . . . . .	6
1.3. Vigilancia del espectro electromagnético . . . . .	8
<b>2. Objetivos</b>	<b>9</b>
2.1. Descripción del problema . . . . .	9
2.2. Requisitos . . . . .	10
2.3. Metodología . . . . .	10
2.4. Plan de trabajo . . . . .	11
<b>3. Plataformas de desarrollo y herramientas utilizadas</b>	<b>12</b>
3.1. Lenguajes de programación . . . . .	12
3.1.1. Python . . . . .	12
3.1.2. C++ . . . . .	13
3.2. <i>Robot Operating System</i> (ROS) . . . . .	13
3.2.1. Rviz . . . . .	14
3.3. Gazebo 11 . . . . .	14
3.4. Plataformas de programación . . . . .	15
3.4.1. Visual Studio Code . . . . .	15
3.4.2. Github . . . . .	16
3.5. Módulos . . . . .	17
3.5.1. OpenCV . . . . .	17
3.5.2. Matplotlib . . . . .	17
3.5.3. PX4 autopilot . . . . .	18
3.6. Iris . . . . .	19

<b>4. Diseño</b>	<b>20</b>
4.1. Preparación del entorno . . . . .	20
4.1.1. JdeRobot - drones . . . . .	21
4.1.2. Teleoperador . . . . .	21
4.2. Modelo de propagación de señal . . . . .	26
4.2.1. Aproximación de Friis . . . . .	26
4.2.2. Módulo python de Friis . . . . .	28
4.2.3. Aplicación de Friis . . . . .	29
4.2.4. Módulo de Propagación de Señal Simulado (MPSS) - Integración de Friis con ROS . . . . .	31
4.3. Navegación autónoma para localizar el origen de una señal RF . . . . .	32
4.3.1. Algoritmo de Navegación Autónoma basado en la Información de la Vecindad (ANAV) . . . . .	33
4.3.2. ANAV mejorado . . . . .	35
4.3.3. Algoritmo de navegación basado en <i>Reinforcement Learning</i> (RL) .	36
4.3.4. Experimentos y resultados . . . . .	40
4.4. Comportamiento sigue señal basado en RF en un entorno dinámico . .	46
4.4.1. Introducción al problema . . . . .	46
4.4.2. Algoritmos . . . . .	46
4.4.3. Experimentos y resultados . . . . .	47
<b>5. Conclusiones</b>	<b>51</b>
5.1. Objetivos cumplidos . . . . .	51
5.2. Balance global y competencias adquiridas . . . . .	51
5.3. Líneas futuras . . . . .	52
<b>Bibliografía</b>	<b>54</b>

# Índice de figuras

---

1.1.	Robótica industrial VS robótica móvil . . . . .	2
1.2.	Definición de robot . . . . .	3
1.3.	Historia de los drones . . . . .	4
1.4.	Descripción gráfica de <i>Unmanned Aerial Systems</i> (UAS) (GCS + Data Links + UAV) . . . . .	4
1.5.	Clasificación de aprendizaje máquina . . . . .	6
1.6.	Aprendizaje por refuerzo . . . . .	7
2.1.	Insights Github por meses TFG . . . . .	11
3.1.	Esquema de comunicaciones en ROS . . . . .	14
3.2.	Ejemplo de uso de Rviz . . . . .	15
3.3.	Simulación dron en Gazebo . . . . .	16
3.4.	Interfaz gráfica usando OpenCV . . . . .	17
3.5.	Representación de un mapa de calor usando matplotlib . . . . .	18
3.6.	Panorámica de PX4 con ROS . . . . .	18
3.7.	Iris drone en Gazebo 11 . . . . .	19
4.1.	Diagrama esquemático del sistema completo . . . . .	21
4.2.	3DR Iris simulado . . . . .	22
4.3.	Esquema comunicaciones del controlador . . . . .	23
4.4.	Teleoperador . . . . .	24
4.5.	Uso del módulo Friis con señal WIFI (5 GHz) VS señal GSM (0.9 GHz)	29
4.6.	Versión final de la interfaz . . . . .	30
4.7.	Esquema comunicaciones MPSS . . . . .	31
4.8.	Esquema comunicaciones del bloque de comportamiento autónomo . . . . .	32
4.9.	Algoritmo de navegación autónoma basado en la información de la vecindad . . . . .	33
4.10.	Algoritmo de navegación autónoma basado en la información de la vecindad (mejorado) . . . . .	35

4.11. Representación de estados y acciones . . . . .	36
4.12. Diagrama condición de finalización . . . . .	37
4.13. Gráfico de entrenamiento . . . . .	40
4.14. Transmisor centrado (12x12) . . . . .	41
4.15. Comparativas (12x12), transmisor centrado . . . . .	41
4.16. Transmisor en la esquina (12x12) . . . . .	43
4.17. Comparativas (12x12), transmisor en la esquina . . . . .	43
4.18. Transmisor en la esquina (30x30) . . . . .	44
4.19. Comparativas (30x30), escenario planteado . . . . .	45
4.20. Trayectorias seguidas durante la inferencia . . . . .	45
4.21. Escenario con obstáculos (30x30) . . . . .	46
4.22. Simulación de sensor para obstáculo . . . . .	48
4.23. Diagrama de funcionamiento híbrido . . . . .	48
4.24. Funcionamiento general del algoritmo <i>Virtual Force Field</i> (VFF) . . . . .	49
4.25. Enfoque híbrido . . . . .	50

# Listado de códigos

---

3.1. Obtención del parámetro lambda en función de una frecuencia (en este caso 5G) . . . . .	12
3.2. Hello world en C++ . . . . .	13
4.1. Bloque de código principal de la versión final del teleoperador . . . . .	25
4.2. Ejemplo básico de uso del módulo Friis . . . . .	28
4.3. Código simplificado de la función de recompensa y actualización tabla Q	39

# Índice de cuadros

---

4.1. Valores n de referencia . . . . .	27
4.2. Parámetros usados durante todos los entrenamientos . . . . .	38
4.3. Características de la señal por defecto . . . . .	42

---

# Capítulo 1

## Introducción

---

En la actualidad, la tecnología forma parte de nuestro día a día. Prácticamente, constituye un elemento imprescindible para llevar a cabo cualquier actividad, sea profesional o cotidiana. Su función consiste en solucionar problemas para hacernos la vida más sencilla.

Con esto en mente, se presenta la robótica que, según la Real Academia Española (RAE), se define como “*técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales.*” (Real Academia Española, s.f., definición 2)<sup>1</sup>. Sin embargo, no es precisa, por ello una definición más concreta podría ser, ciencia que engloba diversas ramas tecnológicas, encargada del estudio y diseño de dispositivos mecánicos, provistos de sensores y actuadores, capaces de realizar tareas a través de la extracción y posterior procesamiento de la información, con el fin de generar respuestas adecuadas para resolver determinados problemas<sup>2</sup>.

Dentro de la robótica, existen diversas maneras de clasificar, sin embargo, una de las más comunes esta relacionada con la movilidad del dispositivo, esto es, si el mecanismo se puede desplazar por su entorno o no (figura 1.1), por tanto se distingue lo siguiente:

1. Robótica industrial: que involucra mecanismos fijos, capaces de realizar tareas de manera rápida, precisa y eficiente. Como es el caso de los brazos robóticos<sup>3</sup>.
2. Robótica móvil: la cual abarca a los dispositivos móviles que se engloban en múltiples entornos y aplicaciones, como pueden ser, robótica aérea, terrestre y submarina<sup>4</sup>.

---

<sup>1</sup><https://dle.rae.es/robótico#WYTncqf>

<sup>2</sup><https://revistaderobots.com/robots-y-robotica/que-es-la-robotica/?cn-reloaded=1>

<sup>3</sup><https://www.geeksforgeeks.org/industrial-robots/>

<sup>4</sup><https://www.geeksforgeeks.org/mobile-robots/>

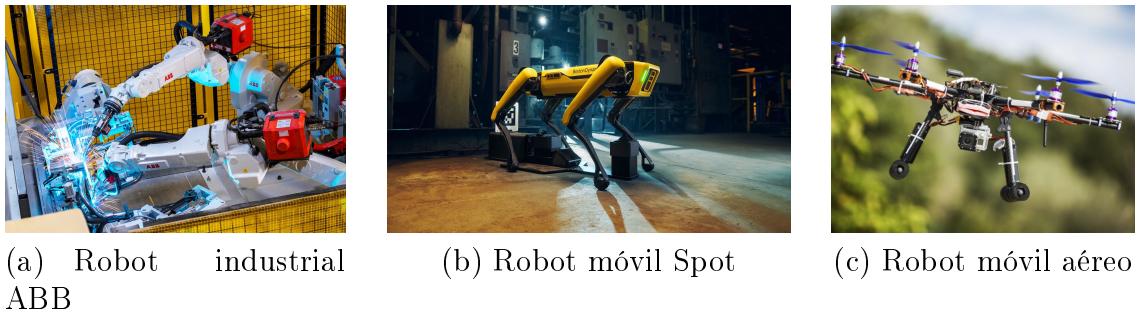


Figura 1.1: Robótica industrial VS robótica móvil

Como tal, la robótica ayuda a resolver tareas repetitivas, peligrosas, delicadas y en ambientes problemáticos (conocidas como las 4 D's, *dull, dirty, dangerous and dear*)<sup>5</sup>. Sin embargo, uno de los problemas más complicados de abordar, es el contexto, es decir, la capacidad de entender y adaptarse a las circunstancias del problema, como por ejemplo en el caso de la conducción autónoma, donde detectar un simple peatón, puede derivar en infinitos inconvenientes (condiciones de visibilidad, clima, atuendo, entre muchos otros). Es ahí, donde se presenta el segundo punto importante, la IA [1].

## 1.1. Robots

Un robot es un dispositivo provisto con sensores, o elementos capaces de extraer información del entorno (por ejemplo una cámara), actuadores, o elementos que permiten al dispositivo realizar acciones (por ejemplo un motor), y una unidad de procesamiento, que se encarga de generar acciones a través de la información obtenida con los sensores, todo ello mediante algoritmos [2], tal y como se puede ver en la figura 1.2.

Existen múltiples robots capaces de satisfacer estas condiciones, vease, los *Automated Guided Vehicle* (AGV)/*Autonomous Mobile Robot* (AMR) o plataformas robóticas terrestres ampliamente empleadas logística, que permiten mover mercancía y navegar de forma autónoma por almacenes y naves industriales<sup>6</sup>; los robots bipedos, los cuales emulan el movimiento humanoide, lo que aumenta su adaptabilidad a cualquier entorno real (ya que el mundo esta diseñado para la biomecánica humana), sin embargo, son bastante complejos debido a la dificultad de replicar la marcha bípeda [3]; y por

<sup>5</sup><https://bernardmarr.com/the-4-ds-of-robotisation-dull-dirty-dangerous-and-dear/>

<sup>6</sup><https://www.mobile-industrial-robots.com/insights/get-started-with-amrs/agv-vs-amr-whats-the-difference/>

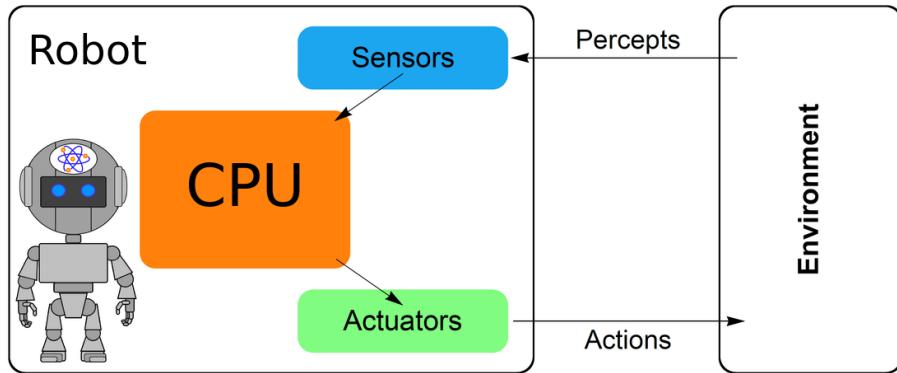


Figura 1.2: Definición de robot

último, los drones, empleados en labores de *Search and Rescue* (SAR), o de inspección en lugares poco accesibles, entre otros.

### 1.1.1. Drones

Los drones tienen origen en la primera guerra mundial, con el biplano llamado Kettering bug. Se trataba de un torpedo que era lanzado desde una carretilla, capaz de volar de forma no tripulada, hasta que se liberaba de sus alas y caía sobre el objetivo<sup>7</sup>. Más tarde, entre la primera y segunda guerra mundial (1935), se diseño el Queen Bee, de donde surgió el termino “drone”, como abeja macho en busca de la reina, que se trataba de un avión no tripulado, con el fin de servir de objetivo para realizar prácticas de artillería aérea<sup>8</sup>. Sin embargo, no fue hasta Operation Aphrodite, en la segunda guerra mundial, donde realmente se vió el primer dron radio tripulado, con el fin de poder volar en entornos “sucios” o dirty, dado el nuevo paradigma de las bombas atómicas<sup>9</sup>. En la figura 1.3, se pueden ver los ejemplos mencionados.

Existen múltiples avances y ejemplos posteriores, pero en la actualidad podemos definir un UAS (ver figura 1.4) teniendo en cuenta lo siguiente<sup>10</sup>:

1. *Ground Control Station* (GCS): es la estación de tierra o el elemento encargado de controlar la nave<sup>11</sup>
2. Comunicación: conecta y gestiona la transmisión de datos entre el UAV y la GCS, mediante data links, o canales de transmisión [4].

<sup>7</sup><https://www.nationalmuseum.af.mil/Visit/Museum-Exhibits/Fact-Sheets/Display/Article/198095/kettering-aerial-torpedo-bug/>

<sup>8</sup><https://www.dehavillandmuseum.co.uk/aircraft/de-havilland-dh82b-queen-bee/>

<sup>9</sup><https://warfarehistorynetwork.com/article/operation-aphrodite/>

<sup>10</sup><https://srmconsulting.es/blog/uav-uas-rpa-dron-como-llamarlos.html>

<sup>11</sup><https://www.trentonsystems.com/blog/ground-control-stations>

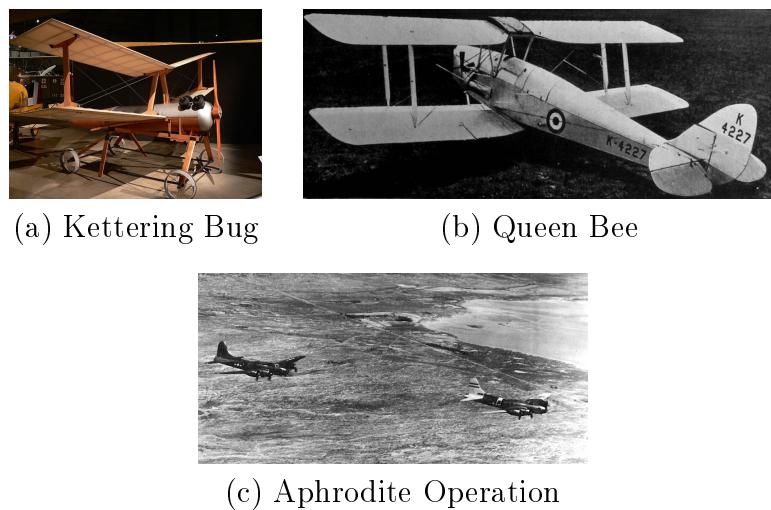


Figura 1.3: Historia de los drones

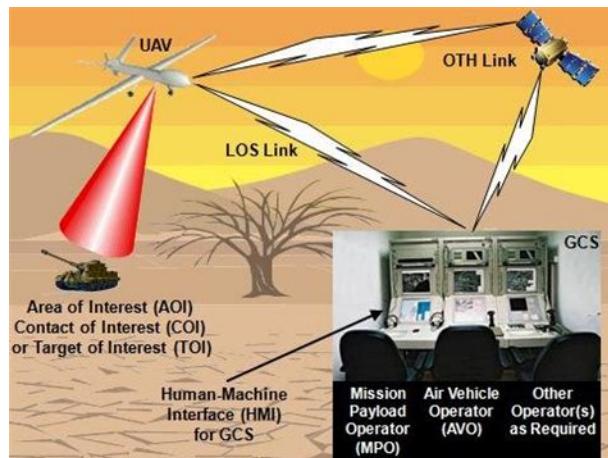


Figura 1.4: Descripción gráfica de UAS (GCS + Data Links + UAV)

3. UAV: hace referencia directamente a la aeronave.

También cabe destacar que hay variedad de drones, según su peso y capacidad de carga de pago, o elementos que sea capaz de cargar, lo cual influye en la legislación detrás de su uso (de forma general, cuanto mayor sea el peso, más legislación debe cumplir y mayores restricciones de uso tiene)<sup>12</sup>. Tal y como fue mencionado, la gran ventaja del uso de vehículos aéreos es poder evitar las irregularidades del terreno, sin embargo, hay ligados al uso de estos dispositivos ciertos problemas, como son el clima, la carga de pago que afecta a la autonomía (peso de las baterías), los interiores (afectan a la señal GPS), entre otros.

Agrupando la robótica y los drones, se pueden observar múltiples ejemplos de

<sup>12</sup><https://www.safedroneflying.aero/en/drone-guide/drone-regulations>

uso, uno muy conocido es el de un dron “*sigue-persona*”, el cual permite a un *Small Unmanned Air Vehicle* (SUAU) detectar y moverse al son de un objetivo móvil, tal y como puede ser una persona [5]; o bien para controlar desastres naturales, como por ejemplo un incendio, donde mediante visión artificial se puedan localizar y controlar los focos activos<sup>13</sup>.

Estos comportamientos, son especialmente complejos debido a que se navega por entornos desconocidos (es decir, sin un mapa disponible), además de estar sujetos a una reactividad elevada, lo que requiere un procesamiento de datos eficiente y una baja latencia en las comunicaciones, lo cual está directamente relacionado con las condiciones del entorno y su contexto.

## 1.2. Inteligencia artificial

La IA ha tenido un auge importante en los últimos años, especialmente en el ámbito de los drones dado su amplio abanico de soluciones sinérgicas con la robótica, desde “*Computing Machinery and Intelligence*” (Alan Turing, 1950), donde se buscó responder fue la siguiente ¿Puede una máquina pensar?, formulada en “*Computing Machinery and Intelligence*” (Alan Turing, 1950); pasando por Logic Theorist en el Dartmouth Summer Research Project on Artificial Intelligence [?]; hasta la actualidad, donde los algoritmos mejoraron a la par de la capacidad de computación, destacando por ejemplo la navegación autónoma, empleada en drones entre otros vehículos<sup>14</sup>.

En general, la IA se puede clasificar en función de los siguientes enfoques (ver figura 1.5):

1. Aprendizaje supervisado: es decir, se emplea un conjunto de datos del que se conocen tanto las salidas como las entradas a las que pertenecen. La idea es conseguir obtener una salida exacta dada una entrada concreta. Por ejemplo, para detectar obstrucciones o cualquier obstáculo presente en la ruta generada por el plan de navegación de un dron entrenado a través de datos etiquetados por imágenes [6].
2. Aprendizaje no supervisado: donde se tiene un conjunto de datos de entrada sin etiquetar. Básicamente, se encarga de distribuir dicho conjunto en sets con

---

<sup>13</sup><https://www.euronews.com/2023/09/19/could-ai-powered-drones-be-the-solution-to-europees-wildfire-problems>

<sup>14</sup><https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/>

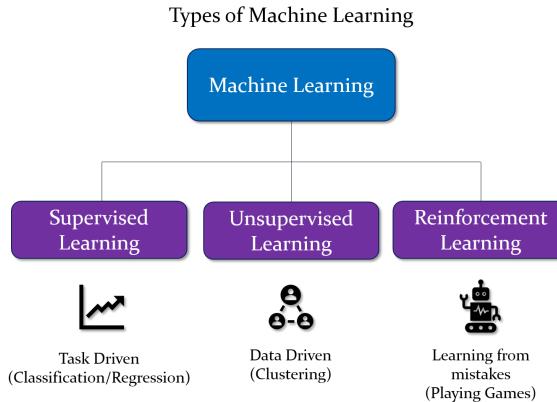


Figura 1.5: Clasificación de aprendizaje máquina

características comunes. Un ejemplo común es la segmentación de imágenes, donde se clasifica cada elemento de la imagen según su naturaleza, como puede ser el caso de detectar turbinas defectuosas o no defectuosas, en aeronaves empleando reconocimiento por imagen [7].

3. Aprendizaje por refuerzo: resuelve un problema a base de prueba y error, mediante un sistema de recompensas. Como por ejemplo “*Stockfish*”, que es un modelo entrenado para ganar una partida de ajedrez en el menor número de movimientos posible, superando incluso a grandes maestros de la actualidad<sup>15</sup>, o por ejemplo, en un caso más relacionado, para mejorar la navegación en drones [8].

16

### 1.2.1. Aprendizaje por refuerzo

Se basa en un sistema de recompensas y penalizaciones, que permite entrenar a un modelo para converger hacia la toma de buenas decisiones. Este enfoque se basa en los llamados procesos de Markov, que se definen como aquellos que, para un instante dado, contienen toda la información relevante sin depender de todos los procesos anteriores.<sup>17</sup>

En particular, hablamos de agente, o modelo encargado de tomar decisiones en un entorno (que es el medio en el que interactúa dicho agente, y está regido por una serie de reglas); estados, o circunstancias en la que se sitúa el agente en un determinado instante

<sup>15</sup><https://stockfishchess.org/about/>

<sup>16</sup><https://www.springboard.com/blog/data-science/regression-vs-classification/>

<sup>17</sup><https://www.geeksforgeeks.org/what-is-reinforcement-learning/>

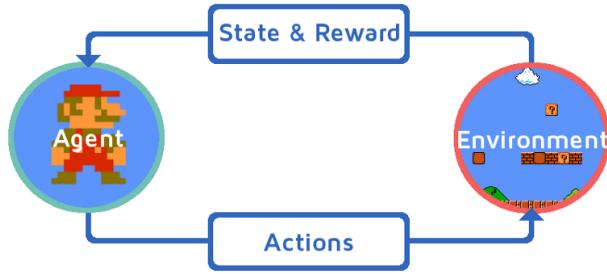


Figura 1.6: Aprendizaje por refuerzo

temporal; y acciones, o decisiones que toma el agente y que le permiten cambiar de estado, como se puede ver en la figura 1.6. En términos de Markov, decimos que el estado actual no depende de todos los estados previos.<sup>18</sup>

Cabe destacar que, este enfoque está directamente extraido de la psicología y el estudio del comportamiento, donde en función de recompensas y castigos se induce al aprendizaje en distintas tareas, como por ejemplo, enseñar a jugar al ping pong a dos palomas<sup>19</sup>, o más en relación a aplicaciones con aeronaves, para obtener rutas óptimas en carreras de drones [9].

Entre los distintos modelos, encontramos Q-Learning, que busca generar una tabla numérica donde cada fila se interprete como un estado del robot, que puede ser su posición; y cada columna sea una determinada acción, como puede ser moverse hacia algún lugar. De este modo, y a través de una función de recompensa, se llenan los valores de la tabla, los cuales, según el tipo de función escogida, convergerá comportamientos de un tipo u otro. Una vez obtenida la tabla, el robot solo debe identificar en qué estado se encuentra (fila) y elegir la columna con mayor valor numérico, lo que se traducirá en la mejor acción para dicho estado<sup>20</sup>.

Existen múltiples ejemplos de aplicación de esta metodología a casos reales, vease para controlar de forma adaptativa una señal de tráfico; para jugar a la Atari 2600; o para realizar un control híbrido sobre la navegación de un Robot [10]; o para seleccionar qué vehículos reduce costes y mejorar la eficiencia, de cara a entregar mercancía (vía aérea empleando drones o vía terrestre) [11].

<sup>18</sup><https://www.alexanderthamm.com/es/blog/refuerzo-aprendizaje-marco-y-ejemplo-de-aplicacion/>

<sup>19</sup><https://pressbooks.online.ucf.edu/lumenpsychology/chapter/operant-conditioning/>  
<https://pressbooks-dev.oer.hawaii.edu/psychology/chapter/operant-conditioning/>

<sup>20</sup><https://towardsdatascience.com/reinforcement-learning-explained-visually-part-4-q-learning->

### 1.3. Vigilancia del espectro electromagnético

Las comunicaciones inalámbricas son aquellas donde tanto el emisor como el receptor se intercambian información mediante ondas electromagnéticas. En su defecto usan ondas electromagnéticas moduladas transmitidas generalmente por el aire. En este caso concreto, hablamos de señales RF, como son por ejemplo Wi-Fi, radio FM, 4G, 5G, entre otros tipos de señales distribuidas a lo largo del espectro Electromagnético (EM).

Dicho espectro se divide por bandas de frecuencia, que se reparten para diversos uso. El ejemplo más claro es la banda FM de radio, que se reparte entre los 87-108 MHz para España, donde cada emisora tiene un ancho asignado para emitir, o por ejemplo la banda GPS, dispuesta para el posicionamiento, situada en 1575 MHz, o también la banda GSM, que distribuye la telefonía móvil y se encuentra en las bandas de 900 y 1800 MHz.<sup>21</sup>

De este modo, se pueden encontrar soluciones a problemas como el rastreo de una señal de móvil para una persona perdida en la montaña, o seguir emisores concretos, como pueden ser convoyes, o también en casos de ataques del tipo jamming (introducción de interferencias para invalidar la comunicación), donde se necesite hallar el origen del ataque, entre otros. Lo único que hay que establecer, es la banda de frecuencia adecuada y establecer un comportamiento que permita navegar hasta la señal de manera autónoma. Otro ejemplo de uso, es para mejorar la localización en robots, mediante el uso de redes 5G [12]. De manera más específica, podemos pensar en una aplicación donde se requiera navegar hacia una señal en un entorno interior de baja visibilidad, sea por ejemplo para guiar a las personas hacia una salida de emergencia, situando un transmisor en la misma, en una situación de incendio.

En definitiva, este proyecto se centra en desarrollar un comportamiento autónomo de un dron, basado en aprendizaje por refuerzo, con el fin de detectar el origen de una señal RF en un entorno dinámico, esto es, un escenario con obstáculos sobre el cual se navegue hasta la fuente de la señal.

---

<sup>21</sup>[https://www.wikiwand.com/en/FM\\_broadcast\\_band](https://www.wikiwand.com/en/FM_broadcast_band)

---

## **Capítulo 2**

# **Objetivos**

---

En este capítulo se describen los objetivos de este proyecto, así como los requerimientos, el método seguido y la estructura del mismo.

### **2.1. Descripción del problema**

Los drones son una herramienta tremadamente versátil, ya que permiten solventar los inconvenientes orográficos de forma sencilla, y pueden ser provistos de múltiples sensores, lo que incrementa su adaptabilidad para solucionar un gran abanico de retos ingenieriles.

Por tanto, el objetivo principal de este TFG es desarrollar un comportamiento autónomo basado en RL, sobre un dron para detectar y localizar el origen de una señal RF. Esto puede ser especialmente útil en labores de rastreo e identificación de objetivos, tales como en casos de escenarios catastróficos donde se deben localizar personas perdidas o de rastreo de señales en entornos de interior.

Para ello, se establecen los siguientes subobjetivos:

1. Desarrollo de una aplicación, enfocada a teleoperar un UAV, empleando herramientas de simulación y visualización.
2. Implementación de un modelo de propagación de señales, que nos ayude en el análisis para entornos simulados.
3. Desarrollo de un comportamiento autónomo capaz de detectar y navegar hacia la fuente de una señal RF.
4. Comparativa frente a los algoritmos navegación tradicionales.

5. Desarrollo de un comportamiento autónomo para que el dron navegue y detecte el transmisor, en un escenario dinámico.

## 2.2. Requisitos

Las especificaciones que se deben cumplir son las siguientes:

1. El modelo de propagación de señal debe basarse en el modelo de Friis.
2. Se debe usar ROS como middleware robótico y Gazebo como herramienta de simulación.
3. Se debe seleccionar el algoritmo que navegue de forma más eficiente en cuanto a tiempo, movimientos y acciones hacia puntos con mayor potencia de señal.
4. Se debe seleccionar el algoritmo más seguro en cuanto a salvaguardar la integridad del dispositivo y sus alrededores.

## 2.3. Metodología

Este trabajo, comenzó oficialmente en Septiembre de 2022, aunque se pusieran en común las ideas a principios del verano, y se concluye a finales de Septiembre de 2023.

La metodología para llevarlo a cabo fue la siguiente:

1. Reunión de control semanal o cada dos semanas vía Teams con el tutor, donde se realizaba una valoración del estado del proyecto y se establecían los futuros puntos a seguir.
2. Uso de la metodología Kanban, que es una metodología visual para gestionar y optimizar el flujo de trabajo a través de tarjetas y límites de trabajo en curso.
3. Empleo de la plataforma Github, a fin de establecer un repositorio común<sup>1</sup>, como sistema de control de versiones y de almacenamiento de backups.
4. Desarrollo de un blog donde se describe el estado del proyecto<sup>2</sup>.

---

<sup>1</sup><https://github.com/RoboticsLabURJC/2022-tfg-cristian-sanchez>

<sup>2</sup><https://roboticslaburjc.github.io/2022-tfg-cristian-sanchez/>

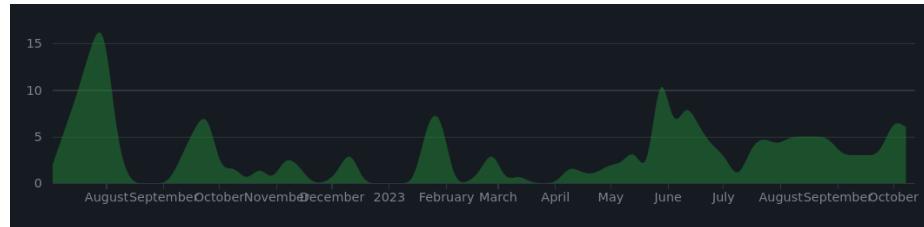


Figura 2.1: Insights Github por meses TFG

## 2.4. Plan de trabajo

Para concluir este capítulo, los pasos seguidos han sido:

1. Etapa inicial, donde tras establecer los objetivos del proyecto, se empezó por investigar el estado del arte del uso de drones para aplicaciones robóticas.
2. Los primeros pasos en el TFG se centraron realizar una aplicación para teleoperar a un dron.
3. El siguiente punto se basó en el estudio y comprensión de las señales RF dentro del espectro de radiofrecuencia, con el fin de desarrollar un modelo de propagación.
4. Posteriormente, se desarrollaron diversas soluciones encargadas de resolver el problema de detectar y navegar hacia una señal.
5. A continuación, se realizó una fase de pruebas y extracción de información sobre la que se realizaron diversas comparativas.
6. Cerrando la fase de desarrollo, lo último fue implementar soluciones sobre escenarios más realistas que incluían obstáculos.
7. Finalmente, se realizó la redacción de la memoria.

---

# Capítulo 3

## Plataformas de desarrollo y herramientas utilizadas

---

En este apartado se hablará de los recursos ingenieriles empleados para hacer posible el proyecto.

### 3.1. Lenguajes de programación

#### 3.1.1. Python

---

```
#! /usr/bin/env python

if __name__ == "__main__":
    C = 3.0 * (10 ** 8)
    freq = 5 * (10 ** 9)
    lmbda = C / freq
```

---

Código 3.1: Obtención del parámetro  $\lambda$  en función de una frecuencia (en este caso 5G)

A día de hoy, es considerado el lenguaje de programación más popular<sup>1</sup>. Se ideó en 1991 por Guido van Rossum y se desarrolló en la Python Software Foundation<sup>2</sup>. Es interpretado, es decir, usa un programa que traduce las líneas de código para la máquina en tiempo de ejecución (lo cual lo hace más intuitivo pero menos eficiente). Además, permite la programación orientada a objetos en alto nivel, lo que ofrece gran dinamismo a la hora de usarlo<sup>3</sup>.

Debido a su amplia popularidad, podemos acceder a una gran variedad de módulos y utilidades desarrollados por la comunidad, los cuales se integran perfectamente en la

<sup>1</sup><https://www.tiobe.com/tiobe-index/>

<sup>2</sup><https://www.geeksforgeeks.org/history-of-python/>

<sup>3</sup><https://www.python.org/doc/essays/blurb/>

<https://www.educative.io/blog/compiled-vs-interpreted-language>

resolución de nuestro problema.

En nuestro caso, python se usó para el crear la mayor parte del código empleado, es decir, para desarrollar interfaces gráficas, para trabajar con el middleware robótico ROS (detallado posteriormente) y para el desarrollo de los diversos algoritmos. Todo ello haciendo uso del módulo **numpy**, el cual nos permite realizar operaciones matemáticas y trabajar con vectores de forma rápida y eficiente; así como del módulo **matplotlib**, del cual hablaremos más adelante.

### 3.1.2. C++

---

```
#include <iostream>

int main(int argc, char ** argv) {
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

---

Código 3.2: *Hello world* en C++

También bastante popular, se encuentra el lenguaje de programación creado por Bjarne Stroustrup, en los laboratorios Bell en 1971. En este caso es compilado, lo que implica la traducción y enlazado previo a la ejecución. De corte más eficiente que Python, también permite la programación orientada a objetos. Se sitúa a medio camino entre un lenguaje de alto nivel y uno de bajo nivel<sup>4</sup>.

El uso designado en este proyecto para este lenguaje fue para poder trabajar con mapas de calor, mediante la biblioteca de *Anybotics* de grid maps<sup>5</sup>.

## 3.2. ROS

Si se habla de robótica, se habla de ROS, ya que es el framework para el desarrollo de soluciones de este ámbito, pero, ¿qué es exactamente ROS?.

Se trata de un *middleware*, es decir, una infraestructura software situada entre el sistema operativo y el desarrollador, que incluye una serie de módulos y funcionalidades

<sup>4</sup><https://www.geeksforgeeks.org/history-of-c/>

<sup>5</sup>[https://github.com/ANYbotics/grid\\_map](https://github.com/ANYbotics/grid_map)

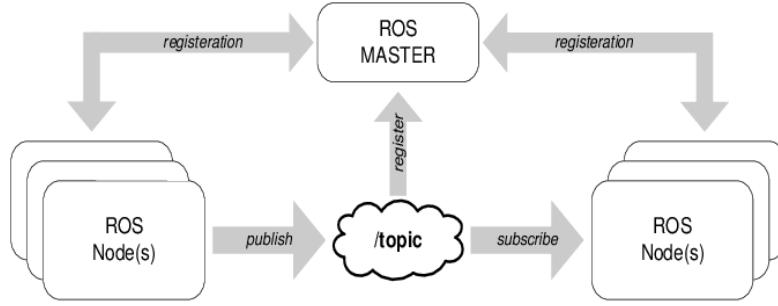


Figura 3.1: Esquema comunicaciones en ROS

enfocadas al desarrollo de aplicaciones robóticas<sup>6</sup>. La idea detrás, busca estandarizar soluciones que no dependan de los drivers de cada sensor y actuador presentes. De forma general, se trata de una arquitectura basada en el paradigma de publicador-suscriptor, donde una serie de nodos se comunican entre sí, transmitiendo mensajes propios, a través de canales compartidos llamados *topics*, esto es, un nodo suscriptor se suscribirá a un determinado topic, que permita la transmisión de un tipo de mensaje concreto, quedando en espera de que un nodo pulicador, envíe mensajes de este tipo a ese topic, tal y como se puede apreciar en la figura 3.1.

Concretamente, se usa para desarrollar todos los algoritmos robóticos y para realizar las comunicaciones necesarias en la simulación del dron.

### 3.2.1. Rviz

Es un visualizador 3D diseñado para la depuración de aplicaciones ROS<sup>7</sup>, tal y como se puede ver en la figura 3.2.

En nuestro caso, nos permite ver como se dispersa la señal RF, que trayectoria y orientación sigue el dron, que efecto tiene sobre la señal la presencia de obstáculos, y otras tantas opciones.

## 3.3. Gazebo 11

Se trata del simulador sobre el cual se desarrolla el proyecto. Concretamente consta de un conjunto de módulos optimizados para desarrollar aplicaciones robóticas,

<sup>6</sup><https://www.ibm.com/topics/middleware> <https://www.ros.org/>

<sup>7</sup><https://github.com/ros-visualization/rviz>

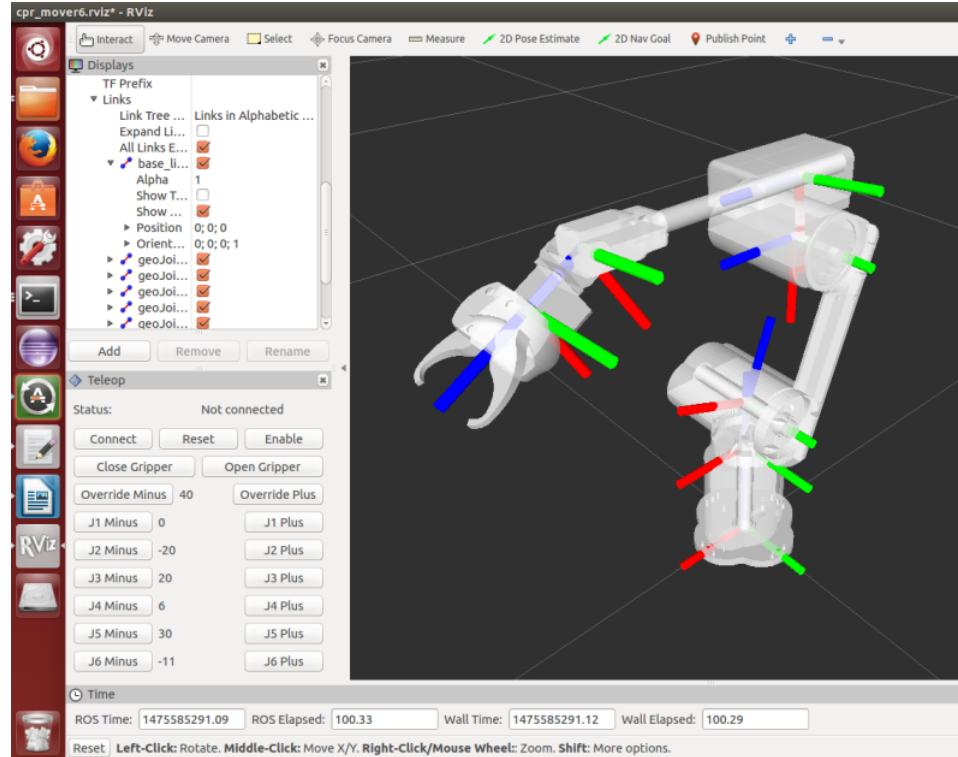


Figura 3.2: Ejemplo de uso de Rviz

ampliamente compatible con ROS (ver figura 3.3). Además, posee un motor de físicas basado en ODE, lo que permite simular con precisión el funcionamiento de un dron<sup>8</sup>.

Esta herramienta, nos permite visualizar en directo, el comportamiento del dispositivo frente a los diversos escenarios que se planteen.

## 3.4. Plataformas de programación

### 3.4.1. Visual Studio Code

Entre las plataformas usadas para programar, *Visual Studio Code*, o mejor conocido como *VS code*, es un editor de código ligero, funcional tanto en Linux, Windows y macOS<sup>9</sup>.

Su principal ventaja, es que es altamente personalizable a el tipo de desarrollo software que se deseé realizar. Todo ello a través de las múltiples extensiones que ofrece, así como la conexión directa y fluida con plataformas como Github, que se detallarán a continuación.

<sup>8</sup><https://gazebosim.org/about>

<sup>9</sup><https://code.visualstudio.com/docs>



Figura 3.3: Simulación dron en Gazebo

### 3.4.2. Github

Github nace de la herramienta git, creada por Linus Torvalds (desarrollador de Linux), que es un sistema de control de versiones, que funciona a grandes rasgos a través de repositorios (o lugares donde se almacenan los sistemas de versiones de forma local), y commits (que permiten actualizar la versión del código almacenado del repositorio)<sup>10</sup>.

De este modo, Github consiste en trasladar la idea de tener repositorios locales, para distribuirlos en una plataforma online, donde además se permite el desarrollo de aplicaciones de manera colaborativa.

Por ello, el papel que toma en este proyecto es de vital importancia, ya que asegura un seguimiento y una seguridad, de cara a tener copias de seguridad, donde todo el que desee puede acceder a ver en qué punto se encuentra el TFG pueda hacerlo.

---

<sup>10</sup><https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/>

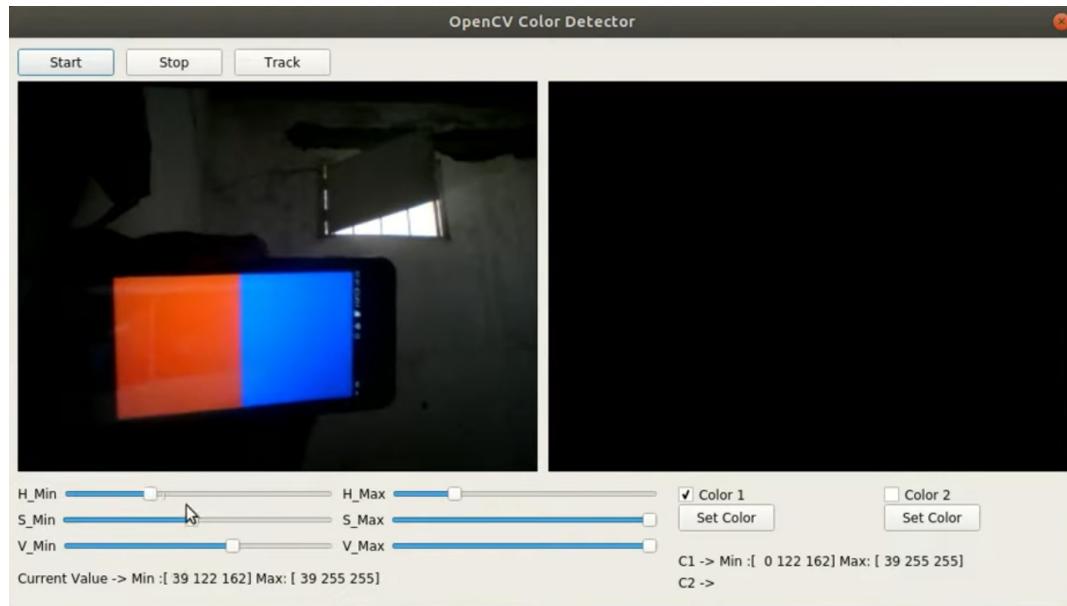


Figura 3.4: Interfaz gráfica usando OpenCV

## 3.5. Módulos

### 3.5.1. OpenCV

Es una biblioteca software de python, enfocada a visión artificial, y que además dispone de otras funcionalidades útiles para el desarrollo <sup>11</sup>.

Por tanto, el uso designado en este proyecto es el de implementar interfaces gráficas sencillas (vease el ejemplo de la figura 3.4), sobre las que interactuar dinámicamente con el dron, a través de barras de acción y botones.

### 3.5.2. Matplotlib

Presentada por John Hunter en 2002, se usó como alternativa para el desarrollo de interfaces gráficas. La gran diferencia radica en que está diseñada para trabajar con estructuras numéricas del tipo array (muy compatible con Numpy <sup>12</sup>), lo que permite ofrecer una gran visualización y una interfaz responsive <sup>13</sup>.

Concretamente en este proyecto se usa para desarrollar implementar mapas de calor (que en definitiva son estructuras numéricas del tipo array), dentro de una interfaz gráfica (de forma similar a la representación de la figura 3.5), para simular el

---

<sup>11</sup><https://opencv.org/about/>

<sup>12</sup><https://numpy.org/about/>

<sup>13</sup><https://www.geeksforgeeks.org/python-introduction-matplotlib/>

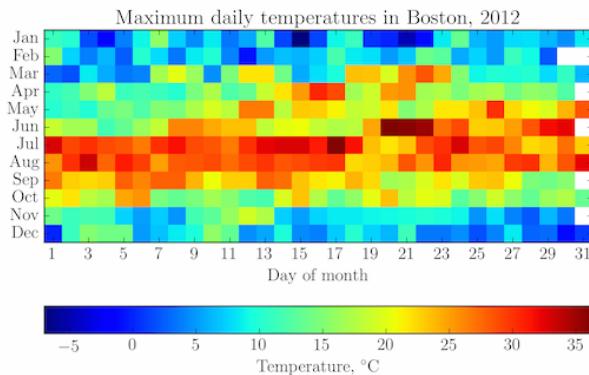


Figura 3.5: Representación de un mapa de calor usando matplotlib

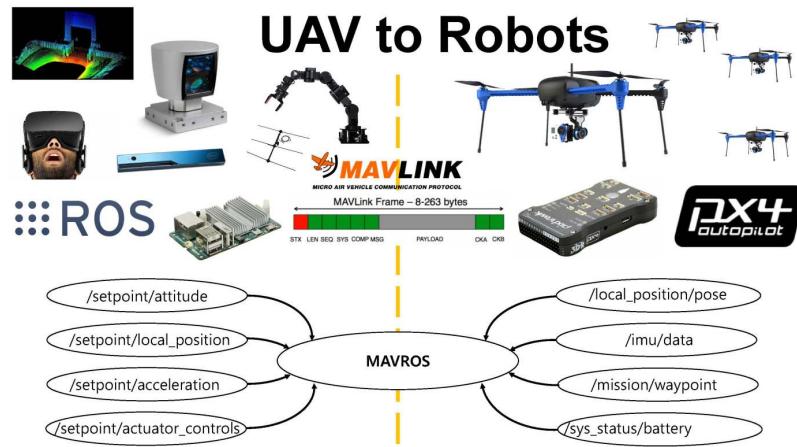


Figura 3.6: Panorámica de PX4 con ROS

comportamiento de una señal RF, permitiendo modificar los parámetros de la ecuación en tiempo real.

### 3.5.3. PX4 autopilot

Es la capa de software que permite hacer funcionar las aeronaves con sus componentes, esto es a través del controlador de vuelo, que a efectos prácticos se trata del cerebro del sistema, es decir, interconecta los sensores y actuadores, permitiendo comandar diversas acciones<sup>14</sup>.

Este sistema, usa un conocido protocolo de comunicaciones llamado **MAVLink**

---

<sup>14</sup><https://www.droneblog.com/drone-controller/> <https://docs.px4.io/main/en/>

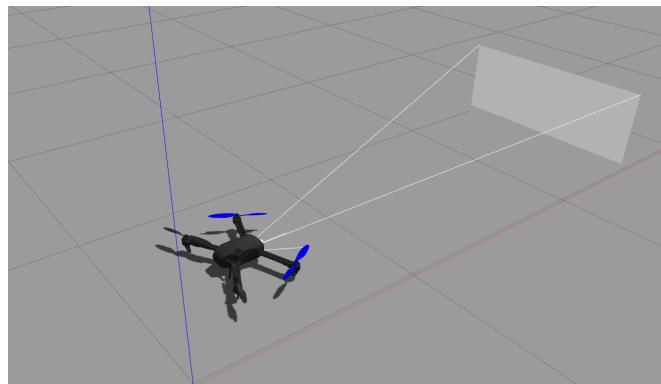


Figura 3.7: Iris drone en Gazebo 11

(ver figura 3.6), que se encarga de gestionar la comunicación entre el controlador de vuelo y la GCS. En nuestro caso, y como queremos desarrollar aplicaciones mediante ROS, debemos añadir una capa más, que se encarga de traducir los mensajes ROS a mensajes compatibles con el protocolo MAVLink, y de esto se encarga **MAVROS**<sup>15</sup>.

### 3.6. Iris

Es el nombre de la aeronave representada en la figura 3.7 y usada para solucionar los problemas planteados. En síntesis, es un dron cuadracóptero provisto de una cámara y un sensor de RF simulado. Dicha aeronave es cortesía de **JdeRobot**, que es una organización sin ánimo de lucro, asociada a *RoboticsLabURJC*, que provee de un conjunto de herramientas pensadas para desarrollar aplicaciones robóticas<sup>16</sup>.

---

<sup>15</sup><https://docs.px4.io/main/en/middleware/mavlink.html> <https://docs.px4.io/main/en/ros/ros1.html>

<sup>16</sup><https://jderobot.github.io/>

---

## Capítulo 4

# Diseño

---

En este capítulo se expone, de forma detallada, el proceso seguido para conseguir que un dron detecte y navegue autónomamente hacia una señal RF. Además, se muestra el desarrollo de una aplicación responsiva, que simula el comportamiento de una señal (en un espacio libre de obstáculos), basada en la aproximación de Friis. A continuación, en la figura 4.1, se muestra un diagrama de bloques donde se ve un esquema del sistema empleado.

Tal y como se puede apreciar, se tiene un controlador, encargado de comandar órdenes al dron; un simulador, que nos permite visualizar el modelo y el comportamiento del dron; el propio dron, que devuelve datos a otros bloques, tales como su posición o la imagen de la cámara instalada, entre otros; Rviz, como bloque de depuración y visualización; MPSS, o Módulo de Propagación de Señal Simulado, encargado de gestionar la transmisión de información del bloque de algoritmos de navegación autónoma y Rviz, a través del modelo de propagación de señal; bloque de comportamientos autónomos, que almacena la algoritmia responsable de resolver la navegación autónoma hacia el transmisor de la señal RF, en el se incluyen los ANAIV, o Algoritmos de Navegación Autónoma basados en la Información de la Vecindad (que extraen la información de la cercanía para navegar), y los RL, que son los algoritmos basados en Reinforcement Learning; y por último, el bloque de teleoperación, el cual se encarga de solicitar órdenes al controlador para desencadenar acciones en el dron, tales como desplazamientos y giros.

### 4.1. Preparación del entorno

Inicialmente, se pone en funcionamiento el entorno de simulación (compatible con ROS), así como el sistema de control de versiones, para mantener la trazabilidad y los backups. Por ello, se crea un repositorio común en GitHub y se instala y prueba el paquete de herramientas dispuesto por JdeRobot.

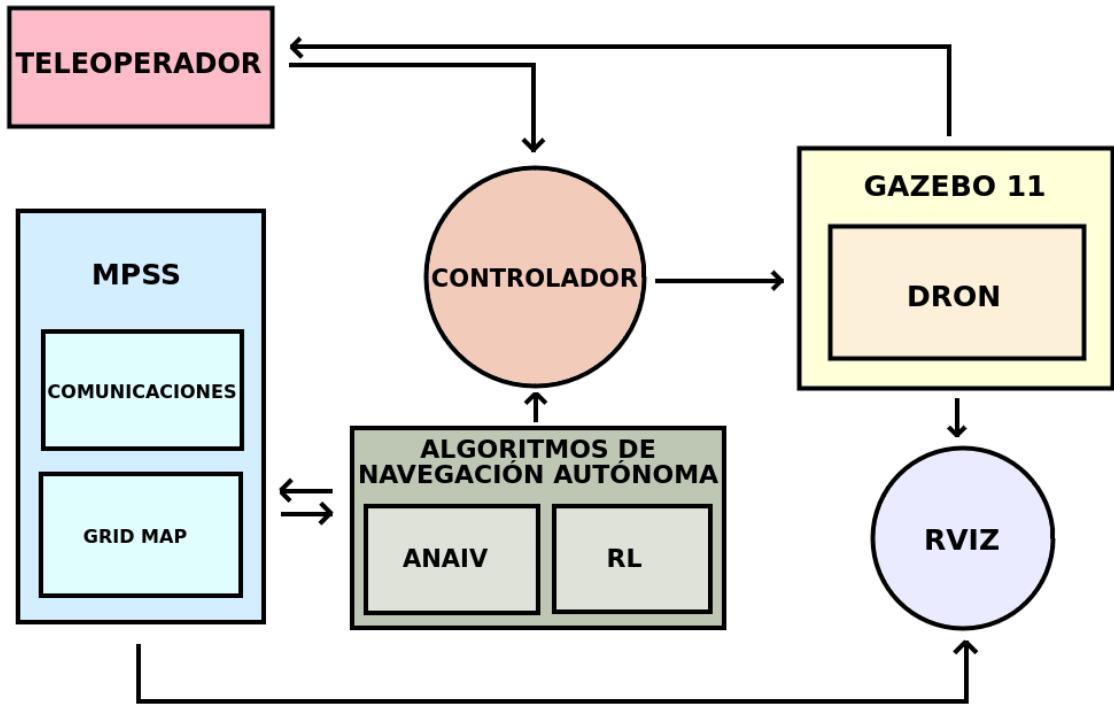


Figura 4.1: Diagrama esquemático del sistema completo

#### 4.1.1. JdeRobot - drones

Gracias a esta plataforma, se obtienen los modelos y módulos necesarios para simular en Gazebo 11 un cuadracóptero, provisto de un sistema autopilot PX4. Específicamente, el modelo usado es el 3DR Iris simulado, con un plugin de una cámara frontal, el cual se ha incluido manualmente en el fichero original, debido a que el modelo específico disponible no funcionaba correctamente. Este dispositivo utiliza MAVROS para realizar la comunicación, lo que nos permite enviar y recibir mensajes ROS compatibles con el protocolo de comunicaciones típico de estas aeronaves, MAVLink.

#### 4.1.2. Teleoperador

Una vez es funcional el entorno y los modelos, la primera aproximación ha consistido en realizar una interfaz gráfica, para enviar órdenes a la aeronave a modo de teleoperador. Para ello, se debe conseguir enviar mensajes de forma programática. Por tanto, se diseña un script controlador encargado de la comunicación directa con el controlador de la aeronave, que a su vez se encarga enviar y recibir diversos datos vía MAVROS. De igual modo, se satisfacen una serie de requisitos para asegurar el

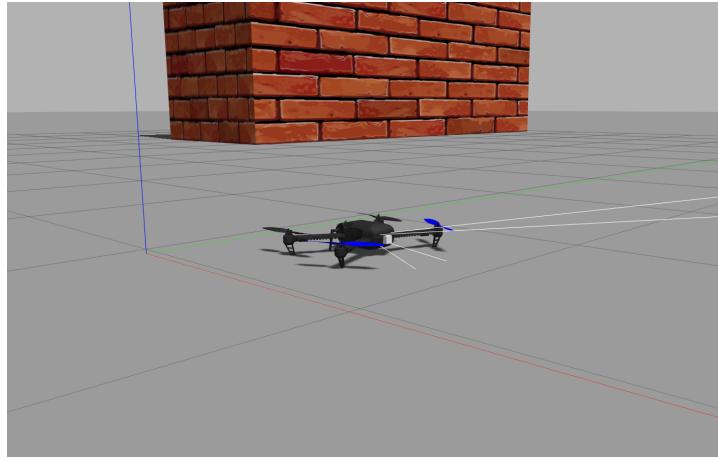


Figura 4.2: 3DR Iris simulado

correcto funcionamiento del sistema:

1. La comunicación se debe dar a más de 2 Hz, para evitar cambios indeseados en el funcionamiento interno del controlador PX4.
2. Antes de realizar cualquier comunicación, se debe asegurar que el estado es “*connected*”, lo que significa que el dron esta armado y en modo *OFFBOARD* (nuestra aeronave posee 7 modos distintos, *HOLD*, que mantiene la posición, *RETURN*, que vuelve al punto de despegue, *MISSION*, que permite cargar rutas programadas con anterioridad, *TAKEOFF*, habilita el despegue, *LAND*, habilita el aterrizaje, *FOLLOW ME*, que permite seguir objetivos, y *OFFBOARD*, que permite comandar al dron sin necesidad de GPS, lo que es útil de cara al desarrollo de aplicaciones robóticas)<sup>1</sup>.
3. Una vez está conectado, se deben enviar velocidades al controlador PX4, con el fin de evitar el cierre de la conexión. Estos datos carecen de utilidad más que la de asegurar dicha conectividad.
4. Por último, y antes de enviar cualquier posición, velocidad o comando (distintos modos de actuación), se debe comprobar siempre que el modo activo es *OFFBOARD* y que el dron esta armado (listo para volar). En caso contrario, se debe solicitar al controlador, mediante servicios, dichos requerimientos.

Por tanto, la manera de generar comportamientos en el dron en sí, es mediante *topics*. Concretamente, los que genera MAVROS automáticamente cuando se lanza todo el sistema. Tal y como se comentó en apartados anteriores, estos *topics* sólo

---

<sup>1</sup>[https://docs.px4.io/main/en/getting\\_started/flight\\_modes.html](https://docs.px4.io/main/en/getting_started/flight_modes.html)

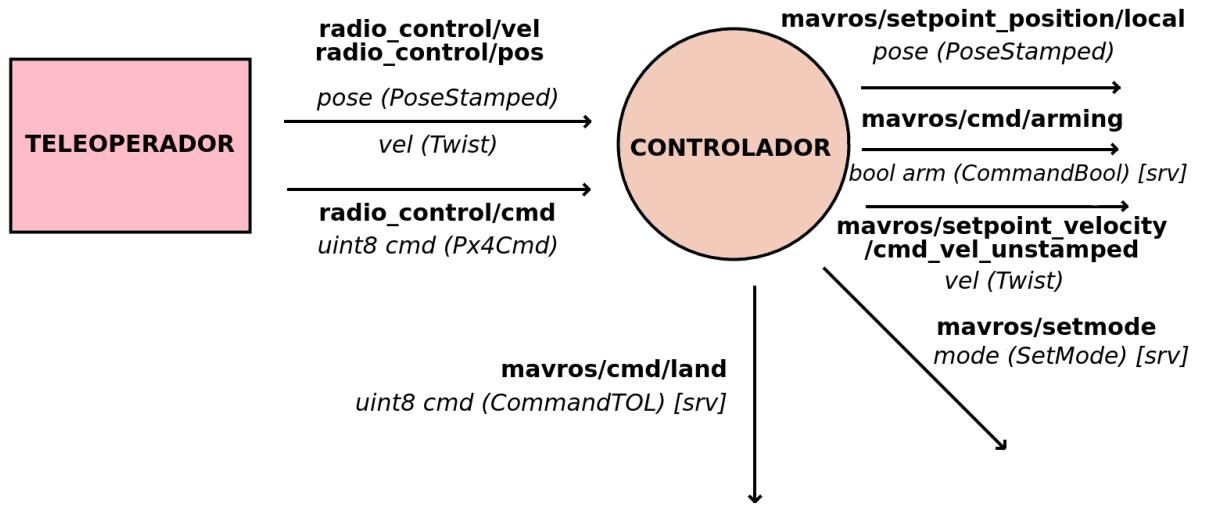


Figura 4.3: Esquema comunicaciones del controlador

admiten mensajes ROS, lo que encapsula el mensaje real transmitido al controlador PX4, que solo es compatible con MAVLINK. En nuestro caso, se envían posiciones (PoseStamped), velocidades (Twist) y comandos (sevicios formados por mensajes personalizados, creados por MAVROS, con formato ROS, como por ejemplo aterrizar). Esto, nos permite conectar el resto de aplicaciones con el script controlador, mediante *topics* comunes, es decir a través de “*radio\_control/pos*, *vel* o *cmd*”, de forma de que el script controlador se encargue de enviar la acción final al dron, empleando los *topics* propios de MAVROS, vease “*mavros/setpoint\_position/local*” (que solicita al dron una posición en coordenadas del simulador), “*mavros/setpoint\_velocity/cmd\_vel\_unstamped*” (que de igual modo solicita velocidades), “*mavros/cmd/land*” (para solicitar el aterrizaje), “*mavros/cmd/arming*” (para armar el dron, o dicho de otro modo, para que el dron este listo para actuar), y por último “*mavros/setmode*” (para solicitar un modo de actuación determinado). Para más detalle, mirar la figura 4.3, donde se especifican los tipos de mensaje enviados de forma gráfica.

De este modo, el teleoperador se diseña con el fin de ofrecer una interfaz gráfica, capaz de generar comportamientos que se usarán en las fases finales del TFG. Sin embargo, para la primera versión, tan solo se construye una interfaz gráfica, encargada

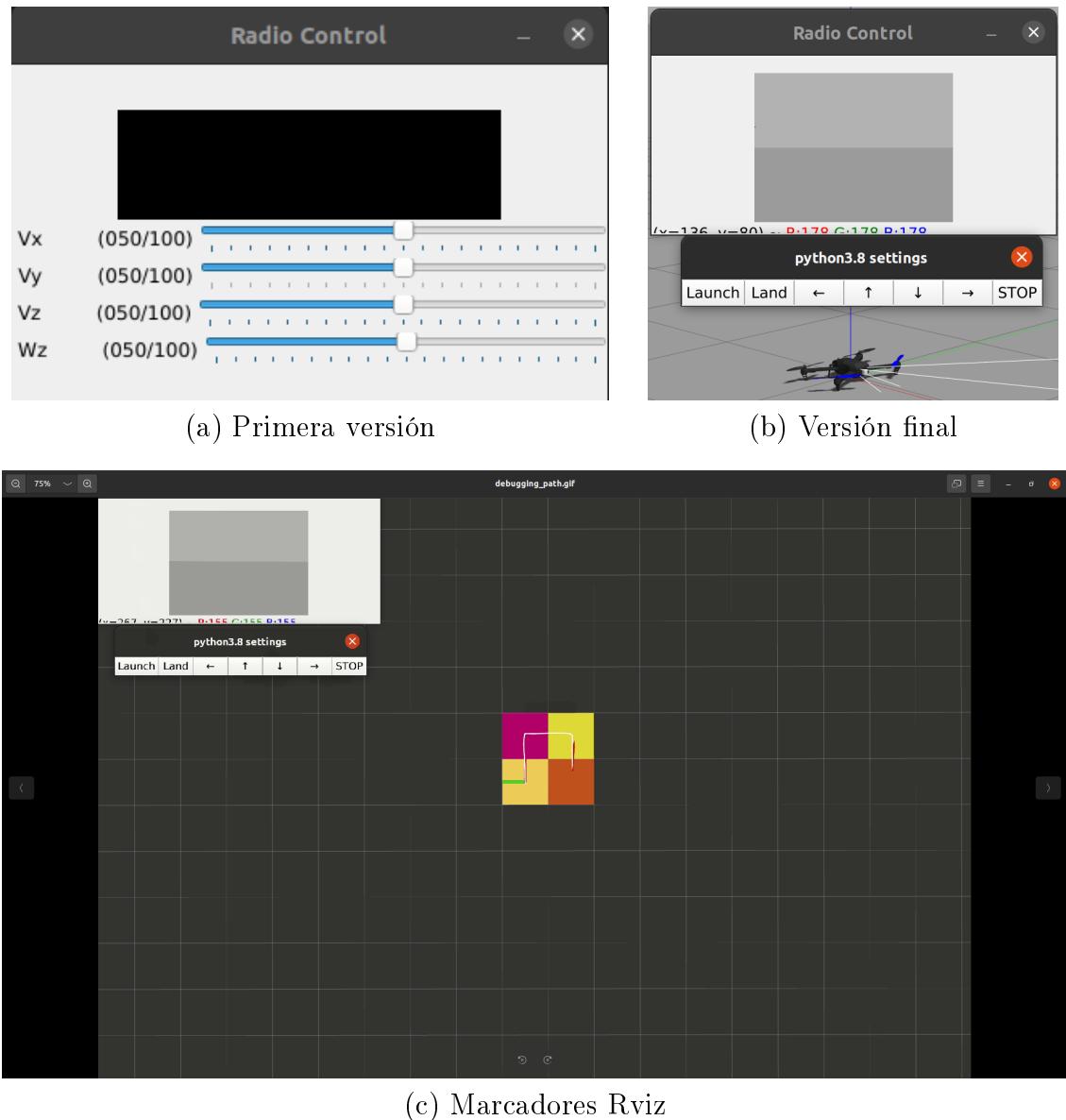


Figura 4.4: Teleoperador

de enviar órdenes usando ROS que, en última instancia, llegan al dron y producen diversos comportamientos, tales como moverse y girar, a través de barras de acción.

En la siguiente versión, se programan comportamientos predefinidos, es decir, acciones predeterminadas tales como desplazarse distancias concretas en ciertas direcciones, o girar un número específico de grados en un sentido u otro. Para ello, se diseña una ampliación sobre la interfaz anterior, en la que se añade un botón por cada acción concreta desarrollada, además de una imagen de la cámara frontal en tiempo real.

---

```

if __name__ == '__main__':
    try:
        rospy.init_node(NODENAME, anonymous=True)

        # Msgs
        ## Subscribers
        image_sub = rospy.Subscriber(IMAGE_TOPIC, Image, callback =
            image_cb)
        current_pos_sub = rospy.Subscriber(LOCAL_POSE_TOPIC, PoseStamped,
            callback = current_pos_cb)

        ## Publishers
        pos_pub = rospy.Publisher(RADIO_CONTROL_POS_TOPIC, PoseStamped,
            queue_size=10)
        cmd_pub = rospy.Publisher(RADIO_CONTROL_CMD_TOPIC, Px4Cmd,
            queue_size=10)

        # -- OPENCV -- #
        cv2.namedWindow(WINDOWNAME)

        # Buttons
        cv2.createButton('Launch', launch_button, None,
            cv2.QT_PUSH_BUTTON, 1)
        cv2.createButton('Land', land_button, None, cv2.QT_PUSH_BUTTON, 1)
        cv2.createButton('←', left_button, None, cv2.QT_PUSH_BUTTON, 1)
        cv2.createButton('↑', front_button, None, cv2.QT_PUSH_BUTTON, 1)
        cv2.createButton('↓', back_button, None, cv2.QT_PUSH_BUTTON, 1)
        cv2.createButton('→', right_button, None, cv2.QT_PUSH_BUTTON, 1)
        cv2.createButton('STOP', stop_button, None, cv2.QT_PUSH_BUTTON, 1)

        cv2.waitKey(0)
        cv2.destroyAllWindows()
    except rospy.ROSInterruptException:
        pass

```

---

Código 4.1: Bloque de código principal de la versión final del teleoperador

Por último, se simplifica la interfaz, con el fin de seleccionar los comportamientos más relevantes de cara al proyecto. Además, se agregan marcadores en *rviz*, para determinar las celdas visitadas (con colores aleatorios), junto con otro marcador que muestra la trayectoria que sigue la aeronave, como se puede observar en la figura 4.4.

En el código 4.1, se muestra la última versión comentada de la interfaz. En este caso se muestra el “*main*”, donde, tras inicializar el nodo ROS, se definen por un lado los suscriptores, encargados de recibir los datos de la cámara y la posición del dron (usando MAVROS), los publicadores, cuya función es enviar posiciones y/o comandos al script controlador, y por último la interfaz gráfica diseñada con OpenCV, donde se define la ventana y los botones con las diversas acciones predefinidas<sup>2</sup>.

## 4.2. Modelo de propagación de señal

El siguiente paso consiste en generar un modelo de propagación de señales RF. Este apartado es especialmente relevante, ya que nos permite desarrollar una aplicación reactiva, con la idea de generar entornos en tiempo real que simulan la propagación de una señal, sobre la que probar nuestras soluciones. Para entender bien todo, dejo una referencia en el anexo 5.3 con una serie de conceptos introductorios útiles para abordar esta sección.

En cuanto al modelo de propagación de señal, se define como un modelo que nos ayuda a calcular de que manera se propaga una señal por el aire, en función de la potencia, frecuencia, distancia y entorno, entraremos en detalle a continuación. La idea es implementar el modelo e integrarlo en ROS, ya que no hay nada desarrollado y de cara a resolver problemas de este tipo, puede ser realmente útil.

### 4.2.1. Aproximación de Friis

Se opta por emplear la aproximación de Friis, presentada por el ingeniero Danés-American Harald T. Friis en 1946, el cual buscaba una manera de medir el rendimiento de una antena [13]. De este modo, nos ofrece la siguiente ecuación para modelar nuestro problema:

---

<sup>2</sup>Código completo en [https://github.com/RoboticsLabURJC/2022-tfg-cristian-sanchez/blob/main/src/teleop/scripts/c2c\\_control.py](https://github.com/RoboticsLabURJC/2022-tfg-cristian-sanchez/blob/main/src/teleop/scripts/c2c_control.py)

Entorno	Exponente n
Sin obstáculos	2
Zona urbana	2.7 a 3.5
Zona urbana densa	3 a 5
Interior (a la vista)	1.6 a 1.8
Interior (obstaculizado)	4 a 6
Almacen (obstaculizado)	2 a 3

Tabla 4.1: Valores n de referencia

$$P_r = P_t \cdot \left( \frac{G_t G_r \lambda^2}{(4\pi)^2 d^n L} \right) \quad (4.1)$$

Donde cada término significa lo siguiente<sup>3</sup>:

1.  $P_t$  y  $P_r$ : aluden a la potencia del transmisor y la potencia del receptor respectivamente.
2. *Ganancias* ( $G_t$ ,  $G_r$ ): representan un valor incremental aplicado a la potencia de emisión y de recepción, respectivamente.
3. *Valor lambda* ( $\lambda$ ): hace referencia a la longitud de onda. Está directamente relacionado con la frecuencia.
4. *Distancia* (d): desde el origen de la señal a un punto en el espacio.
5.  $L$ : representa todas aquellas pérdidas no asociadas a la propagación de la señal.
6. *Path-Loss Exponent* (PLE) (n): permite ajustar el modelo a diversos entornos. Es un valor constante extraido de forma empírica. Como se puede observar en la figura 4.1.

Además, empleando este método, podemos modelar las pérdidas de una señal estimadas durante su propagación, a través de la siguiente ecuación:

$$P_L(dB) = -10 \log_{10} \frac{\lambda^2}{(4\pi d)^2} \quad (4.2)$$

---

<sup>3</sup><https://www.gaussianwaves.com/2013/09/friss-free-space-propagation-model/>

### 4.2.2. Módulo python de Friis

Una vez desarrollado lo anterior, lo siguiente consiste en encontrar la forma de que todo esto sea accesible para cualquier aplicación que lo deseé en el ecosistema Python. Debido a esto, surge la idea de crear un módulo python encargado de modelar las ecuaciones previamente mencionadas. Para ello, se diseña una clase cuyo constructor recibe, por parámetros, las variables implicadas en las ecuaciones de Friis, además de las dimensiones del mapa y su resolución (la cual afecta al tamaño de celda).

Básicamente, el proceso a seguir para usar este módulo es el siguiente, primero se crea un objeto de la clase Friis donde se especifican las características de la señal y las variables relacionadas, lo que genera internamente un array 2D vacío, que será rellenado en función de la configuración seleccionada. Posteriormente, se selecciona el modelo deseado (propagación o pérdidas), pasando las coordenadas del origen de la señal por parámetros. Esto, retornará el array lleno con los valores de potencia asociados a las ecuaciones del modelo de Friis seleccionado. A continuación, en el código 4.2, se muestra un ejemplo de uso sencillo, donde se obtiene un mapa de propagación de señal, en forma de Numpy array 2D.

---

```
#! /usr/bin/env python
import friss as fr

if __name__ == '__main__':
    friis_object = fr.Friis(power_tras=10.0,
                           gain_tras=1.5,
                           gain_recv=2.0,
                           freq=fr.FREQ_WIFI,
                           losses_factor=1.0,
                           losses_path=2.0,
                           world_sz=(10,10),
                           resolution=1.0)

    signal_map = friis_object.model_power_signal(origin=(5,3))
```

---

Código 4.2: Ejemplo básico de uso del módulo Friis

Concretamente, se genera un array 10x10 con resolución 1 (es decir, representa celdillas de 1x1 metros), de una señal WIFI (2.4 GHz), donde el transmisor emite a 10 W, con una ganancia de 1.5, el receptor posee una ganancia de 2, un factor de pérdidas (L) de 1, es decir, sin pérdidas, y por último, el exponente n (PLE) con un valor de 2, que representa el espacio vacío. Luego se genera el modelo de propagación de la

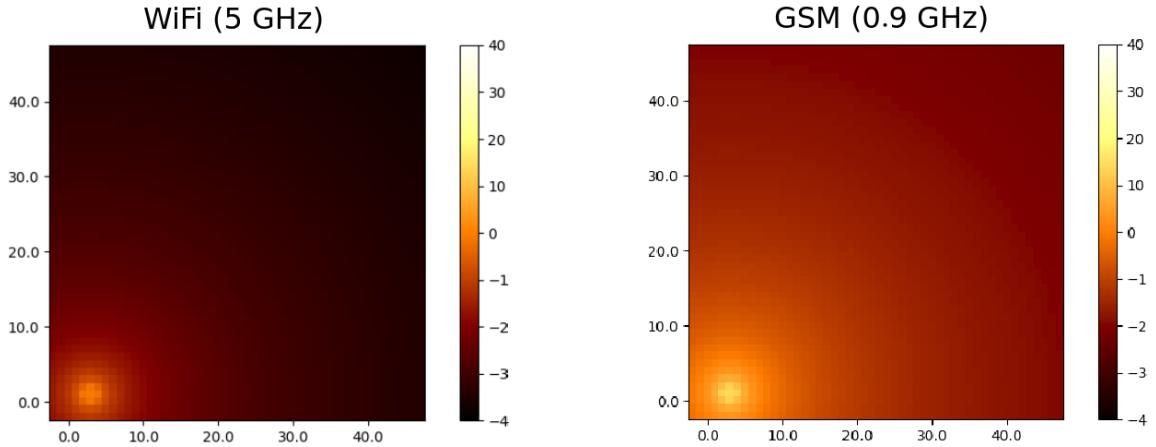


Figura 4.5: Uso del módulo Friis con señal WIFI (2.4 GHz) VS señal GSM (0.9 GHz)

señal, indicando que la fuente se encuentra en las coordenadas (5,3). En la figura 4.5, se muestra una representación del código 4.2, pero para un mapa de mayor dimensión (50 x 50 metros), de modo que se aprecie mejor las diferencias entre unos y otros.

Adicionalmente, este módulo posee otras funcionalidades útiles para trabajar con él, como son:

1. *reset\_world*: modifica el mapa que hubiera, estableciendo todos sus valores a cero.
2. *get\_world\_sz*: retorna las dimensiones del mapa.
3. *set\_values*: modifica las características de la señal simulada.

Hay que tener en cuenta que, aunque se modifiquen los parámetros, se debe modelar de nuevo el mapa para que surtan efectos los cambios.

#### 4.2.3. Aplicación de Friis

Finalmente y agrupando lo anterior, se integra el módulo previo, en una interfaz gráfica intuitiva que ha servido para depurar. La idea es estudiar, en tiempo real, como evoluciona la señal cuando algunos de sus parametros, en la ecuación de Friis, son modificados.

Para ello, se emplea la librería matplotlib, debido a la enorme funcionalidad que dispone, así como de su sencillez a la hora de crear nuevas aplicaciones. Funciona de

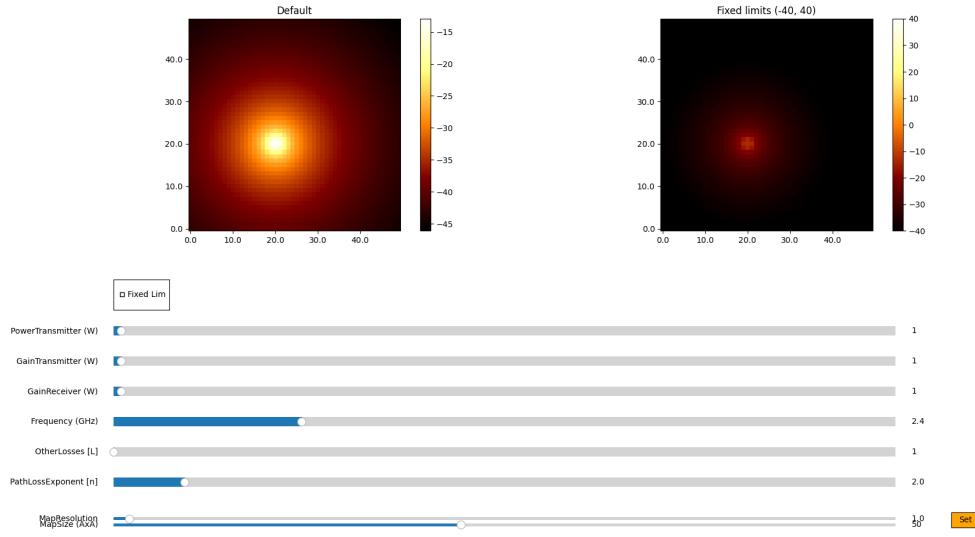


Figura 4.6: Versión final de la interfaz

manera que se generan eventos que son gestionados en “*callbacks*”, es decir, se generan bucles asociados a dichos eventos, que reaccionan a cambios en la interfaz generada. En nuestro caso, la estructura base consta de una figura, sobre la que se agregan todos los elementos, entre los que encontramos los mapas de calor o “*heatmap*” en forma de plots, las barras de acción o “*sliders*”, botones, entre otros elementos que se explicarán más adelante.

Inicialmente, se opta por representar un mapa de calor con una barra de color asociada a los distintos valores de la señal. Además, se integran sliders correspondientes a cada valor presente en la ecuación de Friis. El problema es, que al actualizar los valores, también lo hace la representación, por lo que no se aprecia el efecto de los cambios en el plot. Por ello, se decide agregar dos mapas de calor, uno con el máximo y el mínimo fijados a mano (donde sí se aprecian los cambios), y el anterior mencionado. Para elegir cual usar, se añade una casilla marcable. Además, se incluyen dos variables relevantes a la hora de modelar, el tamaño del mapa y la resolución, manejadas a través de “*sliders*”, los cuales a su vez se activan al pulsar un botón de SET, que recarga la interfaz. Además, se ajustan los saltos de valores para que sean coherentes en el resto de barras de acción, tal y como se puede apreciar a continuación:

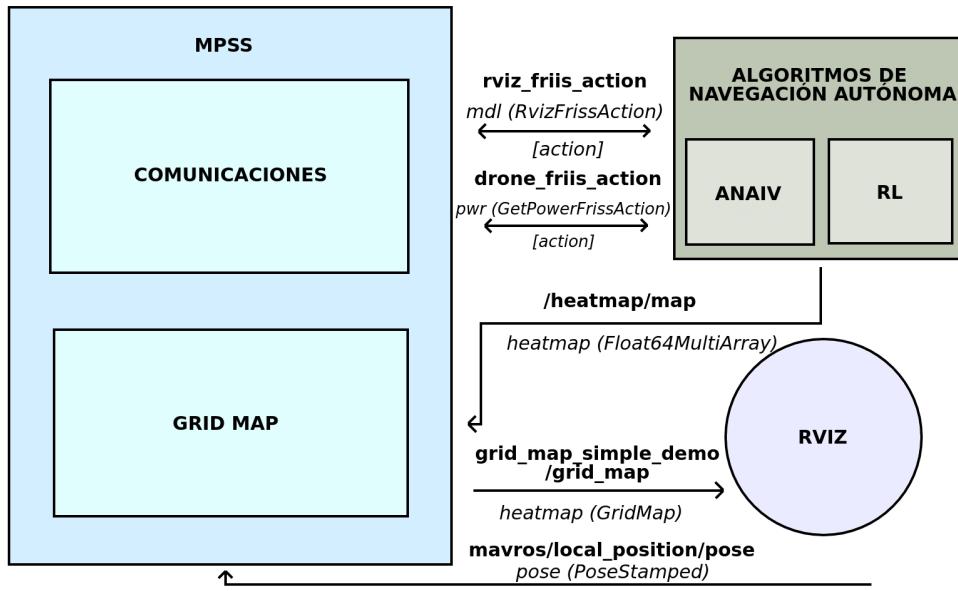


Figura 4.7: Esquema comunicaciones MPSS

#### 4.2.4. MPSS - Integración de Friis con ROS

Cómo se comentó al inicio de la sección 4.2 y siguiendo el esquema de la figura 4.7, el módulo de propagación de señal simulado, se ha diseñado como una aplicación servidor de datos, que funciona como intermediaria entre el módulo de Friis y otros nodos ROS (en este caso, con un diseño enfocado a simular de manera realista el escenario con el dron). Básicamente, la aplicación hace uso del módulo de Friis para gestionar la información que envía a según que aplicación. Este programa se compone de dos servidores basados en acciones ROS, los cuales son especialmente útiles dada su naturaleza asíncrona. Dichos servidores, se encargan de las peticiones y respuestas para el dron y para rviz, tal y como se detalla a continuación:

1. *Bloque de comunicaciones*: en términos generales, el dron envía su posición en coordenadas transformadas al sistema de referencia del “*heatmap*”, y recibe el valor de la señal de dichas coordenadas, mediante acciones ROS (“`drone_friis_action`”). En un caso real, el dron tan solo accedería al valor de la señal a través de un sensor que se lo permitiera (por ejemplo un receptor RF como el RTL-SDR V3<sup>1</sup>). Adicionalmente, se le ha agregado la funcionalidad de enviar, en dicha petición, si se deseaba un mapa con obstáculos o no en la misma petición (de manera que MPSS generase un modelo con obstáculos). En paralelo, recibe una petición para agregar todas las características de la señal y sobre la que generar

<sup>1</sup>Más información aquí <https://www rtl-sdr.com/about-rtl-sdr/>

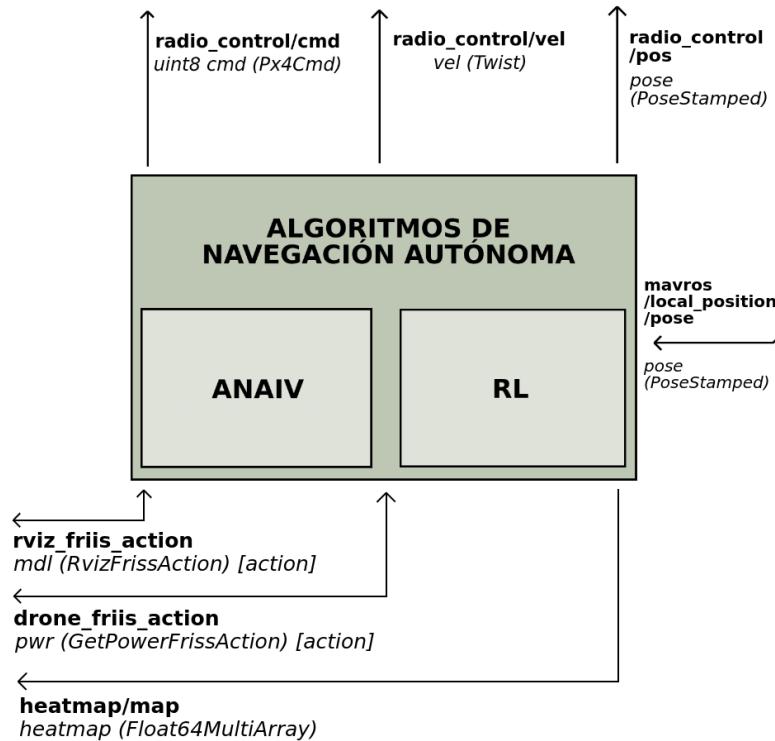


Figura 4.8: Esquema comunicaciones del bloque de comportamiento autónomo

el “*heatmap*”, vease el origen y sus componentes. Esto, genera como respuesta un array de floats que contienen la información del mapa de calor, en un formato adecuado para su representación. Todo ello, nuevamente a través de acciones ROS, en “*drone\_friis\_action*”. De igual modo, ha sido agregada la funcionalidad de los obstáculos para la experimentación futura.

2. *Bloque grid map*: se encarga de transformar el mapa de calor en formato array, a un objeto compatible con la biblioteca *grid\_map*, que a través del topic de ROS “*heatmap/map*”, permite enviar los datos a un nodo que los transforma y reenvía a un plugin de rviz (usando el topic “*grid\_map\_simple\_demo/grid\_map*”), el cual genera la representación visual buscada<sup>4</sup>.

### 4.3. Navegación autónoma para localizar el origen de una señal RF

Tal y como se observa en la figura 4.8, el bloque de algoritmos de navegación autónoma se comunica con el controlador para enviar las órdenes al dron, tal y como sucede con el teleoperador (vease sección 4.1.2), enviando a través de los mismos *topics*

---

<sup>4</sup>Toda la funcionalidad englobada en el directorio *heatmap\_util* del proyecto

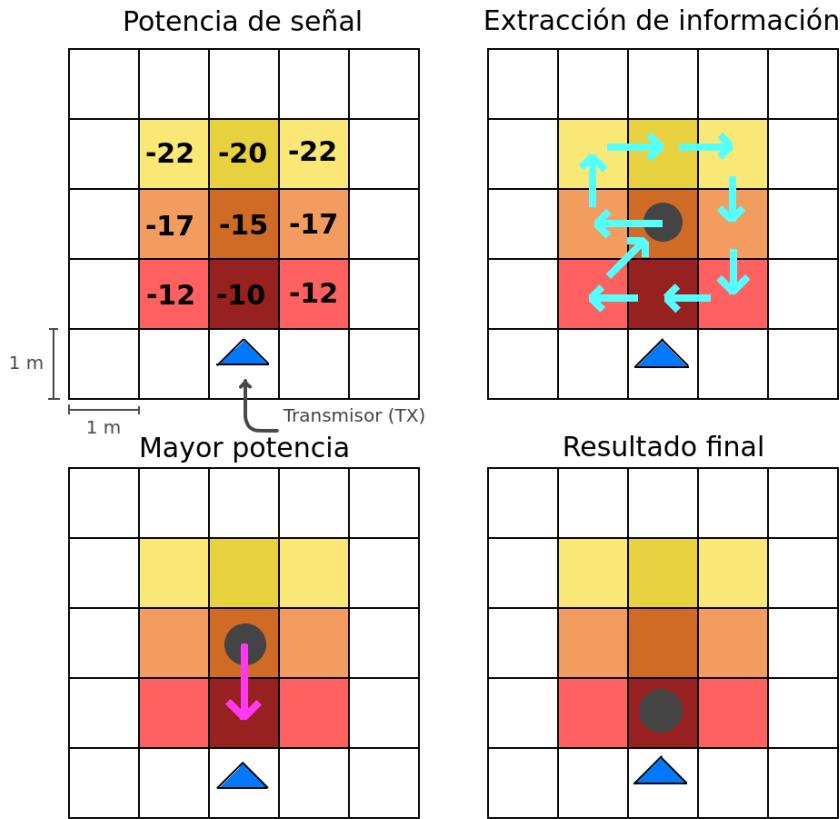


Figura 4.9: Algoritmo de navegación autónoma basado en la información de la vecindad

mencionados la información. Además se comunica con el MPSS, tal y como se describió previamente en la figura 4.7. Adicionalmente y para el correcto funcionamiento de algunas funciones, recibe información de la posición del dron en el simulador a través del topic “*mavros/local\_position/pose*”.

#### 4.3.1. ANAIV

En los algoritmos de navegación autónoma basados en la información de la vecindad, parte de una clase “*Drone*” que se usa como base para todas las soluciones desarrolladas del bloque de algoritmos de navegación autónoma, cuyo constructor se encarga de conectar los topics al controlador PX4 para comandar ordenes a la aeronave. Además, se encarga de establecer la comunicación con el servidor de datos (tanto para la potencia como para rviz) y define los diferentes atributos pertenecientes a la clase, que en este caso aluden a los parámetros necesarios para el funcionamiento de los algoritmos y la extracción de datos en los resultados, siguiendo esta estructura:

1. *Métodos para comandar al dron*: que se definen como el conjunto de funciones

encargadas del movimiento del dispositivo (como despegar, aterrizar, desplazarse, entre otros). Mucha de esta funcionalidad ha sido adaptada del teleoperador comentado al de este capítulo.

2. *Métodos de tolerancia*: encargados de establecer un margen aceptable entre la posición del dron y el objetivo deseado. Estos métodos permiten controlar con precisión problemas que surgen de la deriva y de condiciones externas, como puede ser el viento.
3. *Métodos de conversión*: que permiten transformar las coordenadas entre los distintos sistemas de referencia presentes en el problema, tal y como se puede apreciar a continuación.
4. *Algoritmos*: o las soluciones “*sigue señal*” propiamente dichas, que en sí, contienen el conjunto de métodos que cada cual necesita para llevarse a cabo. Se puede distinguir entre manual, manual optimizado y Q-Learning.

Adicionalmente, se deben cumplir una serie de premisas de cara a la simulación. Estas son que, todos los movimientos realizados por el dron deben estar contenidos en el mapa de calor generado; además, la medida de la señal sólo podrá tomarse cuando el dron esté en el centro de la celda; los movimientos del dron deberán ser de centro en centro aunque esto abarque más celdas de distancia (problema resuelto y adaptado del teleoperador); y que la métrica de cada celda es de 1x1 metros.

Por ello, la primera aproximación se basa en visitar todos los vecinos más cercanos (un total de 9 posiciones) y realizar el movimiento que lleve a una posición con más potencia, tal y como se puede apreciar en la figura 4.9. Para ello, se toma la medida del sensor a bordo (simulado) en cada una de las posiciones vecinas disponibles, de donde se obtiene un valor en dB, de modo que las coordenadas candidatas para ser objetivo, son aquellas que posean mayor valor de señal. Posteriormente se realiza el desplazamiento y se repite el proceso hasta llegar a una condición de finalización o parada. En este caso, la condición de parada analiza si las coordenadas objetivo de la iteración anterior, son las mismas que las coordenadas objetivo de la iteración actual, cumpliendo además que todos los vecinos colindantes tienen un valor de la señal inferior al candidato a origen de la señal.

En cuanto a los métodos que se usan, se encuentra el de verificar movimientos válidos y el de comprobar si ha llegado a la casilla final, mediante la verificación anterior

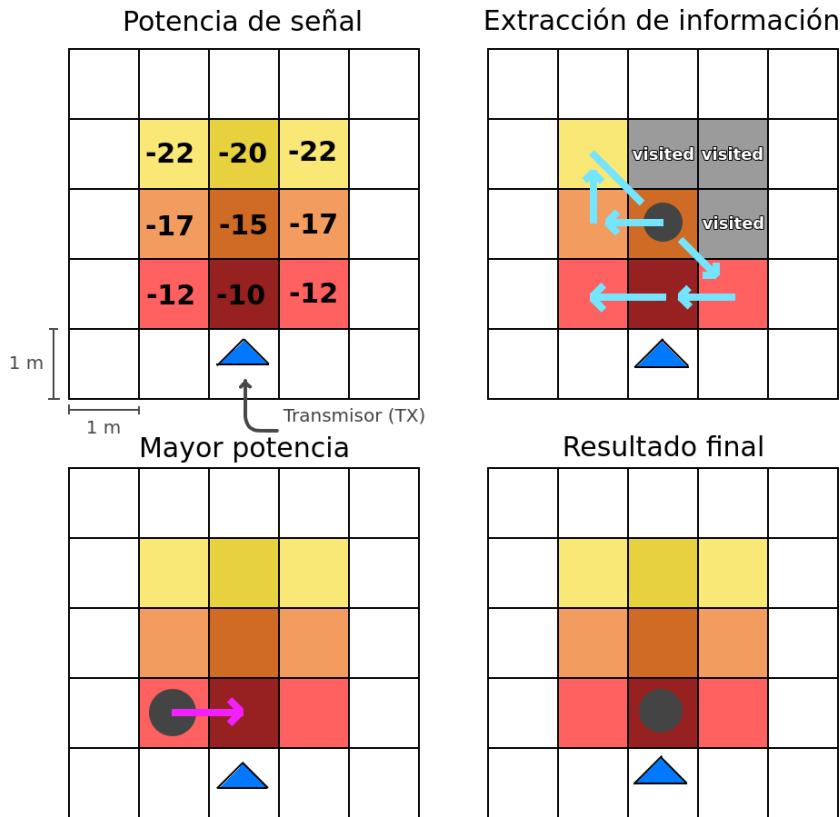


Figura 4.10: Algoritmo de navegación autónoma basado en la información de la vecindad (mejorado)

(vecindad con menor señal).

#### 4.3.2. ANAIV mejorado

Teniendo en cuenta lo anterior, se han agregado ciertas mejoras y eficiencia. El principio es el mismo, se obtiene la información de los vecinos y se navega hacia el mejor candidato. Sin embargo, en este caso no se revisita vecinos cuya información se conozce, como podemos ver en la figura 4.10. Para ello, se ha implementado un array que almacena hasta 18 coordenadas de vecinos ya visitados, de modo que solo se navega hacia coordenadas nuevas, y que por supuesto cumplan las condiciones del problema (no salirse del mapa de calor, moverse de centro a centro, entre otras). Además, la condición de parada es idéntica a la anterior, y los métodos usados también.

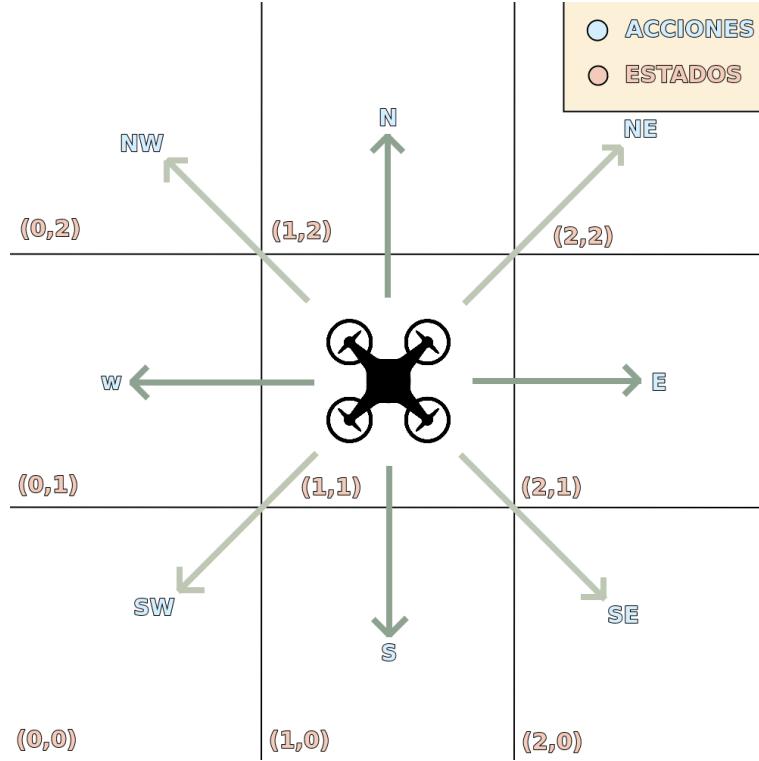


Figura 4.11: Representación de estados y acciones

#### 4.3.3. Algoritmo de navegación basado en RL

Por último, se ha planteado un algoritmo de navegación basado en técnicas de aprendizaje por refuerzo. Concretamente empleando Q-Learning, que tal y como comentamos en la sección 1.2.1, consiste en la obtención de una tabla Q, de estados y acciones, donde se asignan valores numéricos cada acción según su estado, de modo que la acción más favorable acaba teniendo mayor valor numérico que el resto. Esto es a través de las recompensas, donde si la acción es buena, la recompensa es positiva y el valor asignado es mayor que en el caso contrario. En nuestro caso, los estados son las coordenadas del dron en términos del mapa de calor, y las acciones son los movimientos cardinales y diagonales, de una o más celdas de distancia, tal y como se puede observar en la figura 4.11

Como todo algoritmo basado en RL, se distinguen dos fases: la fase de entrenamiento, cuyo objetivo es llenar de forma eficaz la tabla Q, y la fase de inferencia, donde se prueban los resultados obtenidos durante el entrenamiento.

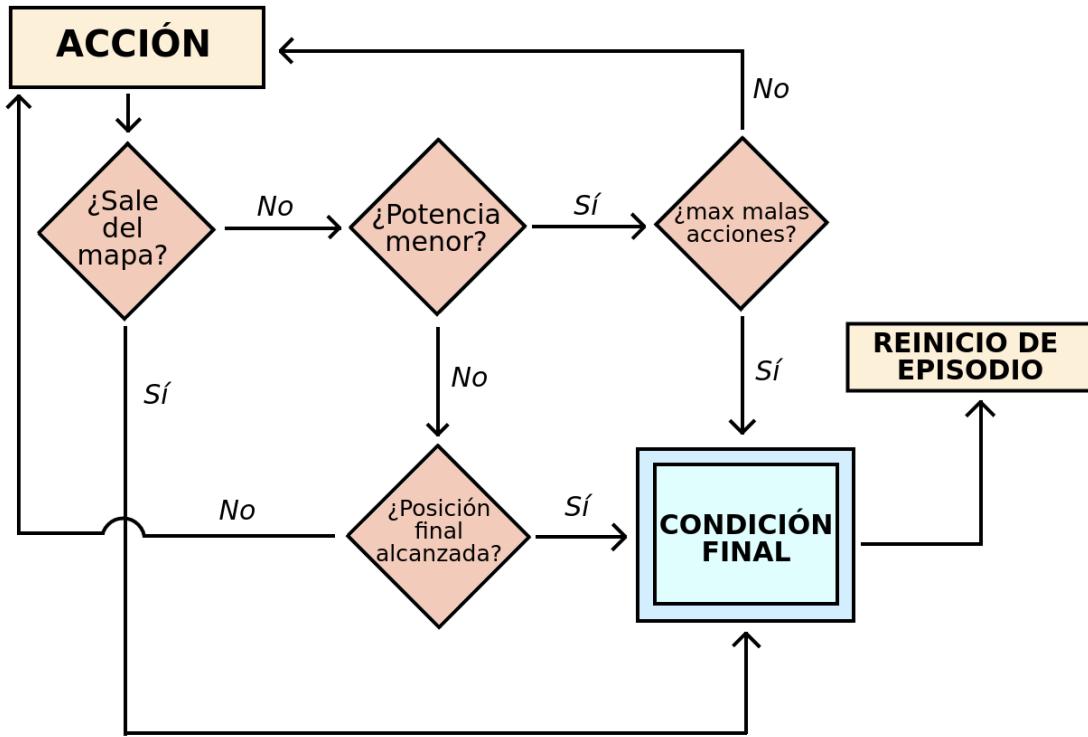


Figura 4.12: Diagrama condición de finalización

### Fase de entrenamiento

Dentro del entrenamiento, se definen episodios como intervalo de eventos dentro del algoritmo, que abarca desde el inicio o despegue del dron hasta donde se alcanza una condición de final, vease llegadas a la casilla final, o a casillas fuera del mapa definido; y las iteraciones, que se definen literalmente como la realización de una vuelta del bucle, es decir, desde obtener la medida de potencia de señal de la posición actual, hasta realizar una acción, tomar la nueva medida y actualizar los valores pertinentes. Además, para llenar el contenido de la tabla, se han definido las pertinentes recompensas y penalizaciones basadas en la diferencia entre la medidas, antes y después de realizar una acción (agregando un pequeño multiplicador a las recompensas negativas), contemplando casos extra, como cuando el dron se sale del mapa, donde se establece una recompensa fija negativa, calculada en proporción al resto de recompensas. Posteriormente, se asignan los valores obtenidos en la tabla Q, haciendo uso de la ecuación 4.3, que representa la ecuación de Bellman.

$$Q(s, a) = (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a')) \quad (4.3)$$

Cabe destacar que, durante el entrenamiento, se especifican una serie de parámetros que han sido ajustados a través de la experimentación, estos son: el número de episodios

Parámetro	Valor
Número de episodios totales	10000
$\alpha$	0.4
$\gamma$	0.7
$\epsilon$ (inicial)	0.99
$\epsilon$ (final)	0.1

Tabla 4.2: Parámetros usados durante todos los entrenamientos

totales, que repercute directamente en la *fase de exploración* (detallado a continuación); el parámetro  $\alpha$ , o la tasa de aprendizaje, que afecta a la convergencia de las soluciones durante el aprendizaje; el parámetro  $\gamma$ , o factor de descuento, que alude a la importancia de las acciones futuras con respecto a las inmediatas; y por último los valores de epsilon ( $\epsilon$ ), que determinan si la acción tomada será aleatoria o extraida de la tabla, esto está directamente asociado a la *fase de exploración*, donde se prioriza la aleatoriedad con el fin de enriquecer con información la tabla Q. En la tabla 4.2 se muestran los parámetros finales seleccionados durante todos los entrenamientos y experimentos realizados.

La fase de exploración ocupa un 20 % del número de episodios, de forma lineal, es decir, que cada vez hay más probabilidad de que se tome la acción con mayor peso en la tabla y no de tomar una acción aleatoria (durante el entrenamiento siempre se mantiene cierta posibilidad de tomar una acción arbitraria, para seguir actualizando los datos). Además, para evitar el overfitting (o sobreajuste al entrenamiento, lo que reduciría la adaptabilidad del modelo a escenarios distintos a los del entrenamiento), se han establecido distintos puntos de inicio para el entrenamiento, para el inicio de los episodios, repartidos de forma uniforme por el mapa, tal y como se mencionará en la sección de métricas.

Cabe destacar que, si la acción tomada lleva al dron hacia una condición de final, este finaliza automáticamente el episodio, viajando hacia una nueva posición de entrenamiento y actualizando ciertos parámetros, como es el caso del parámetro  $\epsilon$ . La condición de final se aplica siempre tras actualizar los valores intrínsecos en la iteración, tal y como se puede apreciar en la figura 4.12. De forma general, en la actualización de parámetros, se incluye la adición de pesos en la tabla Q según la función de recompensa y un factor aplicado, lo que se traduce en la ecuación 4.4.

$$r = (P_f - P_o) \cdot \text{factor} \quad (4.4)$$

Donde  $P_f$  representa el valor leido de la potencia de la señal en la nueva posición (tras

haber realizado la acción),  $P_o$  es el valor de la lectura obtenida antes de realizar la acción, y *factor* equivale a un valor numérico fijo establecido en función de si se le quiere dar más importancia a las recompensas positivas y/o negativas (en nuestro caso es de 1.05 sólo para las recompensas negativas). El orden de magnitud trabajado para esta diferencia se encuentra en torno a las unidades, por ello, se establece el caso de que el dron salga del mapa con una recompensa fija de “-10”, que escala al orden de las decenas, lo que nos permite penalizar más dichas decisiones, tal y como se puede apreciar en el código 4.3.

---

```

if pwr_next == NOT_VALID_POWER:
    reward = -10
    error = reward - q_table[current_state_idx, current_action_idx]
else:
    reward = pwr_next - pwr_current
    if target_coords_hm != next_coords_hm and reward < 0:
        reward *= 1.05
    next_state_idx = self.get_coord_state_idx(next_coords_hm, states)
    error = (reward + gamma * np.max(q_table[next_state_idx])) -
            q_table[current_state_idx, current_action_idx]
    q_table[current_state_idx, current_action_idx] += alpha * error

```

---

Código 4.3: Código simplificado de la función de recompensa y actualización tabla Q

Una vez realizado el entrenamiento, podemos analizar su rendimiento a través de la figura 4.13, en este caso, un gráfico triple que nos permite conocer en detalle características del proceso. En concreto, representa tres métricas: el valor de epsilon ( $\epsilon$ ), en el que se distingue la fase de exploración; la recompensa acumulada, que nos permite analizar la convergencia del entrenamiento; y el número de iteraciones, donde se observa que conforme el algoritmo aprende, el número se reduce. Todo ello con respecto a cada episodio. Observamos como en torno al episodio 900 la recompensa acumulada se estabiliza, lo que es una señal de convergencia que indica que el dron alcanza el origen de la señal. Además, conforme se va acabando la fase de exploración, en el episodio 1000, se ve que el número de iteraciones se reduce significativamente, lo que indica menos acciones aleatorias y mayor convergencia hacia una condición de final. En cuanto al tiempo empleado por entrenamiento, usando un portátil ASUS ROG Strix G513QR, con 32 GB de memoria RAM, AMD Ryzen 7 5800h y tarjeta gráfica NVIDIA 3070, tarda aproximadamente 1 hora y 20 minutos, donde en torno a 30 minutos son necesarios para la convergencia del modelo.

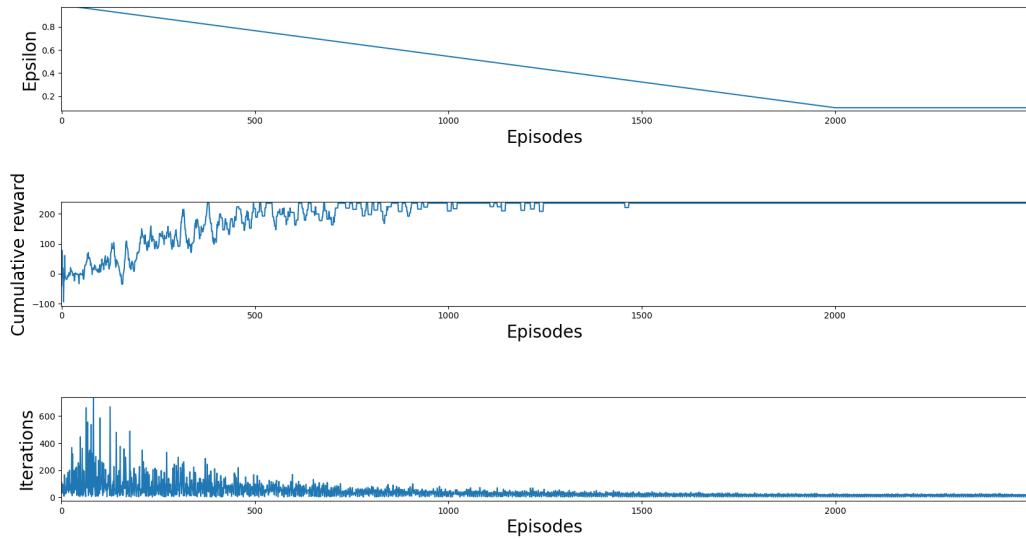


Figura 4.13: Gráfico de entrenamiento

### Fase de inferencia

Por último, en la *fase de inferencia*, el dron analiza su estado (o sus coordenadas dentro del mapa de calor), y observa la mejor acción disponible dentro de la tabla Q ya rellena. Esto lo realiza hasta que detecta la condición de parada, que se cumple cuando la medida anterior de señal es mayor que la actual y todos los vecinos adyacentes a la mayor de las medidas, poseen señal inferior. Para hacer un correcto análisis, se parte siempre de coordenadas distintas a las que se usaron para entrenar y llenar la tabla Q.

#### 4.3.4. Experimentos y resultados

##### Experimento 1 - Comparativa de algoritmos para escenarios de distintas dimensiones

Durante todos los experimentos realizados, los parámetros de la señal siempre son los valores por defecto representados en la tabla 4.3, donde sólo varía el tamaño del mapa, que se va modificando según el experimento. Además, cabe destacar que la señal se toma como un punto estático dentro del mapa de calor y con respecto al dron, distinguiendo la posición centrada y la posición cercana a una esquina, que también dependen del experimento. Además, para todos los entrenamientos realizados, los parámetros usados son los descritos en la tabla 4.2.

Primero se ha probado sobre un escenario de tamaño 12x12 metros (figura 4.14).

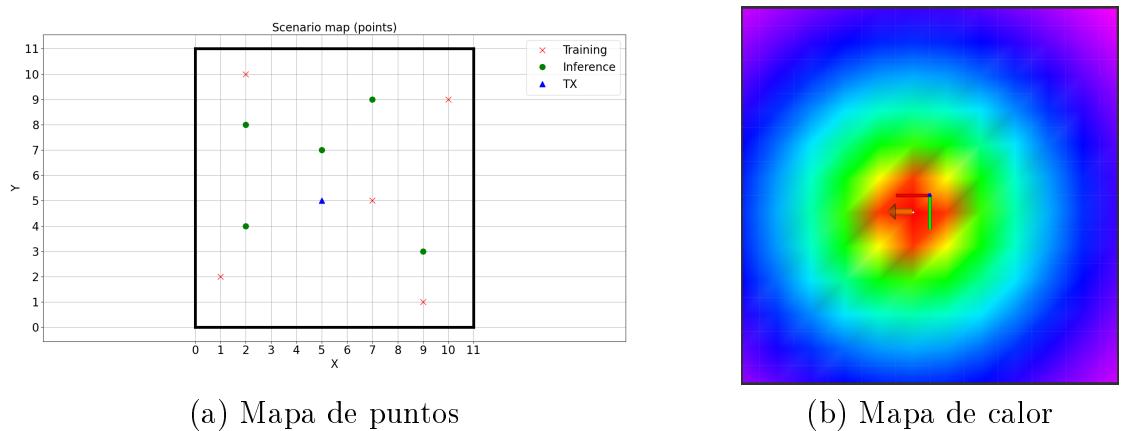


Figura 4.14: Transmisor centrado (12x12)

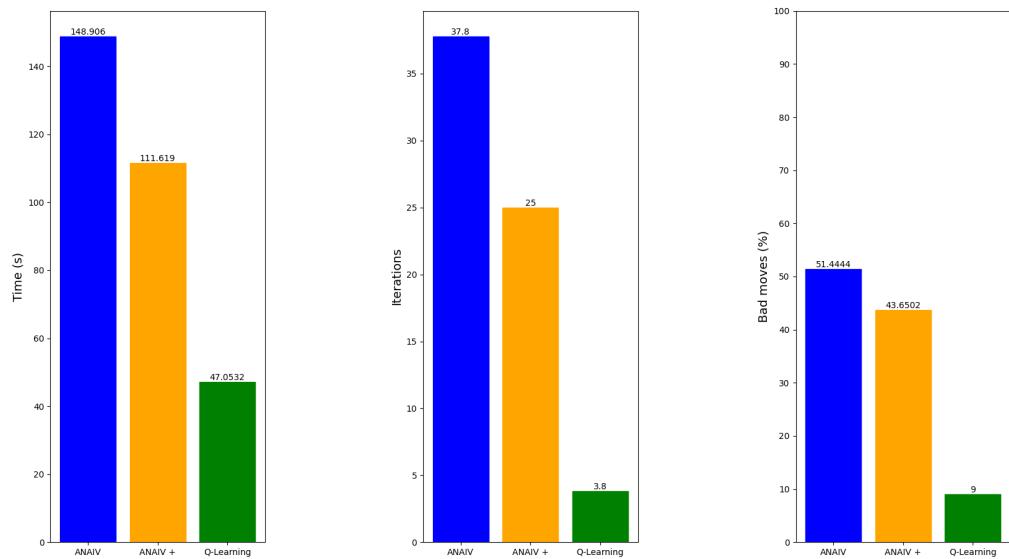


Figura 4.15: Comparativas (12x12), transmisor centrado

Característica	Valor
Potencia del transmisor	1 W
Ganancia del transmisor	1 W
Ganancia del receptor	1 W
Frecuencia	2.4 GHz (WiFi)
Factor de pérdidas (L)	2.4
Exponente n	2
Resolución	1

Tabla 4.3: Características de la señal por defecto

Para el transmisor cerca del centro, situada en las coordenadas (5, 5) del “*heatmap*”, extraemos un mapa de puntos donde se muestran las posiciones de entrenamiento o puntos donde el dron despegó durante su aprendizaje (representados con una “X” roja); inferencia o puntos donde el dron despegó para probar lo que ha aprendido (representados por círculos verdes); y del origen del transmisor de la señal (representado por un triángulo azul), así como de los límites del escenario para el mapa de calor (que se distinguen con líneas gruesas).

Los resultados obtenidos, son representados en una serie de gráficas comparativas. En ellas observamos el rendimiento de cada algoritmo (ANAVI, ANAVI + o mejorado, y Q-Learning), es decir, se analizan tres métricas: el tiempo medio en segundos que tarda el dron desde que despegó hasta que vuelve a su posición de despegue; el número medio de iteraciones empleadas para alcanzar el transmisor; y el número medio de movimientos hacia coordenadas donde la potencia de señal es menor y no mayor, por lo que son considerados como “*bad moves*”<sup>5</sup>. Para este caso, los resultados obtenidos en la figura 4.15 arrojan que el algoritmo más eficiente es el de Q-Learning, ya que tarda menos tiempo, realiza menos iteraciones hasta llegar a la meta y tiene un porcentaje inferior de malas acciones. Esto es esperable, debido a que el algoritmo basado en RL ha realizado un entrenamiento sobre el que aprende como se propaga la señal, lo que aumenta la exactitud y eficiencia del algoritmo.

En el caso del transmisor cerca de la esquina, se sitúa en las coordenadas (3, 1) con respecto a las coordenadas del “*heatmap*”, siendo su mapa de puntos el observado en la figura 4.16. En este caso, se obtiene la misma conclusión que en el escenario anterior, tal y como se aprecia en la figura 4.17. Nuevamente, para el escenario de tamaño 30x30, los resultados obtenidos son idénticos, donde, con la salvedad de que el tiempo empleado

---

<sup>5</sup>Los datos arrojados han sido guardados en formato *csv*.

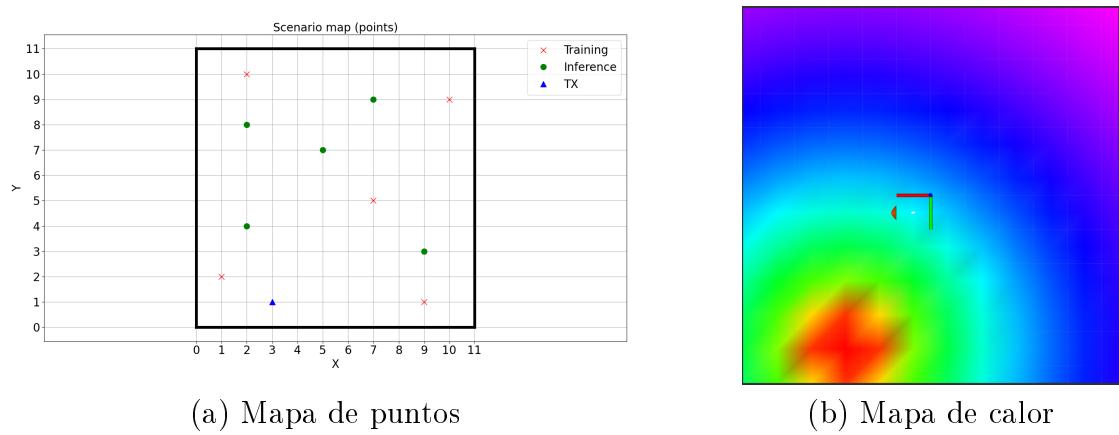


Figura 4.16: Transmisor en la esquina (12x12)

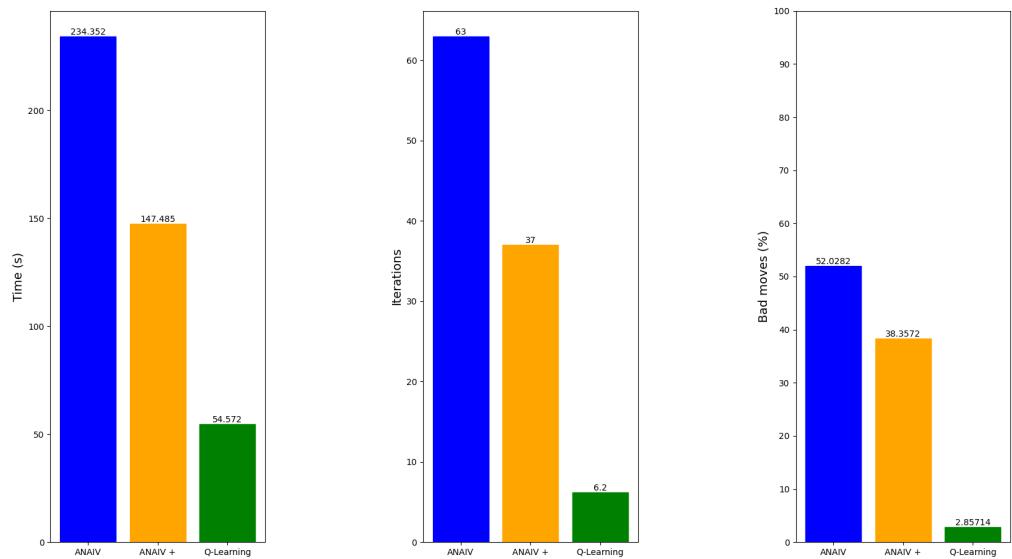
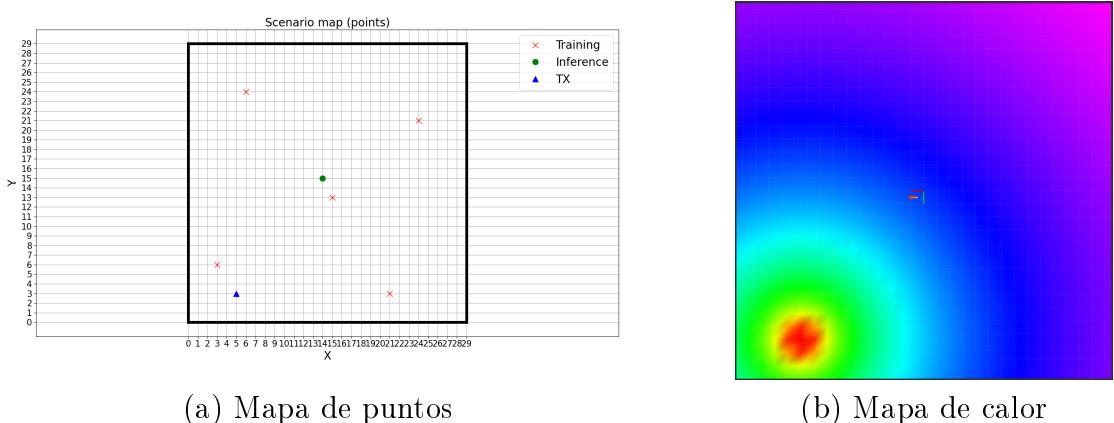


Figura 4.17: Comparativas (12x12), transmisor en la esquina



(a) Mapa de puntos

(b) Mapa de calor

Figura 4.18: Transmisor en la esquina (30x30)

es mayor y el número de iteraciones también, se concluye que el algoritmo de RL es el mejor en todos los casos.

### Experimento 2 - Desempeño del algoritmo Q-Learning para dos señales distintas

En este caso, se ha realizado un experimento en el cual se entrena al modelo de Q-Learning con los parámetros especificados en la tabla 4.2, donde el transmisor emite con las características de señal mencionadas en la tabla 4.3 (en gráficas comparativas hace referencia a “*Q-Learning*”). A continuación se realiza inferencia, y posteriormente se modifican las características de la señal, aumentando la potencia del transmisor al doble (2 W) y cambiando la frecuencia (de 2.4 GHz a 5 GHz), de modo que en las gráficas comparativas se nombra como “*Q-Learning 2*”. Nuevamente se realiza inferencia y se extraen resultados. Las coordenadas en las que se sitúa el transmisor están en (5, 3) y no son modificadas en ningún momento. Además, los puntos de entrenamiento seleccionados se pueden ver en el mapa de puntos de la figura 4.18.

Para este caso, el problema se resuelve de igual forma para ambos casos, con una cierta variación irrelevante en la métrica temporal, derivada probablemente de la propia simulación, lo que se puede concluir como que el modelo aprende como se propaga la señal, independientemente de las características de la misma. Como se puede observar en la figura 4.19, se muestran las gráficas comparativas y un gráfico de trayectorias (figura 4.20), que representa el camino seguido por el dron al aplicar la inferencia en ambos casos.

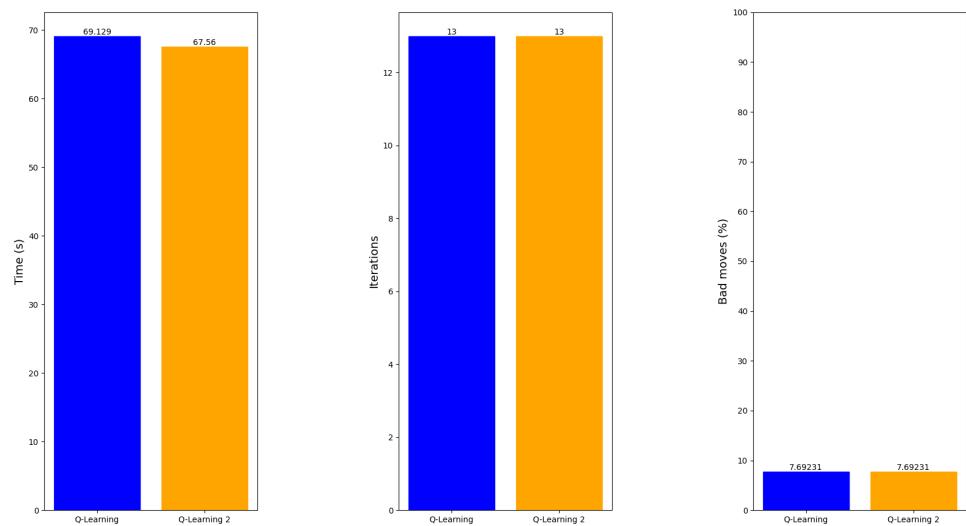


Figura 4.19: Comparativas (30x30), escenario planteado

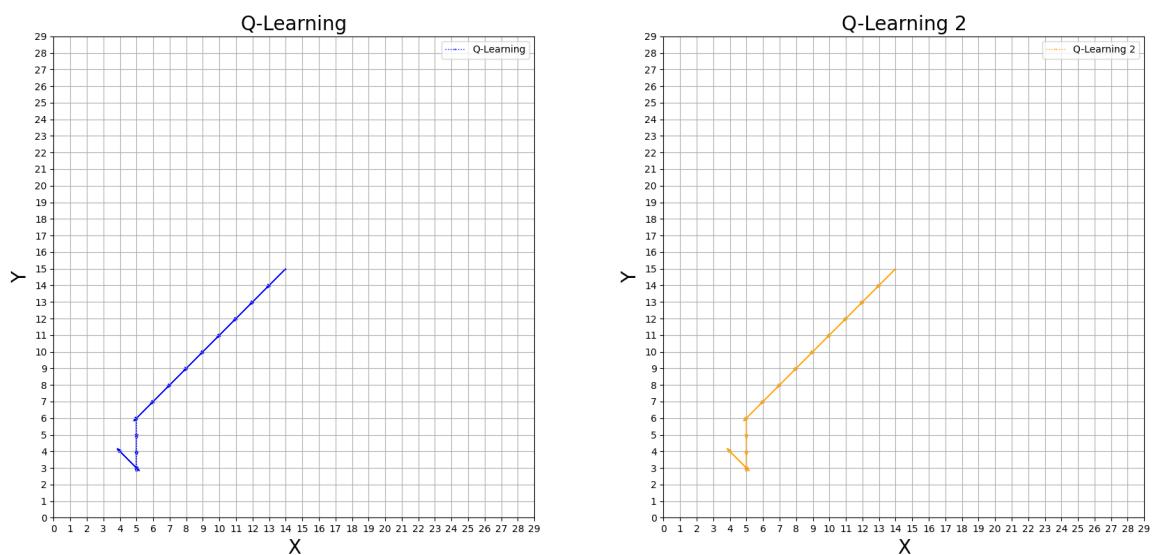
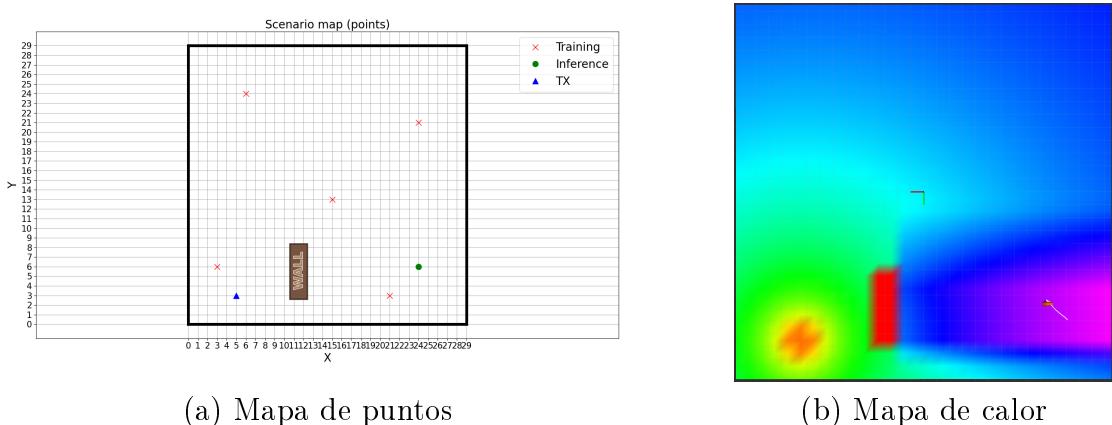


Figura 4.20: Trayectorias seguidas durante la inferencia



(a) Mapa de puntos

(b) Mapa de calor

Figura 4.21: Escenario con obstáculos (30x30)

## 4.4. Comportamiento sigue señal basado en RF en un entorno dinámico

### 4.4.1. Introducción al problema

El escenario planteado anteriormente, es una aproximación poco realista, es decir, en un entorno real se esperan perturbaciones y obstáculos. Por ello, se ha implementado un escenario con muros que distorsionan la señal.

Inicialmente, se ha simulado el degradado de la señal en presencia de obstáculos (ver figura ??), es decir, se ha añadido un muro al mapa original, lo que en consecuencia ha derivado en una degradación de la señal, por lo que se han sobreescrito los valores originales de la señal por valores afectados con un factor de pérdidas (en este caso 1.015 para valores negativos).

### 4.4.2. Algoritmos

En cuanto a los algoritmos empleados, se parte de la idea de realizar un entrenamiento normal sin obstáculos, y ajustar el comportamiento a las situaciones emergentes.

Por ello se distinguen dos casos:

1. *El dron vuela por encima de la altura del obstáculo*: es decir, el dron navega por una zona con interferencias pero a una altura donde no exista colisión.

2. *El dron vuela a la misma altura que el obstáculo:* donde según que camino se escoja puede existir colisión.

#### 4.4.3. Experimentos y resultados

Por ello, se observa que para el primer caso donde el dron sobrevuela cualquier muro, no existe inconveniente, puesto que el dron toma el camino que tomaría si no hubiera obstáculos. Esto es debido a que en inferencia, tan solo se tiene en cuenta su posición con respecto al mapa de calor, y por tanto no identifica si hay o no obstáculos, lo que a efectos prácticos, es cómo si no existiera para el dron. Debido a esto, no se muestran resultados, ya que el comportamiento es idéntico a los casos estudiados en las secciones anteriores. Sin embargo, sí es relevante a la hora de abordar la condición de finalización, ya que al modificar el mapa de señal, se pueden dar falsos positivos y generar mínimos locales. Por ello, se ha empleado una técnica que analiza, por pares, las coordenadas visitadas, de modo que se tiene la coordenada actual y la nueva, y por iteración se compara con la siguiente actual y la siguiente nueva, y si coinciden, se cumple la condición pasando al análisis de los vecinos cercanos, como en todos los casos anteriores. Por último cabe destacar que esta condición funciona para desplazamientos de 1 m, o dicho de otro modo, de celdilla en celdilla.

En el otro escenario, donde el dron vuela a la altura de los obstáculos, se debe implementar un método de detección de obstáculos. En general, se puede generar un caso realista usando un láser LiDAR o una cámara 3D, pero para simplificar el problema, se ha empleado la información del mapa realizando una simulación del comportamiento que tendría un sensor genérico de este tipo. Por ello, primero se prueba excluyendo las acciones que desemboquen en colisión, donde se presupone que el entrenamiento evitará acciones que alejen al dron de la señal, tal y como se observa en la figura 4.22.

Sin embargo y siguiendo el esquema de la figura 4.22, los resultados no son concluyentes, de modo que en ocasiones sorteá el obstáculo, pero en otras simplemente retrocede o alcanza un mínimo local. De este modo, y volviendo a la conclusión extraída del primer escenario, se ve la necesidad de aplicar un comportamiento híbrido, de modo que se navegue hacia la señal, hasta topar con un obstáculo lo suficientemente cercano, como para que cambie el modo de actuación para evitarlo. Una vez sorteado, se vuelve modo de navegación normal, hasta llegar a la señal o a otro obstáculo (ver figura 4.23).

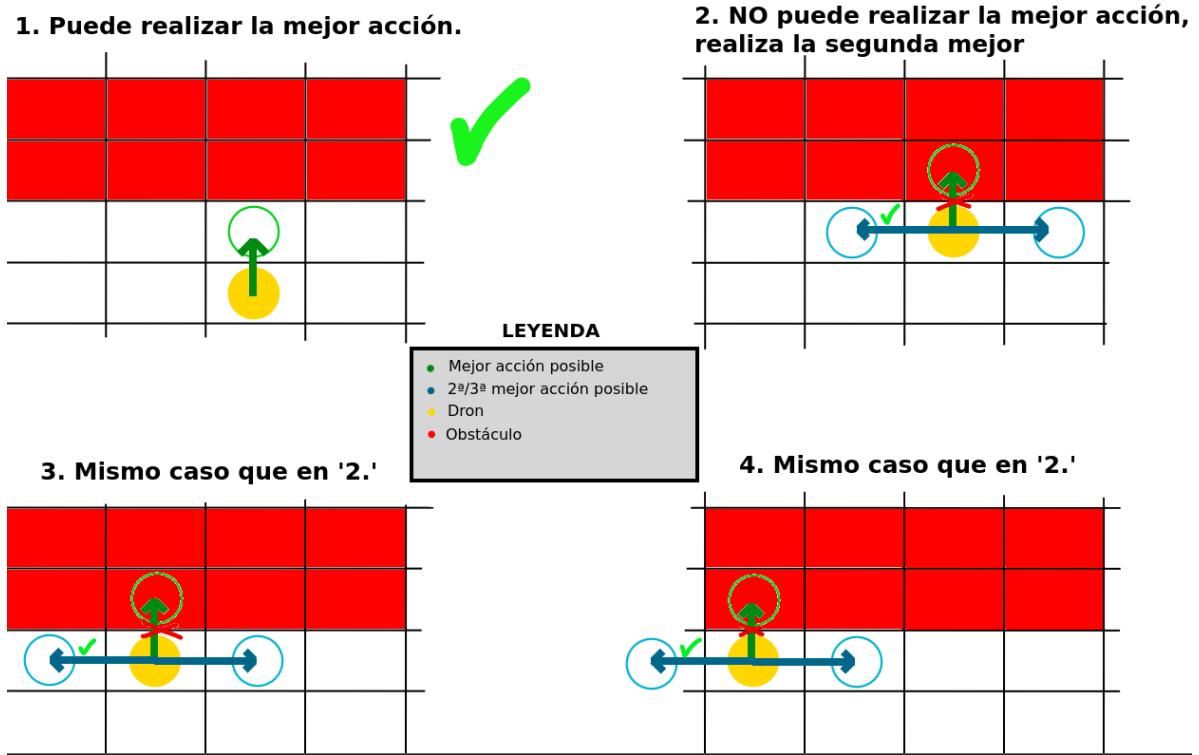


Figura 4.22: Simulación de sensor para obstáculo

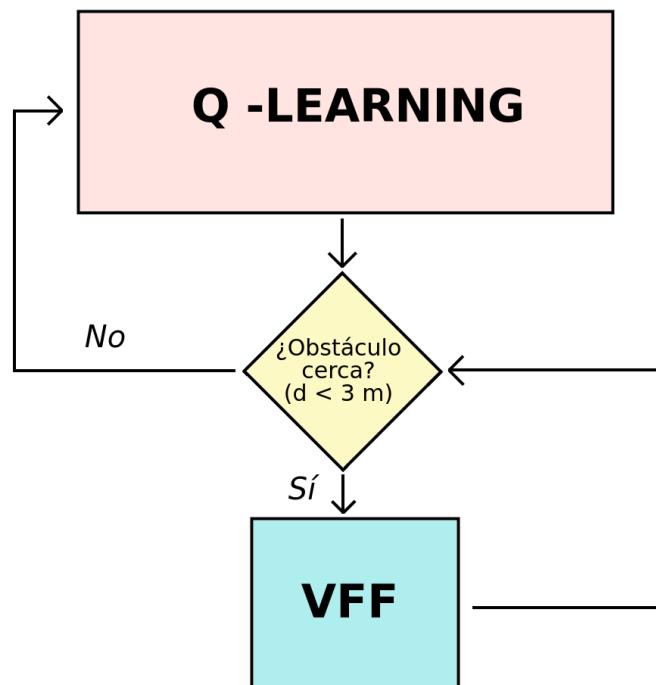


Figura 4.23: Diagrama de funcionamiento híbrido

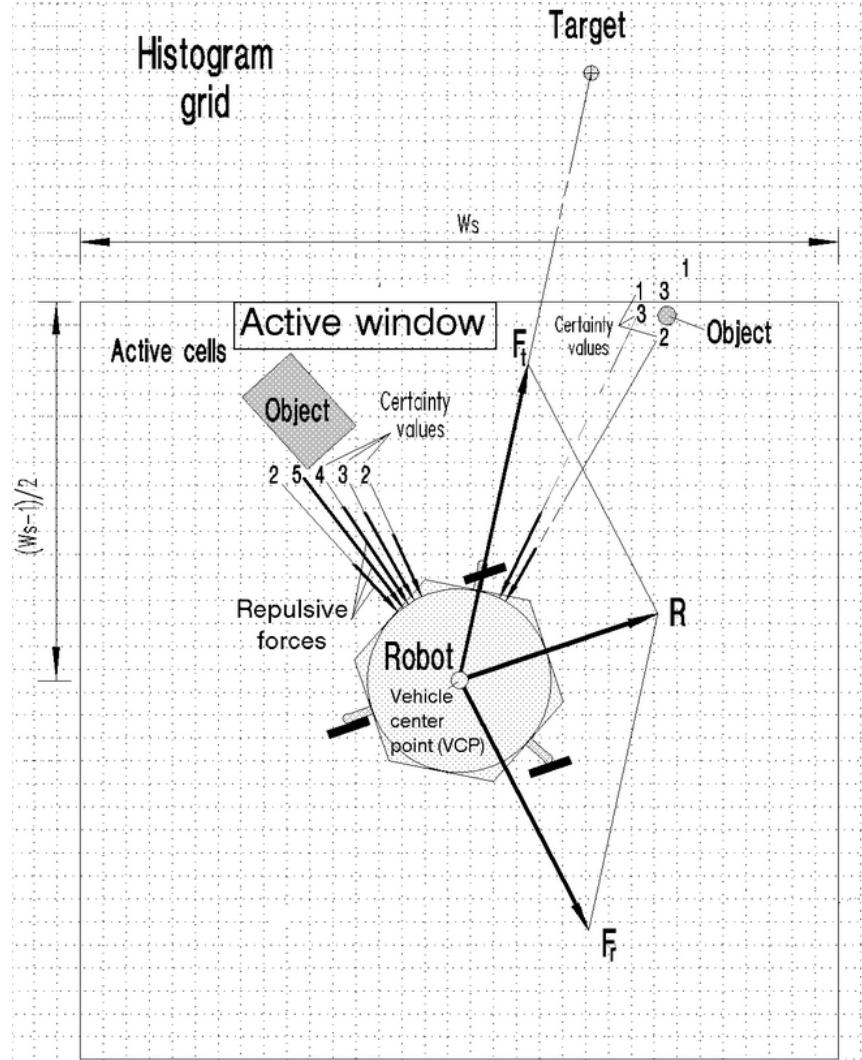


Figura 4.24: Funcionamiento general del algoritmo VFF

Por ello, se opta por emplear un algoritmo basado en VFF (ver figura 4.24), que consiste en generar un comportamiento basado en vectores de fuerzas virtuales, de modo que se alcancen subobjetivos cercanos, evitando colisiones con obstáculos (ver figura). Para nuestro caso concreto, se ha decidido adaptar de modo que: los obstáculos son dispuestos en el mapa por celdillas ocupadas, de modo que cada celdilla genera un vector inversamente proporcional a la distancia, desde la celdilla a la posición del dron, generando un vector de repulsión; los subobjetivos que generaría el vector de fuerza atractiva, son sustituidos por un vector basado en la tendencia seguida por el dron, durante la navegación basada en Q-Learning, es decir, se obtiene un vector unitario resultante de todas las posiciones visitadas por el dron, previamente. Todo esto, se ajusta siguiendo la ecuación 4.5, que traducido a comandos para el dron son velocidades en x, y, ya que es lo que mejor se ajusta al comportamiento reactivo buscado.

$$F_{res} = \alpha \cdot F_{atr} + \beta \cdot F_{rep} \quad (4.5)$$

Donde tras el ajuste por experimentación,  $\alpha$  (que determina la importancia que tendrá la fuerza atractiva en la ejecución) toma el valor de 2.5, y  $\beta$  (que por contra determina la importancia de la fuerza repulsiva) toma el valor de 0.05. Adicionalmente, es necesario comentar un problema derivado del diseño a la hora de simular el escenario. A causa de que el transmisor se sitúa esquinado y el obstáculo también, y dado que el dron solo actuaba sobre el muro, si el algoritmo VFF desplazaba al dron fuera de las coordenadas del mapa de calor, se generaban comportamientos impredecibles (lo cual en la realidad no sucedería, ya que la señal se propagaría en todas direcciones). Para solucionarlo, se agregaron muros virtuales que actúan como obstáculos, con el fin de delimitar estas áreas fuera del mapa de calor. El resultado final es satisfactorio, ya que se consigue navegar hacia la señal, evitando obstáculos. En la figura 4.25, se puede ver una representación gráfica del escenario y su funcionamiento, donde se dispone un muro, el transmisor y el algoritmo funcionando en ambos modos.

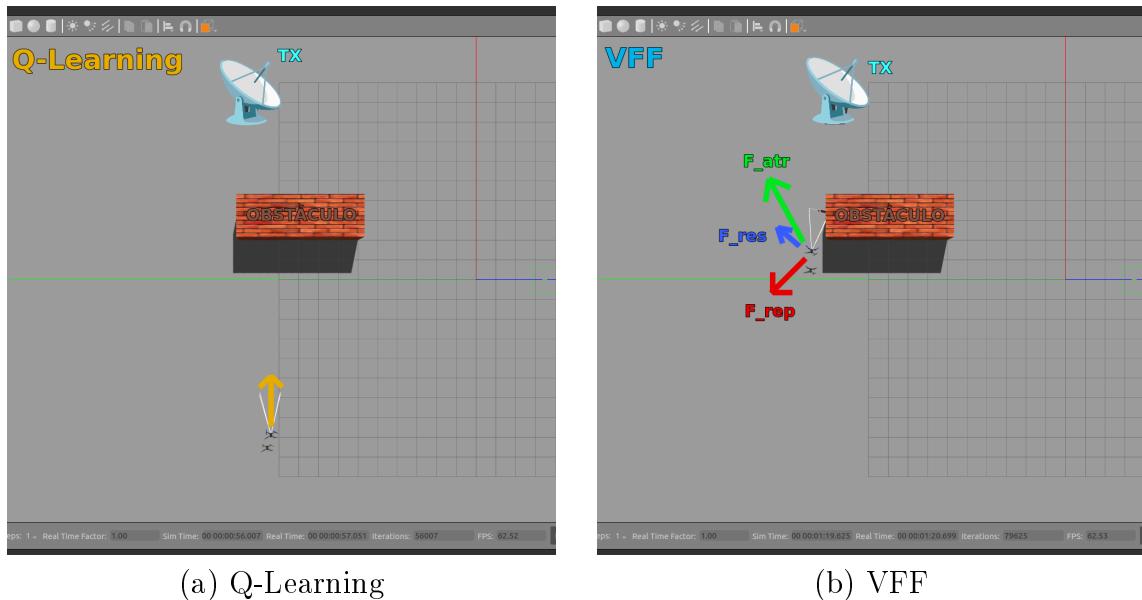


Figura 4.25: Enfoque híbrido

---

# Capítulo 5

## Conclusiones

---

En esta última sección, se comentarán las ideas extraídas tras el desarrollo del proyecto, así como el conocimiento obtenido y la posible continuación del mismo.

### 5.1. Objetivos cumplidos

Durante el desarrollo de este TFG, se ha conseguido satisfactoriamente desarrollar una solución de navegación autónoma para comandar a un dron hacia un transmisor de RF, demostrando que la aproximación realizada con RL cumple con lo esperado. En detalle se puede afirmar que:

1. Se ha desarrollado una interfaz capaz de interactuar de manera reactiva con el dron, comandando tanto posiciones como velocidades.
2. Se ha creado un nodo ROS disponible para la comunidad<sup>1</sup>, el cual gestiona un modelo de propagación de señales, a demanda de las aplicaciones que lo usen.
3. Se ha conseguido implementar diversos algoritmos capaces de navegar hacia transmisores RF de manera autónoma.
4. Se ha demostrado que el algoritmo basado en Q-Learning es el más eficiente en términos de tiempo, número de movimientos, y movimientos malos (hacia potencias de señal inferiores), de todos los planteados.
5. Se ha adaptado la mejor solución a un entorno realista donde se presentan obstáculos, a través de un enfoque híbrido empleando VFF.

### 5.2. Balance global y competencias adquiridas

En cuanto a los conocimientos adquiridos, podemos distinguir:

---

<sup>1</sup>[https://github.com/RoboticsLabURJC/2022-tfg-cristian-sanchez/blob/main/src/heatmap\\_util/scripts/rf\\_data\\_server.py](https://github.com/RoboticsLabURJC/2022-tfg-cristian-sanchez/blob/main/src/heatmap_util/scripts/rf_data_server.py)

1. Desarrollo de aplicaciones usando OpenCV y Matplotlib.
2. Uso de plugins y trabajo con el modelo SDF del Iris Drone.
3. Creación de entornos personalizados empleando Gazebo 11.
4. Uso de marcadores y del módulo grid\_map para rviz, así como su aplicación conjunta a través ROS en C++ y Python.
5. Empleo de PX4 y MAVROS para el control de la aeronave.
6. Desarrollo de soluciones basadas en Q-Learning a través de Python.
7. Adquisición de conocimientos relacionados al estudio de señales y su comportamiento.

### 5.3. Líneas futuras

Finalmente, y tras demostrar que el mejor algoritmo según los resultados obtenidos alude a la solución por Q-Learning, cabe preguntarse si se podrían encontrar variantes relacionadas (siguiendo la línea de RL) que arrojasen mejores resultados, es decir, ver si rinden mejor que las soluciones planteadas, vease *Proximal Policy Optimization* (PPO), *Deep Deterministic Policy Gradient* (DDPG), *Soft Actor Critic* (SAC), entre otros. Todo ello, con el fin último de transladarlo a un dispositivo real y verificar su funcionamiento.

# Anexo

---

A continuación muestro una serie de conceptos básicos relacionados con el estudio del procesamiento de señales<sup>1</sup>:

1. *Señal*: se trata de una función que describe un fenómeno físico, y que se emplea para la transmisión de información.
2. *Dominio temporal*: establece el eje de abcisas con el tiempo.
3. *Dominio de la señal*: determina si la señal se expresa en tiempo o en frecuencia (a través de transformadas).
4. *Analog to Digital Converter* (ADC): elemento electrónico que permite la conversión de señales analógicas a señales digitales.
5. *Received Signal Strength Indicator* (RSSI): permite establecer el nivel de potencia de una señal, con respecto a 1 mW de potencia. Se expresa en dBm.
6. *Signal to Noise Ratio* (SNR): métrica que permite medir la potencia de la señal con respecto al ruido ambiente.
7. *Frecuencia*: parámetro de la función que define a la señal, el cual determina el número de veces que se repite en un segundo. Se mide en hercios (Hz).
8. *TX*: Se refiere a la transmisión de la señal.
9. *RX*: Hace referencia a la recepción de la señal.

---

<sup>1</sup>Información extraída de la siguiente colección de videos didácticos [https://www.youtube.com/playlist?list=PL8bSwVy8\\_IcPCsBE71CYBLbQSS8ckWm6x](https://www.youtube.com/playlist?list=PL8bSwVy8_IcPCsBE71CYBLbQSS8ckWm6x)

# Bibliografía

---

- [1] Daniel Dworakowski and Goldie Nejat. Robots understanding contextual information in human-centered environments using weakly supervised mask data distillation, 2020.
- [2] Jiefei Wang and Damith Herath. *What Makes Robots? Sensors, Actuators, and Algorithms*, pages 177–203. Springer Nature Singapore, Singapore, 2022.
- [3] Andrea Maiorino and Giovanni Gerardo Muscolo. Biped robots with compliant joints for walking and running performance growing. *Frontiers in Mechanical Engineering*, 6, 2020.
- [4] R. K. Nichols; H.C. Mumm; W.D. Lonstein; C. M. Carter; and J.P. Hood. *Chapter 13: Data Links Functions, Attributes and Latency*. New Prairie Press, 2018.
- [5] Anaís Garrell, Carles Coll, René Alquézar, and Alberto Sanfeliu. Teaching a drone to accompany a person from demonstrations using non-linear asfm. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1985–1991, 2019.
- [6] Max Christl. Vision-based autonomous drone control using supervised learning in simulation, 2020.
- [7] Y. Wang, R. Yoshihashi, R. Kawakami, et al. Unsupervised anomaly detection with compact deep features for wind turbine blade images taken by a drone. *IPSJ Transactions on Computer Vision and Applications*, 11(3):1–10, 2019.
- [8] Ahmad Taher Azar, Anis Koubaa, Nada Ali Mohamed, Habiba A. Ibrahim, Zahra Fathy Ibrahim, Muhammad Kazim, Adel Ammar, Bilel Benjdira, Alaa M. Khamis, Ibrahim A. Hameed, and Gabriella Casalino. Drone deep reinforcement learning: A review. *Electronics*, 10(9), 2021.
- [9] Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. Autonomous drone racing with deep reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8, 2021.

- International Conference on Intelligent Robots and Systems (IROS)*, pages 1205–1212, 2021.
- [10] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667, 2019.
  - [11] Xinwei Chen, Marlin W. Ulmer, and Barrett W. Thomas. Deep q-learning for same-day delivery with vehicles and drones. *European Journal of Operational Research*, 298(3):939–952, 2022.
  - [12] Meisam Kabiri, Claudio Cimarelli, Hriday Bavle, Jose Luis Sanchez-Lopez, and Holger Voos. A review of radio frequency based localisation for aerial and ground robots with 5g future perspectives. *Sensors*, 23(1), 2023.
  - [13] R.C. Johnson and H. Jasik. *Antenna Engineering Handbook*. Number v. 2 in McGraw-Hill handbook. McGraw-Hill, 1984.
  - [14] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.
  - [15] Md Moin Uddin Chowdhury, Fatih Erden, and Ismail Guvenc. Rss-based q-learning for indoor uav navigation, 2019.