



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



TFG del Grado en Ingeniería Informática  
Sistema de Navegación Semiautónomo en Interiores



Presentado por Mario Bartolomé Manovel  
en Universidad de Burgos — 3 de junio de 2018

### Tutores

Dr. Alejandro Merino Gómez  
Dr. César Ignacio García Osorio  
Dr. José Francisco Díez Pastor





UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



D. Dr. Alejandro Merino Gómez, profesor del Departamento de Ingeniería Electromecánica, Área de Ingeniería de Sistemas y Automática. D. Dr. César Ignacio García Osorio y D. Dr. José Francisco Díez Pastor, profesores del Departamento de Ingeniería Civil, Área de Lenguajes y Sistemas Informáticos.

**Exponen:**

Que el alumno D. Mario Bartolomé Manovel, con DNI 71298657Z, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado «Sistema de Navegación Semiautónomo en Interiores».

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección de los que suscriben, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 3 de junio de 2018

Vº. Bº. del Tutor:

D. Dr. Alejandro  
Merino Gómez

Vº. Bº. del Tutor:

D. Dr. César Ignacio  
García Osorio

Vº. Bº. del Tutor:

D. Dr. José Francisco  
Díez Pastor



## Resumen

El objetivo del proyecto es diseñar un sistema de navegación semi-autónomo en espacios cerrados, destinado a la asistencia en vigilancia de seguridad mediante *drones*. Dada una estancia, el *drone* deberá ser capaz de realizar un recorrido por el interior, grabando vídeo que será emitido a un servidor, y transmitiendo su posición dentro del mapa a un responsable de seguridad. El *drone* estará equipado con una RaspberryPi que hará las veces de sistema de control y de transmisor de vídeo. Para llevar a cabo la evasión de obstáculos se han implementado dos soluciones, una basada en campos potenciales, y otra, conocida como *Vector Field Histogram*, que complementa a esta primera, solucionando algunos de los problemas que esta presenta.

La localización en interiores no es fiable mediante GPS, por ello se hará uso de un sistema probabilístico basado en simulación de instancias. Dicho sistema, conocido como filtro de partículas, está basado en el método de Montecarlo como estimador de una variable en base a observaciones del entorno.

Para lograr la comunicación entre el *drone* y la RaspberryPi, se ha creado una implementación del protocolo *MultiWii Serial Protocol*, el cual permite tanto el envío de comandos, como la solicitud de información a la controladora de vuelo.

## Descriptores

*drone*, vehículo aéreo no tripulado, semi-autónomo, semi-automático, vigilancia, navegación, interior, vídeo, filtro de partículas, campos potenciales, Histograma de Campos Vectoriales, VFH, búsqueda de ruta, MSP, RaspberryPi

## Abstract

The aim of this project is to design a semi-autonomous navigation system in enclosed spaces, destined to aid security vigilance using drones. Given an enclosed space, the *drone* should be able to make its path through it, recording video which will be streamed to a server, and updating its position inside of the enclosed space to the security guard in charge. The drone will be guided making use of a RaspberryPi, which will be controlling the drone and streaming the video. To achieve the obstacle avoidance, two different solutions have been implemented, the first based on potential fields, and another one, known as *Vector Field Histogram*, to complement the first one, solving some of its caveats.

Enclosed space location through GPS is not reliable enough, therefore a probabilistic method, based on instance simulation will be used. Also known as *Particle Filter*, this method is based on Montecarlo simulations as an estimator of an unknown variable taking into account multiple observations of the environment.

To achieve communication between the drone and the RaspberryPi, an implementation of the MultiWii Serial Protocol was developed, which allows to send commands and retrieve information from the flight controller.

## Keywords

drone, UAV, semi-autonomous, semi-automatic, vigilance, navigation, indoor, video, particle filter, potential fields, Vector Field Histogram, VFH, path searching, MSP, RaspberryPi

---

# Índice general

---

<b>Índice general</b>	<b>III</b>
<b>Índice de figuras</b>	<b>v</b>
<b>Índice de tablas</b>	<b>vii</b>
<b>Introducción</b>	<b>1</b>
<b>Objetivos del proyecto</b>	<b>3</b>
2.1. Objetivos marcados por Requisitos Funcionales . . . . .	3
2.2. Objetivos técnicos derivados . . . . .	3
2.3. Objetivos personales . . . . .	4
<b>Conceptos teóricos</b>	<b>6</b>
3.1. Algoritmia . . . . .	6
<i>Particle Filter localization</i> . . . . .	6
Campos Potenciales . . . . .	11
Vector Field Histogram . . . . .	16
3.2. Dispositivos Físicos . . . . .	26
Raspberry Pi . . . . .	26
Controladora de Vuelo . . . . .	28
Sensores de distancia . . . . .	30
3.3. Protocolos . . . . .	32
Pulse Width Modulation . . . . .	32
MultiWii Serial Protocol . . . . .	33
Secure SHell . . . . .	35
WebSocket . . . . .	37
WebRTC . . . . .	38
<b>Técnicas y herramientas</b>	<b>41</b>

4.1.	Metodologías de Desarrollo . . . . .	41
SCRUM . . . . .	41	
GitFlow . . . . .	41	
Kanban . . . . .	42	
4.2.	Herramientas de gestión de repositorio . . . . .	42
Git . . . . .	42	
GitHub . . . . .	42	
ZenHub . . . . .	43	
GitKraken . . . . .	43	
4.3.	Herramientas de desarrollo . . . . .	44
PyCharm . . . . .	44	
WebStorm . . . . .	44	
4.4.	Herramientas de documentación . . . . .	44
L <sup>A</sup> T <sub>E</sub> X . . . . .	44	
<b>Aspectos relevantes del desarrollo del proyecto</b>	<b>46</b>	
5.1.	Inicio del proyecto . . . . .	46
5.2.	Metodología . . . . .	47
5.3.	Formación . . . . .	48
5.4.	Desarrollo de algoritmos . . . . .	49
Potential Fields . . . . .	50	
Vector Field Histogram . . . . .	50	
Particle Filter . . . . .	52	
PID . . . . .	53	
5.5.	Comunicaciones . . . . .	54
5.6.	Plataforma de usuario . . . . .	57
5.7.	Hardware empleado . . . . .	59
5.8.	Pruebas . . . . .	64
Pruebas de Campo . . . . .	66	
5.9.	Agradecimientos . . . . .	72
<b>Trabajos relacionados</b>	<b>73</b>	
6.1.	Empresas . . . . .	73
TAISEI Co. LTD . . . . .	73	
Erle-Robotics . . . . .	73	
FlyPulse . . . . .	73	
TUDelft . . . . .	74	
6.2.	Militar . . . . .	74
<b>Conclusiones y Líneas de trabajo futuras</b>	<b>75</b>	
7.1.	Conclusiones . . . . .	75
7.2.	Líneas de trabajo futuras . . . . .	76
<b>Bibliografía</b>	<b>78</b>	

---

# Índice de figuras

---

3.1. Gradiente de Potencial de Atracción . . . . .	12
3.2. Gradiente de Potencial de Repulsión . . . . .	13
3.3. Mínimo local en zonas convexas . . . . .	14
3.4. Movimiento oscilatorio en corredores estrechos . . . . .	15
3.5. Distribución Histogramática de Probabilidades . . . . .	18
3.6. Angulos relativos al VCP del <i>drone</i> . . . . .	19
3.7. POD vs smoothPOD . . . . .	21
3.8. Valles estrechos y anchos . . . . .	22
3.9. Movimiento circular del agente . . . . .	23
3.10. Valle estrecho . . . . .	24
3.11. Umbral elevado . . . . .	25
3.12. Umbral bajo . . . . .	25
3.13. Raspberry Pi 3 . . . . .	27
3.14. Benchmark de lector microSD OC . . . . .	28
3.15. Controladora de Vuelo Flip32 . . . . .	29
3.16. Sensor HC-SR04 . . . . .	31
3.17. PWM. Modulación en Ancho de Pulso . . . . .	33
3.18. Composición de un mensaje MSP . . . . .	34
5.19. Logo . . . . .	47
5.20. Web index . . . . .	56
5.21. Página de Login . . . . .	58
5.22. Página de control . . . . .	58
5.23. Página de control y emisora . . . . .	59
5.24. Conceptual de conexión de sensores a RaspberryPi . . . . .	60
5.25. Diagrama de conexión de sensores a RaspberryPi . . . . .	61
5.26. Divisor de voltaje . . . . .	62
5.27. LED IR . . . . .	62
5.28. Lateral del <i>drone</i> . . . . .	63
5.29. Frontal del <i>drone</i> . . . . .	63
5.30. Recepción de la IMU mediante MSPio . . . . .	65

*Índice de figuras*

VI

5.31. Field Test 1. Villalonquejar . . . . .	66
5.32. Field Test 2. Nave industrial . . . . .	68
5.33. Field Test 2. Demasiada kP . . . . .	69
5.34. Field Test 2. Prótesis . . . . .	69
5.35. Field Test 2. Altura alcanzada con suavidad . . . . .	70

---

## Índice de tablas

---

3.1.	Componentes de una Raspberry Pi 3 Model B . . . . .	26
3.2.	Nomenclatura del módulo Struct de la implementación 3.5 de Python	35
3.3.	Mensajes MSP. Estructura de datos y representación . . . . .	36

---

# Introducción

---

Con el creciente uso de sistemas automatizados que ha presentado la industria en los últimos años, el nicho que los *drones* han pasado a crecer de forma rápida haciendo que se descarten métodos muy consolidados hasta el momento. Vigilancia, industria cinematográfica, ocio, deportes y mensajería son solo algunos de los ejemplos en los que los *drones* se van abriendo camino.

La existencia de *drones* cada vez más automatizados ha llegado a un punto en el que puede ser sencillo para un principiante realizar tomas de vídeo, que podrían catalogarse en el ámbito profesional, en una zona de difícil pilotaje.

Y este es precisamente el reto: lograr que un dispositivo tan complejo como un *drone*, que puede causar daños serios, sea sencillo de operar hasta por un principiante.

Dada esta vertiente y los claros beneficios que un sistema así puede tener, la industria incluye cada vez más sistemas que tratan de automatizar mecanismos de seguridad, como la evasión de obstáculos o la geolocalización, sistemas de control de vuelo, como el aterrizaje y despegue, o funciones de seguimiento de objetivos, reconocimiento de imagen, planeamiento y seguimiento de rutas, etc.

La industria parece moverse hacia sectores dedicados actividades en exterior, donde propuestas como la evasión de obstáculos, pueden ser algo más triviales o innecesarias:

- Sistemas de vigilancia de incendios, en sustitución de helicópteros mucho más caros de contratar.
- Sistemas de vigilancia de campos y cultivos.
- Sistemas de fumigación agrícola, en sustitución de avionetas mucho más caras de mantener.
- Sistemas de inspección de zonas de difícil acceso, en sustitución de métodos tradicionales como andamios o rápel.

- Sistemas de grabación a nivel profesional, en sustitución de helicópteros mucho más caros de contratar.

Sin embargo, parece estar obviándose el uso que se puede dar a un *drone* en un espacio cerrado, ya sea para acceder a zonas complejas en edificios o estructuras con gran cantidad de obstáculos, movimiento de mercancías ligeras en ciudades, en el ámbito militar, o vigilancia en general.

Incluir sistemas que provean de autonomía a un *drone* puede ser de tremendo beneficio para el desarrollo de un sector cada vez más en auge. Disponer de los mismos sistemas, sin una necesidad constante de un piloto cualificado, permitiría reducir los costes de esta tecnología, así como incrementar la competitividad.

Este proyecto propone una implementación y la creación de un *drone* provisto de un sistema capaz de recorrer un espacio cerrado basándose en un plano preexistente. Evitará colisionar con elementos del entorno que detectará mediante ultrasonidos, y proporcionará una fuente de vídeo en tiempo real, su localización dentro del mapa, y dará la posibilidad de ser controlado de forma totalmente remota.

En este documento se encuentra toda la información relacionada con el Trabajo de Fin de Grado titulado *Sistema de Navegación Semiautónomo en Interiores*.

En él se puede encontrar la siguiente información:

- **Conceptos teóricos:** Ofrecen una base teórica de la que partir, para llevar a cabo el desarrollo completo del proyecto.
- **Técnicas y herramientas:** Se trata de las implementaciones de los distintos conceptos teóricos anteriormente descritos.
- **Aspectos relevantes del desarrollo del proyecto:** Proporciona información detallada que se ha tenido en cuenta durante las diferentes fases de desarrollo del proyecto.
- **Trabajos relacionados:** Se trata de una lista, junto con una breve descripción, de los diferentes proyectos, artículos científicos o trabajos relacionados con el proyecto llevado a cabo.
- **Conclusiones y líneas de trabajo futuras:** Detalla una serie de posibles mejoras, modificaciones e incluso derivaciones, que pueden surgir del proyecto realizado.

---

# **Objetivos del proyecto**

---

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que se plantean a la hora de llevar a la práctica el proyecto.

## **2.1. Objetivos marcados por Requisitos Funcionales**

- Diseñar un *drone* capaz de recorrer de forma segura un espacio en el que existan obstáculos.
- Diseñar un sistema de acceso seguro al *drone*. Tanto para controlarlo de forma remota, como para activar los mecanismos de control automatizados.
- Diseñar una interfaz web que permita la visualización en tiempo real de la cámara del *drone*, así como su control remoto por un operador.

## **2.2. Objetivos técnicos derivados**

- Implementar un protocolo de comunicación entre el sistema de control de vuelo de un *drone*, y una RaspberryPi o similar.
- Implementar la adquisición de información de una serie de sensores de ultrasonidos mediante una RaspberryPi.
- Implementar un algoritmo de evasión de obstáculos haciendo uso de sensores de ultrasonidos y del magnetómetro disponible en la controladora de vuelo, que permita el movimiento del *drone* por el interior de un entorno cerrado.

- Implementar un algoritmo de localización *sin* hacer uso de GPS.
- Implementar una solución de comunicación en tiempo real para vídeo, a través de una aplicación web, haciendo uso de WebRTC.
- Implementar una solución de control remoto en tiempo real, a través de una aplicación web, haciendo uso de WebSockets.
- Lograr que dicha solución de control remoto permita la utilización de una emisora, u otro tipo de Joystick con el número de ejes necesario, para controlar el *drone* de forma adecuada.
- Hacer uso de protocolos seguros, como SSL/TLS, para el desarrollo de las comunicaciones.
- Aplicar metodologías ágiles, como SCRUM, para el desarrollo del proyecto.
- Utilizar ZenHub como implementación de Kanban, y de Sprints mediante épicas<sup>1</sup>.
- Hacer uso de Git como sistema de control de versiones.
- Hacer uso de GitHub como repositorio remoto del sistema de control de versiones.
- Hacer uso de GitHub como implementación de SCRUM mediante su sistema de gestión de tareas.
- Mejorar el *drone* de que se dispone, incluyendo los componentes *hardware* necesarios para la realización del proyecto.

### 2.3. Objetivos personales

- Crear un **nuevo** sistema de navegación semiautónoma haciendo uso de drones.
- Realizar una implementación de algoritmos utilizados por empresas líder en el sector de sistemas autónomos.
- Realizar una aproximación a herramientas utilizadas en un entorno laboral.
- Profundizar en Python y algunas de sus librerías más utilizadas.

---

<sup>1</sup>Una *épica* es una tarea que por su complejidad, puede ser fragmentada en tareas más pequeñas.

- Disfrutar en la realización de algo tan extenso como un TFG.
- Convertir algo que comenzó como un hobby, en una opción de futuro.

---

# Conceptos teóricos

---

Esta sección aúna los diferentes conocimientos teóricos necesarios para la realización del proyecto. A continuación, y en este orden, se explicarán los algoritmos utilizados, sistemas físicos empleados y protocolos de comunicación.

## 3.1. Algoritmia

### *Particle Filter localization*

Conocer la localización del agente dentro del mapa es fundamental. No es posible realizar un planeamiento de ruta hacia un destino, como el que realiza el sistema de evasión de obstáculos definiendo unas metas, sin que el agente conozca su posición. Dada la naturaleza del proyecto, el uso de un GPS para lograr la localización del agente queda descartado. Se requiere de un sistema de localización lo más preciso posible en un interior, por ello se hace uso de un algoritmo basado en probabilidades.

Los Filtros de Partículas son modelos utilizados para tratar de estimar el estado de un sistema que cambia con el tiempo.

Fue definido como *bootstrap filter* en 1993 por N. Gordon, D. Salmond y A. Smith [1]. Se trata de un método que pretende implementar filtros bayesianos recursivos, haciendo uso del método de Montecarlo, es decir, realiza repetidas medidas del estado para estimar la variable de interés, en el caso de este proyecto la *localización* del sistema. De esta forma, estimará la posición y orientación de un agente en un entorno, haciendo uso de múltiples hipótesis ponderadas [2]. Dicha ponderación, peso o probabilidad, se basa en la similitud de la hipótesis, o partícula, con la instancia que representa el agente.

Para lograr obtener información del entorno, y poder dar una serie de atributos que determinen la afinidad del agente con las partículas, se dispone de los sensores detallados en la subsección 3.2, que proporcionan la distancia

del agente al obstáculo hacia el que se dirige el sensor. Haciendo uso de múltiples sensores de distancia, con diferentes orientaciones, se obtienen múltiples distancias a obstáculos, de forma que se reduce la incertidumbre, dado que se dispone de múltiples medidas del entorno que podrán ser comparadas con las del agente.

Habitualmente, se comienza con una distribución de partículas aleatoriamente dispersas por el mapa y con una orientación definida, obviamente no se establecen partículas dentro de zonas ocupadas por obstáculos. Esto, representa el total desconocimiento de la posición del agente, es decir, todas las hipótesis son equiprobables y por lo tanto la posición del agente es, equiprobablemente, cualquier posición del mapa.

A medida que el agente comienza a moverse las partículas se mueven en la dirección y orientación que determina el agente, por ejemplo: Si el agente ha rotado  $10^\circ$  y se ha desplazado hacia delante 10 cm, todas las partículas deberán rotar  $10^\circ$  y desplazarse hacia delante 10cm. A continuación el agente y las partículas toman medidas de su entorno haciendo uso de los sensores de distancia, en el caso del agente. Cómo calculan las partículas dichas distancias será explicado más adelante.

Una vez realizado el movimiento, la distribución de partículas es remuestreada en base a estimación Bayesiana recursiva, esto es, cuanto se parecen las distancias medidas por el agente a las distancias medidas por cada partícula. O como de parecidas son las hipótesis al estado real del sistema. El remuestreo, o *resample*, conlleva reducir la población de partículas descartando las menos parecidas, o menos probables. De esta manera al cabo de cierto número de iteraciones la distribución converge hacia la posición real del agente.

Conviene tener en cuenta que ni los movimientos (tanto de rotación como de traslación), ni las medidas del agente son perfectos, así que se puede añadir cierto *ruido*, distribuido de forma Gaussiana por ejemplo, al movimiento y medidas del agente. Esto aporta incertidumbre sobre el estado real del agente, lo cual hará que el filtro pueda tardar más iteraciones en converger, pero proporciona un *cubrimiento* más adecuado del estado real del agente.

## Representación del estado

El *estado* del agente en el caso de este proyecto estará compuesto de las coordenadas del agente en el plano, y su orientación. Se puede representar mediante una  $n$ -tupla, donde  $n = 3$  en este caso, tal que:  $[x, y, \theta]$  siendo  $x$  e  $y$  las coordenadas de posición y  $\theta$  la orientación. La altura del agente no es tenida en cuenta, dado que se establece esta como fija, es decir, no existen zonas en las que el agente deba navegar a menor o mayor altura para poder acceder, de manera que no es relevante para el cálculo de la posición conocer este valor.

El *peso* de las partículas, o estimación de la probabilidad de que la partícula sea la ubicación real del agente, es una función de densidad de probabilidad distribuida sobre el espacio de estados, descrito en [3]. En el *Filtro de Partículas* la variable de interés, la posición del agente, es representada por un conjunto de  $M$  partículas en el instante  $t = k$ , tal que  $S^k = [e_j^k, w_j^k] : j = 1..M$  tal y como se muestra en [2]. Cada partícula tiene asignado un peso,  $w_j^k$  que define la contribución de esa partícula a la estimación de la posición.

Aquellas regiones del espacio de estados con una gran cantidad de partículas, se corresponden con zonas en las que hay una alta probabilidad de que el agente se encuentre en ellas. Aquellas zonas con una densidad de partículas baja, representan zonas en las que es improbable que el agente se encuentre.

Claramente es un método que puede ser utilizado como estimador del estado de cualquier sistema.

El algoritmo asume la *propiedad de Markov*<sup>2</sup>, de manera que solo funciona correctamente si el entorno es estático. Por lo general, el agente no tiene información de su localización, así que las partículas se distribuyen de forma uniforme. En implementaciones futuras, se podría hacer uso de SLAM<sup>3</sup>, para evitar tener que codificar el mapa.

## Actualización de movimiento

Durante la actualización de movimiento, descrita en [2, 4], el agente predice su nueva localización basándose en el movimiento, supuestamente, realizado. Tal y como se ha descrito con anterioridad, si el agente rota  $10^\circ$  en el sentido de las agujas del reloj, todas las partículas rotan en ese mismo sentido el mismo número de grados. Sin embargo la capacidad del agente de rotar o trasladarse no es perfecta. Si el sistema de evasión de obstáculos indica al agente que debe rotar  $n^\circ$  para dirigirse hacia la meta por un camino seguro, el agente *tratará* de rotar  $n^\circ$  pero es muy probable que infra/sobrecorra. Si el agente trata de desplazarse en línea recta inevitablemente escorará en una dirección u otra, pese a los sistemas de control embebidos en la controladora de vuelo.

El movimiento *perfecto* se define de la siguiente manera para el instante de tiempo  $t = k$ :

$$\begin{bmatrix} x' \\ y' \\ \hat{\theta}' \end{bmatrix} = \begin{bmatrix} x + \rho \times \cos(\hat{\theta}_k) \\ y + \rho \times \sin(\hat{\theta}_k) \\ \hat{\theta}_k \end{bmatrix} \quad (1)$$

---

<sup>2</sup>En un proceso estocástico que cumple la propiedad de Markov, el estado futuro depende únicamente del estado presente, y no de los estados pasados.

<sup>3</sup>*Simultaneous Localization and Mapping* es una técnica que permite la localización y el mapeo simultáneo de un agente en un entorno desconocido

Donde:

$$[x, y, \hat{\theta}]^T = \text{Posición inicial del agente}$$

$$\hat{\theta} = \arctan \frac{\delta y}{\delta x}$$

$$[x', y', \hat{\theta}']^T = \text{Posición final del agente}$$

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}; \text{ la distancia a recorrer}$$

Aunque para el cálculo de  $\hat{\theta}$  el agente dispone de magnetómetro; ver subsección 3.2 Sin embargo, se debe compensar el *ruido* existente en los sensores y sistemas de movimiento, quedando la ecuación 1:

$$\begin{bmatrix} x' \\ y' \\ \hat{\theta}' \end{bmatrix} = \begin{bmatrix} x + \rho \times \cos(\hat{\theta}_k) \\ y + \rho \times \sin(\hat{\theta}_k) \\ \hat{\theta}_k + f(\text{rand}|\mu, \sigma^2) \end{bmatrix} \quad (2)$$

Donde:

$$[x, y, \hat{\theta}]^T = \text{Posición inicial del agente}$$

$$\hat{\theta} = \arctan \frac{\delta y}{\delta x}$$

$$[x', y', \hat{\theta}']^T = \text{Posición final del agente}$$

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \times f(\text{rand}|\mu, \sigma^2); \text{ la distancia a recorrer } \mathbf{añadiendo ruido}$$

dados por una distribución normal

$$\mu = 0,0; \text{ se desea un valor aleatorio dentro de una gaussiana con centro en 0.0}$$

$$\sigma = \text{Errores de los sensores o sistemas de movimiento}$$

El ruido, como puede verse en la ecuación 2, se genera mediante una función de distribución normal, como la detallada en la ecuación 3, con centro en 0,0 de manera que se pueda infra/sobrecorrejir el movimiento.

De forma inevitable, las partículas que componen el filtro divergirán como consecuencia de la actualización de movimiento aplicando la ecuación 2. Por ello es necesario tomar medidas del entorno.

### Actualización de sensores

Cuando el agente toma medidas a través de sus sensores, actualiza la distribución de partículas de forma ponderada, tal y como se describe en [2, 4], es decir, en base a la probabilidad de que una partícula se corresponda con el estado actual del agente, de manera que aquellas que más afinidad presentan en sus mediciones, serán tenidas en cuenta más a menudo que aquellas que no se parezcan al agente.

Para ello, se calcula para cada partícula la probabilidad,  $x_j^k$  de ser la posición real del agente basándose en el estado que esta representa, y haciendo uso de una función de distribución normal, o Gaussiana:

$$x_j^{k+1} = f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (3)$$

Donde:

$\mu$  = Distancias de la partícula a los diferentes obstáculos.

$\sigma$  = Ruido del sistema de sensores.

$x$  = Distancias reales obtenidas por los sensores del agente.

Cabe destacar, que la probabilidad  $x_j^{k+1}$  tiene como superíndice  $k + 1$ , determinando este el instante de tiempo siguiente, es decir la probabilidad de la partícula  $j$  **después** de haber realizado el movimiento.

Se le asignará a cada partícula un peso, proporcional a la probabilidad de tratarse de la ubicación del agente:

$$w_j^{k+1} = w_j^k * x_j^{k+1} \quad (4)$$

Dicho peso será normalizado de la forma habitual mediante:

$$w_j^{k+1} = \frac{\tilde{w}_j^k}{\sum_j^M \tilde{w}_j^k} \quad (5)$$

Donde  $\tilde{w}_j^k$  es la medida de peso de una partícula  $j$  en el instante de tiempo  $k$ , antes de ser normalizado.

### Remuestreo

Una vez que se han obtenido los pesos, el algoritmo deberá generar un nuevo conjunto de partículas a partir del anterior, teniendo en cuenta el peso de cada partícula. Es decir, si se deben generar  $N$  nuevas partículas, el algoritmo deberá escoger de entre el conjunto  $M$  de partículas, dando mayor prioridad a aquellas que tengan mayor peso. Cabe destacar, que el proceso de selección ha de ser con reemplazo, de manera que puede seleccionarse múltiples veces la misma partícula.

De esta forma se genera un nuevo conjunto de partículas  $M_{k+1}$ , en el que la mayor parte de las partículas se encontrará en aquellas posiciones que proporcionen medidas parecidas a las obtenidas por el agente real.

## Campos Potenciales

A la hora de lograr una navegación segura en un entorno determinado, es necesario implementar un sistema de evasión de obstáculos.

Introducido por Oussama Khatib en 1986 [5], el algoritmo de Campos Potenciales aporta una manera de evitar colisionar con los diferentes obstáculos existentes.

Se basa en la idea de que el agente se mueve en un campo de fuerzas. La posición que debe alcanzar, su meta o destino, se presenta como una fuerza de atracción, y los obstáculos como fuerzas repulsivas. Los diferentes vectores fuerza, determinan la dirección y magnitud de la fuerza atrayente o repulsora.

Sea  $q$  la posición actual del agente, considerada como una partícula moviéndose en un espacio  $n$ -dimensional  $\mathbb{R}^n$ . Por simplicidad, se considera la posición como una tupla, esto es, bidimensional, tal que  $q = (x, y)$ . El campo potencial en el que se encuentra el agente es una función escalar  $U(q) : \mathbb{R}^2 \rightarrow \mathbb{R}$ , generado por la superposición de los campos atrayentes  $U_{att}$  y repulsores  $U_{rep}$ :

$$U(q) = U_{att}(q) + U_{rep}(q) \quad (1)$$

Por supuesto, la suma de todos los campos repulsores  $U_{rep}$  generan un vector repulsor, que sumado al de atracción puede determinar la necesidad de acercarse o alejarse de una determinada zona:

$$U(q) = U_{att}(q) + \sum_i U_{rep_i}(q) \quad (2)$$

Donde  $U_{rep_i}$  representa el campo potencial generado por un obstáculo  $i$ .

Suponiendo que  $U(q)$  es diferenciable, en cada punto  $q$ , el campo potencial  $\nabla(q)$  es un vector que apunta en la dirección que, **localmente**, incrementa  $U(q)$ .

De esta forma, si el potencial de atracción en la meta se considera 0 y a medida que el agente se distancia de ella, se incrementa, se puede establecer, teniendo en cuenta el potencial de repulsión producido por los diferentes obstáculos, un vector  $F(q)$  cuya dirección apunta hacia donde se reduce el potencial  $U$ , y su magnitud establece la velocidad con que este se reduce:

$$F(q) = F_{att}(q) + F_{rep}(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q) \quad (3)$$

**El potencial de atracción**  $U_{att}(q)$  se establece como una función cuadrática proporcional a la distancia a la meta. De esta forma crece exponencialmente a medida que el agente se aleja de su destino, y se reduce considerablemente en su cercanía. Así puede ser utilizado como fuente de información para establecer

la velocidad de aproximación al objetivo, de forma que el agente no sobreactúe:

$$U_{att}(q) = \epsilon \times \delta_{meta}(q)^2 \quad (4)$$

Donde  $\delta$  es la distancia euclíadiana desde la posición del agente  $q$  a la meta  $meta$ , y  $\epsilon$  es un factor de escalado positivo, que puede utilizarse para ajustar la influencia de la distancia en la velocidad del agente. Y su gradiente negativo:

$$-\nabla U_{att}(q) = -\epsilon \times (q - q_{meta}) \quad (5)$$

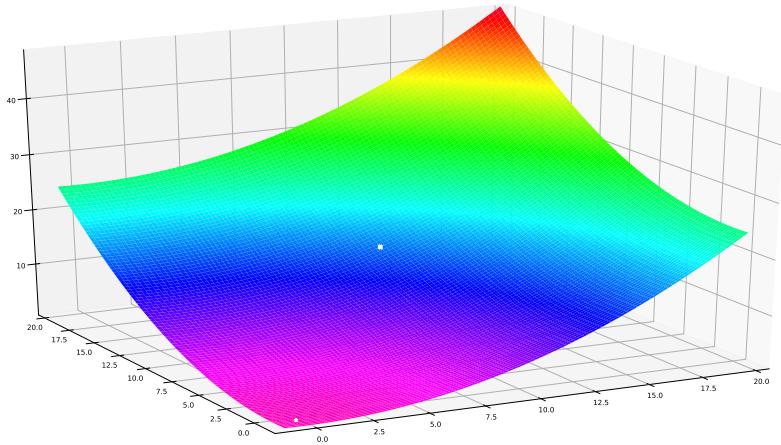


Figura 3.1: Gradiente de Potencial de Atracción.

En la Figura 3.1 se ha establecido el agente en el punto  $(10, 12)$  y la meta en  $(0, 0)$ . Puede verse el efecto del gradiente aplicado.

**El potencial de repulsión**  $U_{rep}(q)$  lo establecen los obstáculos detectados por los sensores del agente. Si se establece una función proporcional a la distancia del agente a los obstáculos, el potencial repulsor crece en la cercanía a estos, y decrece con la distancia. De esta forma un obstáculo lejano al agente no debería presentar ninguna fuerza sobre él:

$$U_{rep}(q) = U = \begin{cases} 0 & \text{si } \delta > \rho \\ \eta \times \left( \frac{1}{\delta_{obs}(q)} - \frac{1}{\rho} \right)^2 & \text{si } \delta \leq \rho; \delta \neq 0 \end{cases} \quad (6)$$

Donde  $\eta$  es un factor de escalado positivo, que permite establecer con qué intensidad se ve afectado el agente por el campo repulsor,  $\delta$  es la distancia euclíadiana desde la posición del agente  $q$  al obstáculo  $obs$ , y  $\rho$  es un factor de

escalado positivo, que permite establecer la distancia de influencia de forma proporcional. Y su gradiente

$$-\nabla U_{rep}(q) = \begin{cases} 0 & \text{si } \delta > \rho \\ \eta \times \left( \frac{1}{\delta_{obs}(q)} - \frac{1}{\rho} \right) \times \frac{q - q_{obs}}{\delta_{obs}(q)^2} & \text{si } \delta \leq \rho; \delta \neq 0 \end{cases} \quad (7)$$

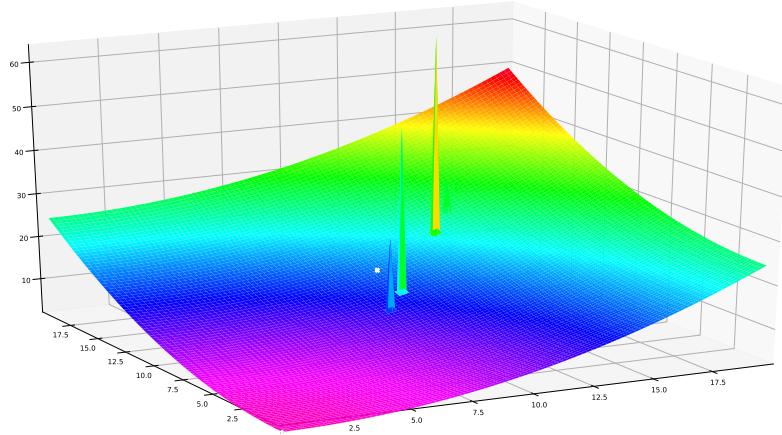


Figura 3.2: Gradiente de Potencial de Repulsión.

En la Figura 3.2 se ha creado una serie de obstáculos artificialmente para ilustrar el comportamiento del algoritmo. El agente se encuentra en la posición [10, 12], la meta esta establecida en [0, 0] y existen obstáculos en las posiciones [3, 2], [5, 5], [9, 12], [10, 11]. El algoritmo no está teniendo en cuenta la fuerza repulsora que representa el obstáculo situado en [3, 2] por encontrarse este demasiado lejos del agente.

### Desventajas

El algoritmo de campos potenciales se puede llevar a cabo con una implementación simple y eficiente haciendo uso de Numpy, dado que se puede crear una representación matricial del plano en que se desplaza el agente, que contenga los diferentes potenciales en cada posición. Esto hace sencillo realizar cálculo vectorizado sobre las matrices.

Sin embargo, tiene ciertas desventajas, tal y como publicaron Borenstein y Koren [6], que lo han hecho, cuanto menos, inútil para las necesidades de este proyecto:

- Es un algoritmo basado en descenso de gradiente, y como tal puede quedar *atascado* en mínimos locales. Estos mínimos pueden encontrarse en zonas que contengan obstáculos en forma cóncava o de caja abierta, tal como se ilustra en la Figura 3.3.



Figura 3.3: Mínimo local en zonas convexas.

- De encontrarse en un espacio con forma de corredor o pasillo, cualquier acercamiento hacia las paredes ocasionará una corrección en el sentido

opuesto al campo repulsor, lo cual puede ocasionar un movimiento oscilatorio si dicha corrección supone aproximar el agente a la pared opuesta, tal y como se ilustra en la Figura 3.4 extraída de [6].

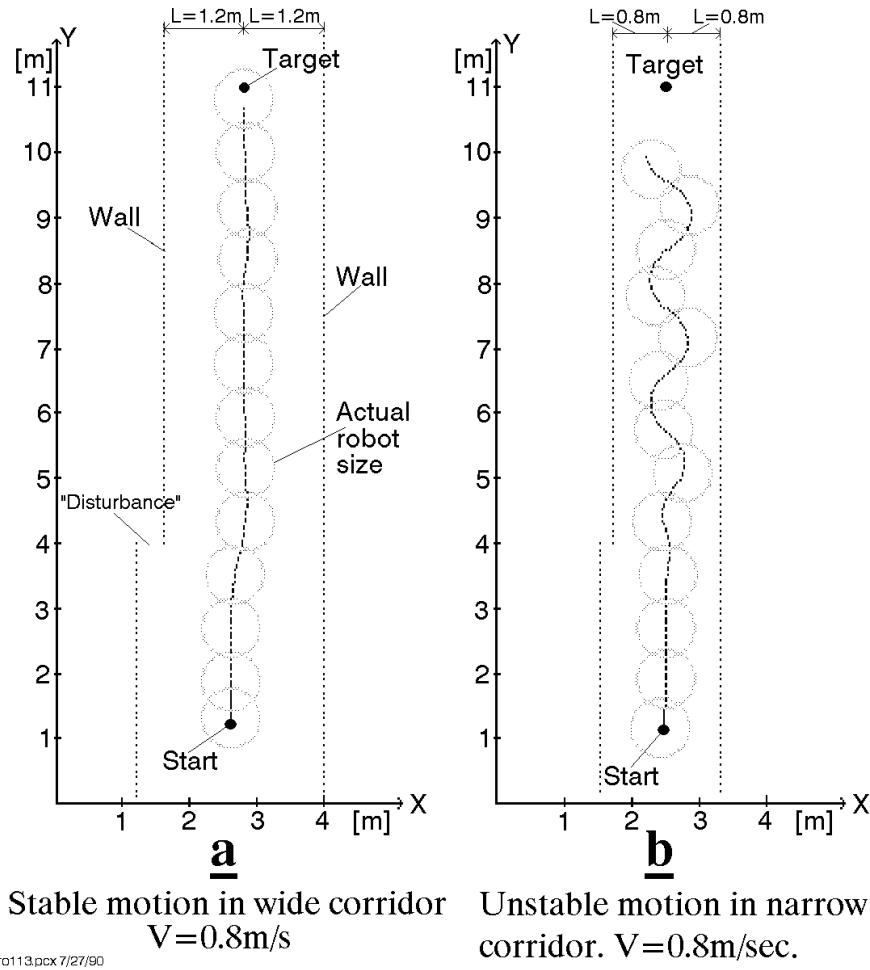


Figura 3.4: Movimiento oscilatorio en corredores estrechos. Imagen extraida de [7]

- Alta complejidad computacional pese a poder modelarse como cálculo vectorizado, requiere de muchas más operaciones que otros métodos más avanzados.
- No funciona correctamente con agentes que se desplazan a alta velocidad.
- Al realizar la adición de todos los campos potenciales, tanto repulsores como atrayente, se produce una gran pérdida de información sobre la distribución local de los obstáculos.

Por estos motivos, se ha desecharado la implementación realizada del algoritmo de Campos Potenciales como sistema de evasión de obstáculos, y se ha pasado a un modelo más avanzado basado en un plano cartesiano, conocido como *Vector Field Histogram* y detallado en el apartado [Vector Field Histogram](#).

## Vector Field Histogram

El Vector Field Histogram, o Histograma de Campos Vectoriales, *VFH* de aquí en adelante, fue introducido por Borenstein y Koren en el año 1991 [8, 9]. Permite la detección de obstáculos y su evasión mediante el control de la dirección y velocidad del agente en tiempo real.

Para ello realiza una representación del entorno del agente en forma de histograma. Dicho entorno, llamado *Región Activa C\** se representa como una *ventana* que se desplaza con el agente en el centro. Se trata un algoritmo de búsqueda local, sin embargo se ha mostrado que suele producir rutas cercanas al óptimo, aunque en el caso de este proyecto no es relevante, dado que la intención es que se exploren todas las zonas del entorno.

Es un algoritmo más robusto, dado que presenta cierta insensibilidad a lecturas erróneas, y eficiente que el algoritmo de Campos Potenciales explicado en la subsección [3.1](#).

Se compone de tres partes:

1. Red Cartesiana de Histogramas: Derivado del concepto de *malla de certidumbre* propuesto por Moravec y Elfes en 1985 [10] en la universidad de Carnegie Mellon. Se construye un plano cartesiano representando el entorno cercano del agente. Por *cercano*, se entiende una distancia que los sensores de distancia puedan abarcar, dado que no tendría sentido tratar de computar zonas inalcanzables en un momento determinado. Dicho plano será representado como un histograma, y de ahí su nombre habitual *Cartesian Histogram Grid*. Ver subsección [3.1](#).
2. Histograma Polar: Una representación unidimensional, obtenida a partir de una reducción de la Red Cartesiana definida en el punto anterior. Se compone de  $n$  sectores angulares  $k$ , de anchura  $\alpha$ , de forma que  $n \times \alpha = 360; n \in \mathbb{Z}$ . Dichos sectores  $k$  contienen un valor  $h_k$  que representa la densidad de obstáculos, *Polar Obstacle Density* o *POD* de aquí en adelante, contenida en él. Ver subsección [3.1](#).
3. Capa de salida: Representa el resultado del algoritmo. A partir del POD, se obtienen los posibles *valles*<sup>4</sup> candidatos existentes entre obstáculos,

---

<sup>4</sup>Así denominados por su representación en forma de histograma

y se elige el que más se aproxime a la dirección de la meta establecida. Finalmente, entrega los valores de cambio de dirección necesario en cada momento. Ver subsección 3.1.

### Cartesian Histogram Grid

Se modela el espacio cercano al *drone* en forma de malla. En cada celda  $(i, j)$  de esa malla se encuentra un valor  $c_{i,j} \in \mathbb{Z}$  que establece la certeza de la existencia de un obstáculo. La cantidad de celdas existentes en la malla se establece en función de los límites del sensor, y teniendo en cuenta que es necesario establecer una precisión  $c$ , por ejemplo: Si se dispone de un sensor con un rango de medidas de hasta 400 cm se puede crear una malla de  $81 \times 81$ , suponiendo que cada celda tenga un tamaño de  $c = 5, 5 \times 5\text{cm}$ , en cuyo centro se encuentra el *drone*, dejando 40 filas y columnas a cada lado. Y estableciendo así, un perímetro de medidas de 200 cm en cada dirección desde el *drone*. De esta forma se consigue una precisión de  $\pm 5\text{cm}$

Dicho valor se obtiene de la lectura proporcionada por el sensor de distancia, en este caso un sonar, y se establece en el centro del ángulo de barrido del sonar, ver subsección 3.2. La eficiencia superior de este algoritmo se basa precisamente en incrementar el valor  $c_{i,j}$  de únicamente una celda con cada medida. Dicha celda  $(i, j)$  se encuentra a una distancia de  $R = \frac{\delta_{obs}q}{c}$  celdas y en la bisectriz del ángulo barrido por el sonar. Si bien está solución puede parecer una simplificación del problema, se llega a generar una distribución probabilística tomando múltiples medidas de forma continua mientras el agente se desplaza por el entorno. Por tanto, la misma celda, cercana a un obstáculo, y sus vecinas serán incrementadas repetidas veces, tal y como se muestra en la Figura 3.5.

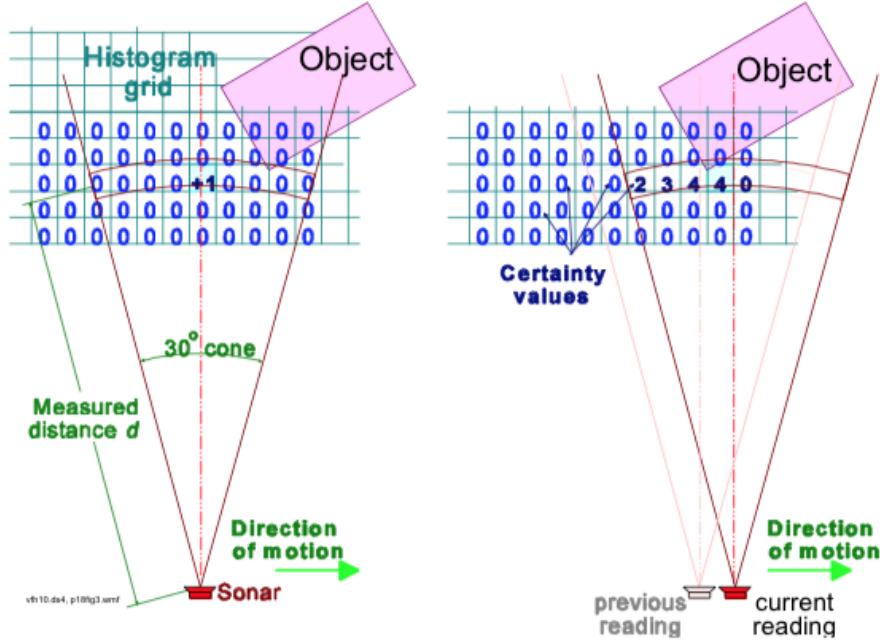


Figura 3.5: Distribución Histográfica de Probabilidades. Extraida de [8]

### Polar Histogram

A partir de las medidas obtenidas en el Histogram Grid, se realiza la primera reducción de información para construir un Histograma Polar. Para ello, los contenidos de la región activa  $C^*$  son tratados como un *vector de obstáculos* cuya dirección está definida por la dirección  $\beta$  de la celda en cuestión al centro del agente  $VCP$ <sup>5</sup>, y que viene definida por la función:

$$\beta_{i,j} = \tan^{-1} \frac{y_i - y_0}{x_i - x_0} \quad (8)$$

La ecuación 8 devuelve un valor  $\beta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$  rad. Al hacer uso de Numpy, es sencillo calcular de forma vectorizada los ángulos en los que se encuentra cada celda de la región activa  $C^*$  con respecto al agente que se encuentra en su centro, de forma vectorizada. Tal y como puede verse en la Figura 3.6.

---

<sup>5</sup>El VCP o *Vehicle Center Point* se define como el centro geométrico del vehículo. En nuestro caso, al tratarse de un *drone*, se considera el centro del mismo como VCP

<b>-135</b>			<b>-90</b>			<b>-45</b>
	<b>-135</b>		<b>-90</b>		<b>-45</b>	
		<b>-135</b>	<b>-90</b>	<b>-45</b>		
<b><math>\pm 180</math></b>	<b><math>\pm 180</math></b>	<b><math>\pm 180</math></b>		0	0	0
		<b>135</b>	<b>90</b>	<b>45</b>		
	<b>135</b>		<b>90</b>		<b>45</b>	
<b>135</b>			<b>90</b>			<b>45</b>

Figura 3.6: Ángulos relativos al VCP del *drone*.

Nótese que para llevar a cabo la obtención de estos ángulos **no** se ha hecho uso de la función `arctan` disponible en Numpy, dado que esta devuelve valores, en radianes, pertenecientes al  $1^{\circ}$  y  $4^{\circ}$  cuadrante únicamente. En su lugar, se ha utilizado la función `arctan2`, que tiene en cuenta el signo de las componentes del vector dirección  $[y_i - y_0, x_i - x_0]$  resultante, de forma que devuelve el ángulo de giro, sea en sentido horario (valores positivos) o antihorario (valores negativos), más corto posible<sup>6</sup>.

La magnitud de cada zona es entonces calculada de la siguiente forma:

$$m_{i,j} = c *_{i,j}^2 \times (a - bd_{i,j}) \quad (9)$$

---

<sup>6</sup>Obviamente es lo deseable, no sería eficiente realizar un giro de  $225^{\circ}$  en el sentido de las agujas del reloj, cuando uno de  $135^{\circ}$  en el sentido contrario a las agujas del reloj bastaría.

Donde:

$m_{i,j}$  = Magnitud de obstáculos en la celda  $(i, j)$

$c*_{i,j}$  = Certidumbre de obstáculo en la celda, en la región activa,  $(i, j)$

y:

$a$  = Representa la fuerza con que un obstáculo afecta al *drone*

$b$  = Representa la distancia desde la que un obstáculo afecta al *drone*  
son positivos

El plano es convertido en una secuencia de sectores que cubren los  $360^\circ$ . Se definen, por tanto,  $n \in \mathbb{Z}$  sectores  $k$  de amplitud  $\alpha$ , por ejemplo  $n = 72; \alpha = 5$ . Y se distribuyen las medidas tomadas anteriormente en los diferentes ángulos relativos al VCP del *drone*, en los sectores  $k_i$  correspondientes.

$$k = \frac{\beta_{i,j}}{\alpha} \quad (10)$$

con un valor  $h_k$ , véase la ecuación 11, de densidad de obstáculos en ellos:

$$h_k = \sum m_{i,j} \quad (11)$$

Dada la discretización de los datos obtenidos del Polar Histogram el resultado, al realizar esta conversión y al obtener el sumatorio  $h_k$  de todos los valores pertenecientes a un sector  $k$ , puede parecer que el histograma varía de forma muy repentina entre dos sectores. Por ello se aplica una función de suavizado. Según J. Borenstein y Y. Koren en [8], la función de suavizado utilizada se corresponde con:

$$h'_k = \frac{h_{k-l} + 2h_{k-l+1} + \dots + lh_k + \dots + 2h_{k+l-1} + h_{k+l}}{2l + 1} \quad (12)$$

Donde  $l$  es una constante positiva a la que se le ha dado un valor de 5.

Sin embargo, existe una errata en el artículo publicado, y la ecuación 12 no es consistente, i.e: La función de suavizado parece poder expresarse de forma general tal que:

$$h'_k = \frac{1}{2l + 1} \sum_{n=-l}^l (l - |n| + 1) * h_{k+n} \quad (13)$$

$$\text{ej: } l = 5 \quad h'_k = \frac{1h_{k-5} + 2h_{k-4} + 3h_{k-3} + 4h_{k-2} + 5h_{k-1} + 6h_k + \dots + 2h_{k+l-1} + h_{k+l}}{2l + 1}$$

Como puede verse, el valor central  $6h_k$  correspondiente a  $n = 0$ , no coincide con la posición dada para ese valor de  $n$  en la función de suavizado propuesta por J. Borenstein y Y.Koren, donde el coeficiente que multiplica a  $h_k$  es igual a  $l$  en lugar de  $l + 1$ .

Por ello, se ha hecho uso de una función de suavizado preexistente en SciPy, en concreto en el módulo `signal`. Se ha elegido la función de `Hann`, llamada así por el meteorólogo austriaco Julius van Hann. Se conoce a esta función por el nombre de Campana de Coseno, y se define por la ecuación 14.

$$w(n) = 0,5 - 0,5\cos \frac{2\pi n}{M-1} \quad 0 \leq n \leq M-1 \quad (14)$$

Donde  $M$  es el número de puntos en la ventana de salida. En este caso tomará el valor de  $l$ .

De esta forma, de obtener un histograma de  $n$  sectores con valores  $h_k$  posiblemente dispares sectores  $k$  contiguos, se pasa a obtener un histograma suavizado, que será de mayor utilidad para el sistema de evasión de obstáculos. Véase la Figura 3.7

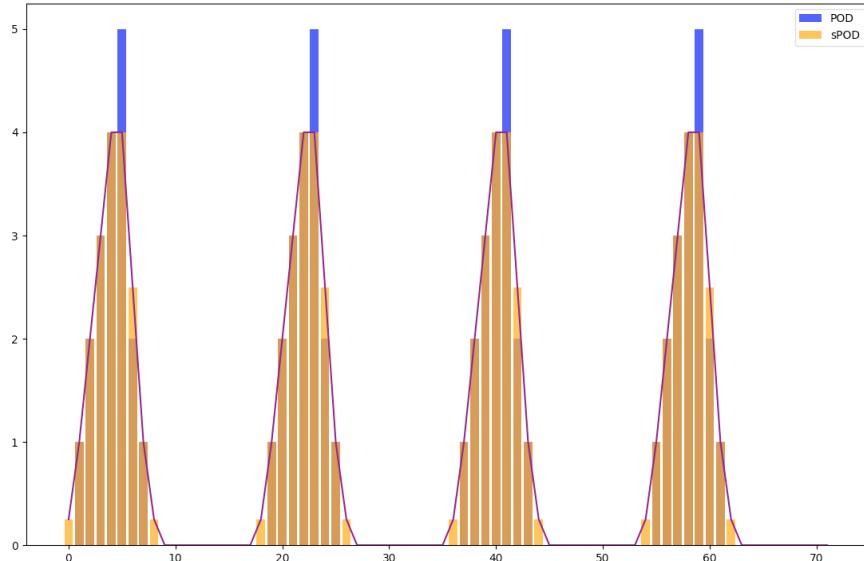


Figura 3.7: En azul Polar Obstacle Density (POD). En naranja el suavizado de este (sPOD).

### Output Layer

En el último estadio del algoritmo VFH, se computa el ángulo de giro  $\theta$  que se debe aplicar al *drone* para dirigirlo, por una ruta segura, hacia el destino establecido. Una vez obtenido el *Polar Obstacle Density*, y aplicado el suavizado correspondiente, se puede determinar en qué sectores  $k$  existe un *valle*. Se define un *valle candidato* como una zona, compuesta de varios sectores por los que es seguro navegar, es decir, están libres de obstáculos o su densidad de obstáculos está por debajo de cierto umbral.

Por lo general, se crean dos o más valles candidatos al analizar el entorno, de forma que es necesario elegir aquel que dirigirá al *drone* en dirección al destino  $k_{targ}$ . Una vez escogido el valle, se deberá seleccionar el sector  $k$  idóneo. Para ello se mide el tamaño del valle, es decir el número de sectores consecutivos por debajo del umbral que lo componen, de esta forma se distinguen dos tipos de valles, amplios y estrechos. Los valles amplios son el resultado de espacios grandes entre obstáculos, o de situaciones en las que únicamente existe un obstáculo lo suficientemente cerca del *drone*. Por *grande* se define una constante  $s_{max}$  que determina el número de sectores  $k$  que componen un valle amplio.

El sector más cercano a  $k_{targ}$  y por debajo del umbral se denomina  $k_n$ , y representa el *borde cercano* del valle. El más lejano  $k_f$  que en el caso de los valles amplios coincide con el valor de  $s_{max} + k_n$ . Véase la Figura 3.8.

El sector que debe seguir el *drone* para dirigirse hacia  $k_{targ}$  sin peligro, se denomina  $\theta_{yaw}$ , y su valor es la media de los valores de  $k_n$  y  $k_f$ , como puede verse en la ecuación 15.

$$\theta_{yaw} = \frac{k_n + k_f}{2} \quad (15)$$

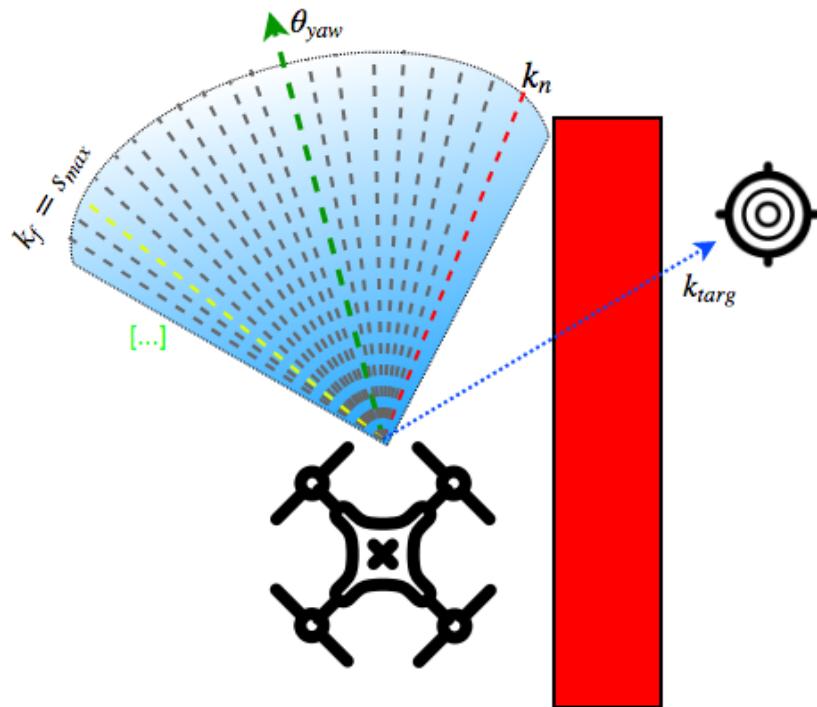


Figura 3.8: Valle ancho encontrado entre el obstáculo, a la derecha, y un espacio abierto, a la izquierda.

Sin embargo, la ecuación 15 presenta un comportamiento errático cuando el destino se encuentra dentro de un valle, dado que se establece  $\theta_{yaw}$  como la media entre los sectores cercano y lejano, al encontrarse la meta en un sector dentro del valle, el *drone* no se aproxima a ella, sino que realiza círculos cada vez más cerrados hasta llegar al destino, como puede verse en la Figura 3.9

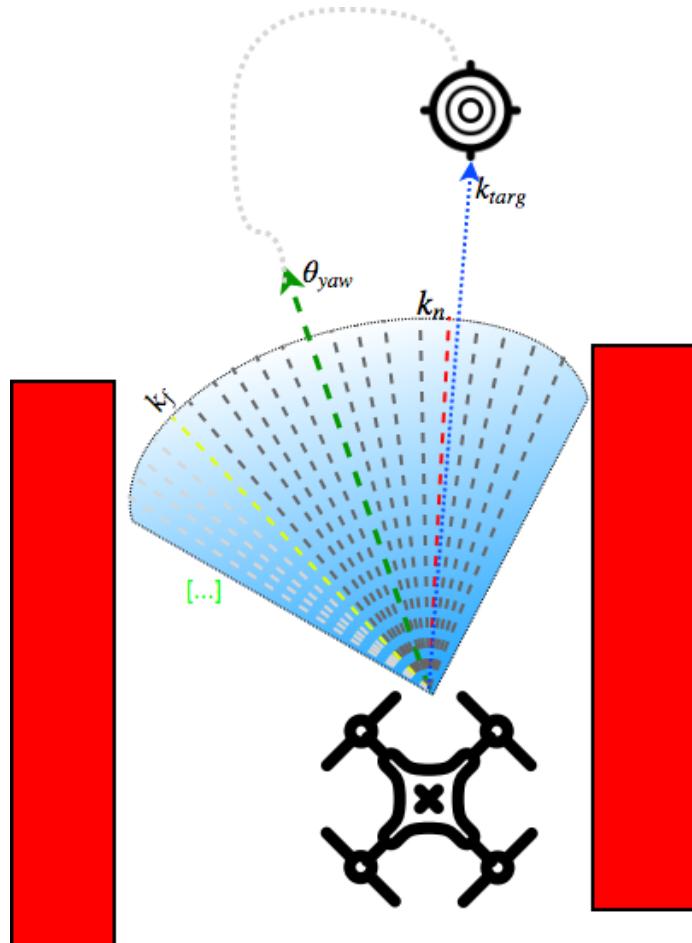


Figura 3.9: Movimiento circular al encontrarse  $k_{targ}$  en un sector seguro por ecuación 15

La intención tras la ecuación 15 es la navegación segura en entornos con obstáculos cercanos, ya que aporta la cualidad de escoger la zona central entre dos obstáculos para usarla como zona de navegación segura<sup>7</sup>. Además en el momento en que la posición del *drone* sea intermedia entre dos obstáculos, la

<sup>7</sup>Una cualidad de la que el algoritmo de campos potenciales adolecía, generando movimientos oscilatorios en espacios cerrados, como ya se vió en la subsección 3.1.

ecuación 15 dará como resultado el mismo sector, con lo que se consigue un movimiento rectilíneo.

Sin embargo, en el momento en el que el destino  $k_{targ}$  se encuentra en un sector  $k$  por el que es seguro navegar, no existe la necesidad escoger la zona intermedia del valle, sino que es deseable dirigirse hacia él directamente.

En el caso de *valles estrechos*, la mecánica es prácticamente la misma. Para este caso el valor del último sector con un POD por debajo del umbral escogido será  $k_f < s_{max}$ . De nuevo se utiliza la ecuación 15 para obtener el valor de  $\theta_{yaw}$ , siempre centrado entre los dos obstáculos, ilustrado en la Figura 3.10.

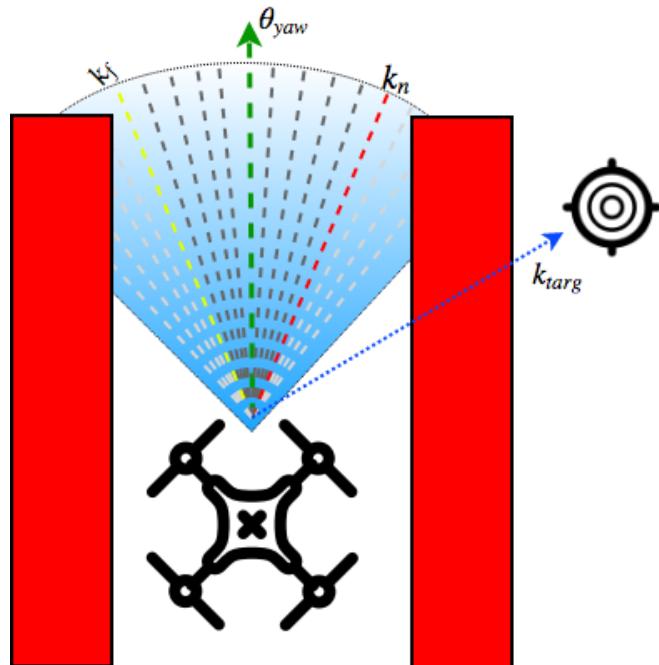


Figura 3.10: Valle estrecho encontrado entre dos obstáculos. La ecuación 15 proporciona un sector seguro ubicado en la zona central del valle.

**Umbral** Se ha referenciado numerosas veces el *umbral* que parece definir un valle como candidato. Su definición se ha dejado para el final debido a que, el VFH presenta una gran robustez ante configuraciones probablemente erróneas. Aumentar o reducir el umbral, únicamente, afecta a la anchura de los valles candidatos cuando estos son estrechos. En el caso de los valles anchos sencillamente se aumenta o reduce la distancia existente entre el obstáculo y el *drone*.

Por ejemplo, establecer un umbral muy alto, ver Figura 3.11, hace que el

*drone* no sea consciente de la existencia de obstáculos y que se aproxime a ellos. Sin embargo, las repetidas medidas de un obstáculo en ese sector harán que el valor de certidumbre del *Polar Histogram* se eleve hasta superar el umbral, produciendo que el *drone* trate de variar su curso. De esta forma el *drone* se acercará mucho a los obstáculos y es posible que llegue a colisionar, si la velocidad de aproximación es demasiado rápida.

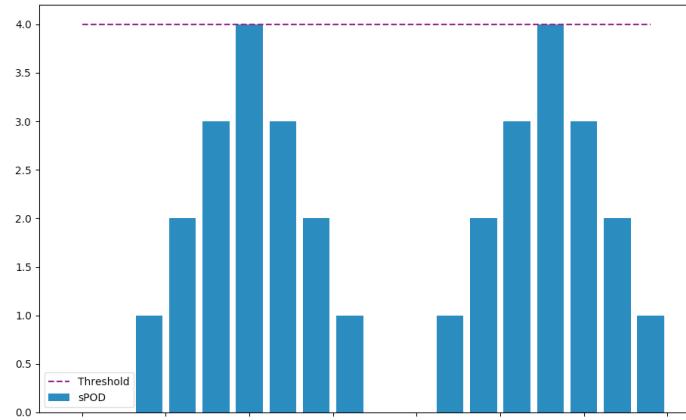


Figura 3.11: Con un umbral muy elevado, puede darse el caso de que todo sean valles hasta estar demasiado cerca del obstáculo.

Por el contrario, si el umbral es muy bajo, ver Figura 3.12, hará que los valles candidatos se vean reducidos, haciendo que el *drone* no pueda pasar por espacios estrechos.

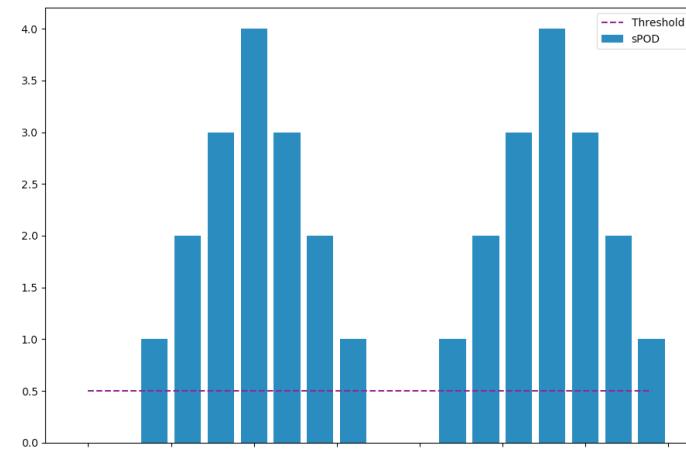


Figura 3.12: Con un umbral muy bajo, puede darse el caso de que los valles sean muy estrechos, o incluso inexistentes

### 3.2. Dispositivos Físicos

Dado que el desarrollo de este proyecto implica también la construcción a medida de un *drone*, en esta sección se explican los distintos componentes utilizados para dicha construcción.

#### Raspberry Pi

Componente	Raspberry Pi 3B
CPU	BCM2837
Núcleos	4
Velocidad	1.2GHz
RAM	1GB
Coms	Ethernet, WiFi, Bluetooth
USB	4 (2.0)
GPIO	40
Consumo máximo	6.7W

Tabla 3.1: Componentes de una Raspberry Pi 3 Model B

Una Raspberry Pi es un pequeño ordenador desarrollado en Reino Unido por la Raspberry Pi Foundation, con la intención de promover el aprendizaje de informática básica en colegios y países en desarrollo. El modelo base se compone de una única placa de medidas  $85mm \times 56mm$  ( $L \times A$ ), y unos 42g de peso.

Concretamente el modelo empleado es una Raspberry Pi 3B, y consta de los componentes detallados en la tabla 3.1

Su reducido tamaño y bajo consumo lo hacen ideal para este tipo de proyecto. En este caso se utiliza bajo una distribución GNU/Linux llamada Raspbian<sup>8</sup>, basada en Debian. Para tratar de mejorar su rendimiento y reducir al mínimo el consumo, se ha escogido la versión Lite del sistema, es decir, un sistema mínimo sin entorno de escritorio y con la mayor parte de servicios desactivados por defecto.

Una vez descargado el sistema, este ha sido instalado en una microSD mediante el siguiente procedimiento por consola<sup>9</sup>:

- `diskutil list` Permite localizar el dispositivo en el que se encuentra la tarjeta. En nuestro caso `/dev/disk4`

---

<sup>8</sup>Descargable desde: <https://www.raspberrypi.org/downloads/raspbian>

<sup>9</sup>Realizado en OSX, aunque en Linux es muy similar

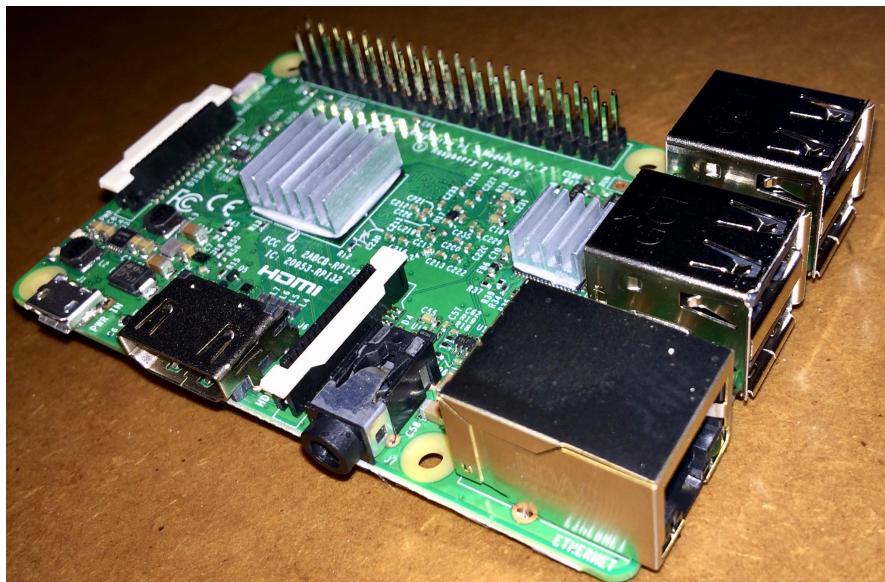


Figura 3.13: Raspberry Pi 3.

- `diskutil umountDisk /dev/disk4` Permite desmontar el volumen.
- `sudo dd if=raspbian-stretch.img of=/dev/rdisk4 bs=1m` El comando `dd` copia la entrada estándar a la salida estándar. Mediante `if/of` se establece el fichero de entrada/salida. Mediante `bs` se establece el tamaño de bloque a copiar. Se está utilizando `/dev/rdisk4` en lugar de `/dev/disk4` debido a la capacidad de OSX de trabajar con dispositivos en bruto, *raw*, de forma que es posible acceder al dispositivo de forma directa<sup>10</sup>, sin almacenar en un buffer la lectura del archivo, proporcionando velocidades de escritura/lectura hasta 20 veces más rápidas.

Una vez realizados estos pasos, se puede insertar la microSD en la Raspberry Pi. Para encenderla basta con utilizar el puerto micro-usb de que dispone. La Raspberry Pi 3 requiere de una fuente de alimentación capaz de proporcionar 2,5A<sup>11</sup> para funcionar al máximo nivel de estrés para el procesador y alimentar dispositivos USB.

Sin embargo, este no es estrictamente nuestro caso, véase la subsección 3.2. Se requiere un dispositivo cuyo consumo sea lo más reducido posible, pero que sea rápido en la ejecución, y que muestre poca latencia en operaciones de IO, que es donde se encuentra el cuello de botella.

<sup>10</sup>Véase `man hdiutil`, sección *DEVICE SPECIAL FILES*

<sup>11</sup>Véase <https://www.raspberrypi.org/help/faqs/#power>

```

CONFIG:
CLOCK : 100.000 MHz
CORE  : 400 MHz, turbo=0
DATA  : 512 MB, /root/test.dat

HDPARM:
=====
Timing O_DIRECT disk reads: 108 MB in 3.02 seconds = 35.78 MB/sec
Timing O_DIRECT disk reads: 108 MB in 3.02 seconds = 35.72 MB/sec
Timing O_DIRECT disk reads: 108 MB in 3.02 seconds = 35.75 MB/sec

WRITE:
=====
536870912 bytes (537 MB, 512 MiB) copied, 21.7232 s, 24.7 MB/s
536870912 bytes (537 MB, 512 MiB) copied, 20.7831 s, 25.8 MB/s
536870912 bytes (537 MB, 512 MiB) copied, 21.6338 s, 24.8 MB/s

READ:
=====
536870912 bytes (537 MB, 512 MiB) copied, 14.3944 s, 37.3 MB/s
536870912 bytes (537 MB, 512 MiB) copied, 14.3785 s, 37.3 MB/s
536870912 bytes (537 MB, 512 MiB) copied, 14.3567 s, 37.4 MB/s

```

Figura 3.14: Benchmark de lector microSD OC.

Una vez encendida, se accede a ella con el usuario por defecto `pi` y la contraseña por defecto `raspberry`. Obviamente ambas **han sido cambiadas** por motivos de seguridad.

## Modificaciones

- Se ha desactivado el puerto HDMI para reducir el consumo en ~30mA:  
Para ello se ha incluido en `/etc/rc.local` la línea `/usr/bin/tvservice -o`. Descrito en [11].
- Se ha *overclockeado* el lector de microSD a 100MHz, en lugar de los 50MHz por defecto:  
Para ello se ha incluido en `/boot/config.txt` la línea `dtparam=sd_overclock=100`. Y que arroja los resultados mostrados en la Figura 3.14. Descrito en [12] y [13].
- Se han incluido una serie de disipadores para evitar sobrecalentamiento de la placa. Así como un pequeño ventilador de bajo consumo.

## Controladora de Vuelo

Una controladora de vuelo (*FC, Flight Controller*, de aquí en adelante) es un pequeño circuito integrado, que contiene un procesador, una serie de sensores, y una serie de entradas y salidas. La FC se encarga de mantener el sistema de estabilización del *drone*, tomando medidas de los sensores de que

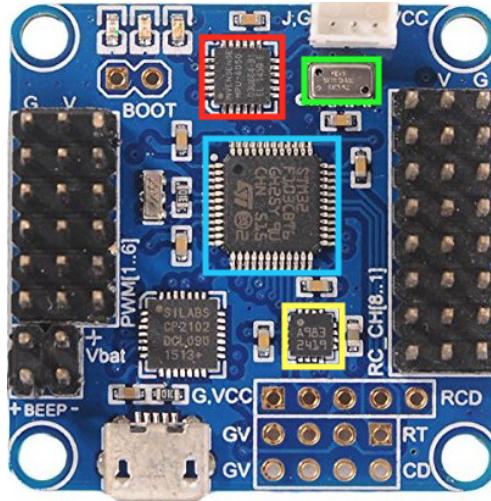


Figura 3.15: Controladora de Vuelo Flip32.

dispone, tales como un acelerómetro, giroscopio, magnetómetro, barómetro... etc.

En el caso de la FC usada para este proyecto, llamada Flip32 y mostrada en la Figura 3.15, se dispone de un IC MPU-6050, en rojo, con acelerómetro y giroscopio, así como de un barómetro M55611, en verde, y un magnetómetro HMC5883L, en amarillo. El núcleo de esta pequeña placa es un procesador STM32F103, en cyan, a 72MHz y basado en arquitectura ARM el cual dispone de dos puertos serie, que permiten establecer comunicación entre la FC y otros dispositivos, como emisoras u otros sensores.

El puerto micro-USB de que dispone es utilizado para establecer comunicación entre el configurador de opciones del *drone*, o en el caso de nuestro proyecto, para hacer uso del [MultiWii Serial Protocol](#).

### Entradas

En el lado derecho de la Figura 3.15 pueden verse una serie de pines que actúan como 8 canales de entrada desde el receptor de radio. Dichos canales de entrada, por defecto, reciben una señal modulada en ancho de pulso o [PWM](#)

### Salidas

En el lado izquierdo de la Figura 3.15, pueden verse otros pines que actúan como salida de señal hacia los controladores de velocidad de los motores del *drone* (ESC o *Electronic Speed Controller*). Estos pines de salida, emiten una señal que será interpretada por los ESC del *drone*, para determinar la frecuencia

dada al voltaje que alimenta los motores. En el caso de nuestro proyecto, los ESC disponibles reciben una señal PWM.

### Sensores

La controladora de vuelo Flip32 en su versión más completa, dispone de los siguientes sensores:

- IMU<sup>12</sup> MPU-6050: Se trata de un circuito integrado compuesto de un acelerómetro de tres (3) ejes y un giroscopio de tres (3) ejes. El acelerómetro mide las fuerzas en los tres diferentes ejes (en  $g$ ) , el giroscopio se encarga de medir la velocidad angular en cada uno de los tres ejes (en  $\text{deg}^\circ/\text{s}$  )
- Magnetómetro HMC5883L: Se trata de un pequeño magnetómetro digital capaz de medir el campo magnético terrestre (en Gauss). Se debe tener en cuenta que según la posición en el planeta, el campo magnético varía entre G0,25 - G0,65, así como la declinación magnética de la zona en la que se realiza la medición.<sup>13</sup>
- Barómetro M55611: Se trata de un altímetro de alta precisión, con resoluciones de hasta 10cm, que funciona midiendo la presión atmosférica (en milibar). Hay que tener en cuenta que los cambios de presión, como los generados por las hélices del *drone*, hacen variar la medida del sensor. Por ello, en el caso de usarlo, se cubre con un pequeño filtro de un material absorbente, lo suficientemente denso como para evitar el contacto directo entre una corriente de aire y el sensor.

### Sensores de distancia

Para medir la distancia existente entre los diferentes obstáculos y el agente, se ha hecho uso de sensores de ultrasonidos. Estos sensores, habitualmente llamados *sonar*, envían un pulso de ultrasonidos en la dirección en que apuntan, y reciben el eco de dicho pulso. Una vez recibido el eco, es posible determinar el tiempo que ha tardado en recorrer el espacio, sabiendo la velocidad del sonido en condiciones ideales, es posible conocer la distancia aproximada del objeto. La ecuación 16 refleja este cálculo.

$$e = \frac{2 * v}{t} \quad (16)$$

Donde  $e$  es el espacio recorrido,  $v$  es la velocidad del sonido, se suele tomar  $340\text{m/s}$ , y  $t$  es el tiempo transcurrido entre el envío del pulso y la recepción del eco. Se multiplica por 2 dado que el sonido tiene que ir y volver.

---

<sup>12</sup>Unidad de Medición Inercial.

<sup>13</sup>Disponible en: <http://magnetic-declination.com/>

Los sensores utilizados, tienen como nomenclatura HC-SR04 [14], y son muy conocidos por dar resultados relativamente buenos, y por su asequibilidad. Son capaces de obtener medidas de entre 400 cm y 2 cm con una precisión de  $\pm 5$  mm, pero para no forzar los extremos en este proyecto se ha establecido una distancia máxima de 350 cm y una mínima de 10 cm.

Disponen de una interfaz muy sencilla. Se trata únicamente de cuatro pines que deben ser conectados, uno de alimentación, uno de tierra, uno para disparar el pulso y uno para la recepción del eco.

En la Figura 3.16 pueden verse los pines y los elementos de disparo y recepción del ultrasonido.

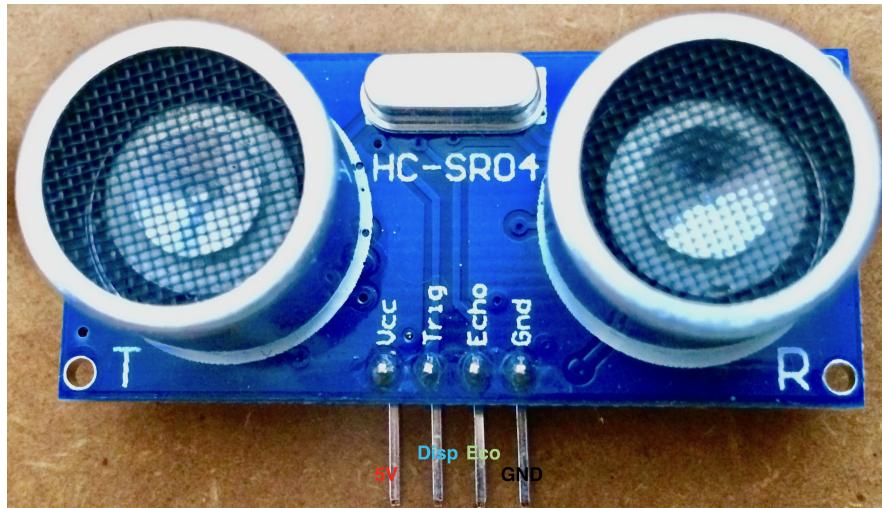


Figura 3.16: Sensor HC-SR04. Detalle de los pines y cabezales de envío y recepción de ultrasonido

### Desventajas

El uso de ultrasonidos para detectar objetos y distancias presenta una serie de inconvenientes que serán detallados en mayor profundidad en el capítulo 4.4. Por enunciar algunos,

- La velocidad del desplazamiento del sonido por el medio cambia con la temperatura. No es una variación muy grande, pero impide una precisión grande por parte de estos sensores.
- Los sensores de ultrasonidos son susceptibles a ecos provenientes de otros sensores, lo que dificulta mucho la posibilidad de disparar varios al mismo tiempo.

- El sonido puede verse absorbido por un obstáculo hecho de un material absorbente, falseando las medidas del sensor.
- El sensor tiene un rango de acción de  $30^\circ$ , lo cual dificulta su posicionamiento y detección de obstáculos de forma precisa, es decir, es fácil conocer la distancia a que se encuentra un obstáculo, pero no saber en qué posición del arco de  $30^\circ$  se encuentra exactamente.
- Este sensor funciona a 5V, y la RaspberryPi a 3.3V, con lo que debe crearse un mecanismo para reducir este voltaje para no dañar los pines del procesador de la Raspberry.

### 3.3. Protocolos

#### Pulse Width Modulation

La modulación por ancho de pulso<sup>14</sup> [15], se basa en medir el transcurso de tiempo entre el flanco de subida de una señal, y el flanco de bajada, tal y como puede verse en la Figura 3.17. En este contexto, un receptor de radio recibe una señal de la emisora, y genera una señal PWM acorde que será transmitida a la FC. Estas son capaces de entender señales de entre 1000 y 2000 $\mu$ s. Cualquier valor por debajo, o por encima, haría entrar la controladora en FailSafe<sup>15</sup>. En el caso de este proyecto, se ha determinado que no se hará uso de este método para la comunicación entre la Raspberry Pi y la controladora de vuelo, véase la subsección 3.3, y por lo tanto el uso de estos pines queda descartado.

Sin embargo, la comunicación entre la FC y los ESC se realiza mediante este tipo de señal, por ello se ha considerado relevante explicar, brevemente, su funcionamiento.

---

<sup>14</sup>PWM no es un protocolo como tal, sino una técnica de codificación de la información haciendo uso de una señal analógica. Se ha incluido en esta sección por simplicidad.

<sup>15</sup>Al recibir una señal inválida por parte del receptor, la controladora de vuelo puede ser configurada para desactivar el *drone*, mantener la última medida buena conocida, intentar aterrizar... etc. Este modo es conocido como FailSafe

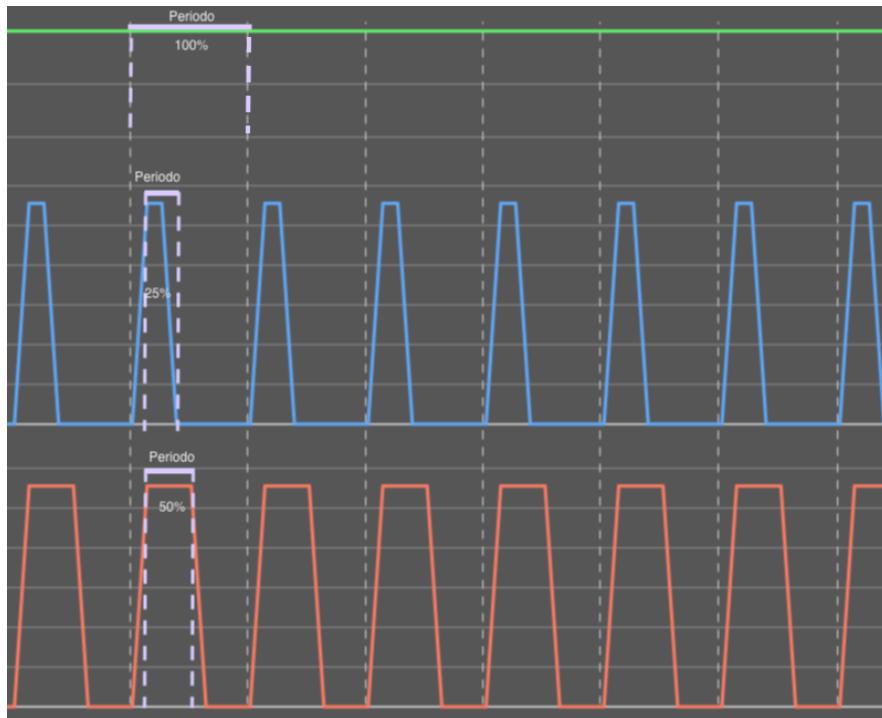


Figura 3.17: Modulación en Ancho de Pulso. Arriba 100 % del ciclo usado. Centro 25 % del ciclo usado. Abajo 50 % del ciclo usado

### MultiWii Serial Protocol

MultiWii es un software de control de multirotores y está basado en componentes de la Nintendo Wii. Concretamente, los mandos de control de la consola de Nintendo poseen tres acelerómetros para determinar la posición angular y medir aceleraciones laterales. El problema es que los acelerómetros no son precisos para variaciones pequeñas, de forma que Nintendo creó un complemento que se podía conectar al propio mando, el Wii Motion Plus, que dispone de tres giroscopios, de forma que unidos a los tres acelerómetros iniciales, proporcionan una medición mucho más precisa de la posición del mando.

A los primeros creadores del sistema de control MultiWii se les ocurrió la posibilidad de hacer uso de estos sensores para crear un sistema de control de drones, [16]. Uniendo estos sensores a un controlador, en principio se trató de un Arduino Mini, y programándolo para tal efecto, se logra crear una FC algo rudimentaria, pero que da muy buenos resultados.

Para crear un entorno amigable para el usuario común, se desarrolló una aplicación de escritorio capaz de configurar los parámetros de esta controladora de vuelo poco convencional. De alguna forma debía lograrse una comunicación

entre la FC y la aplicación, y así se implementó MultiWii Serial Protocol, [17]. Conocido como *MSP*, se trata de un protocolo que se ha mantenido en diferentes implementaciones de sistemas de control de vuelo. No solo aquellos basados en sensores de la Nintendo Wii controlados por Arduino, sino en otros más modernos y potentes que han ido surgiendo los últimos años. Como el utilizado en nuestro proyecto.

Es un protocolo eficiente y sencillo, que permite una comunicación rápida y completa. Su composición puede verse en la Figura 3.18

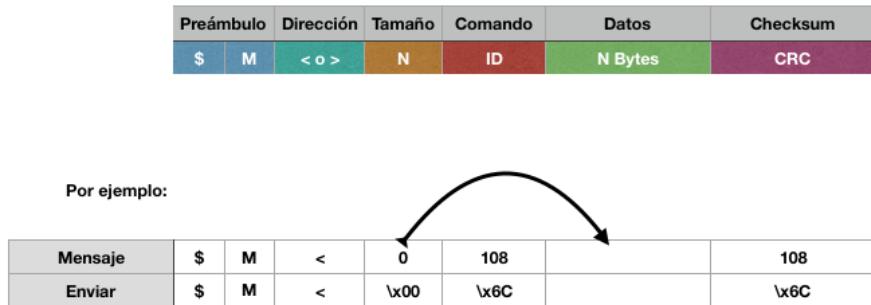


Figura 3.18: Composición de un mensaje MSP.

- Preámbulo: Se trata de dos caracteres ASCII que determinan el comienzo de un nuevo mensaje. Se compone de un símbolo ‘\$’ y una letra ‘M’.
- Dirección: Se trata de un carácter ASCII que determina la dirección del mensaje, **hacia** la controladora ‘<’ o **desde** la controladora ‘>’.
- Tamaño: Se trata de un byte que determina el tamaño, en bytes, de los datos enviados o recibidos.
- Comando: Se trata de un byte que determina el comando a ejecutar. Véase la tabla 3.3 para una relación de los comandos implementados.
- Datos: Se trata de una serie de bytes, de longitud definida por el byte  $<tamaño>$ , que establecen el resto de parámetros a pasar con la función definida por el byte  $<comando>$ .
- *Checksum*: Se trata de un código de detección de errores. Se define su valor mediante la función XOR entre  $<tamaño>$ ,  $<comando>$  y cada byte contenido en  $<datos>$

En el caso de nuestro proyecto, se hará uso de este protocolo dado que existe la posibilidad de establecer la recepción de los canales de radio a través de un puerto serie, así como de solicitar información sobre el estado del *drone* a

la FC. Es decir, en lugar de utilizar un receptor de radio, se utilizará un puerto serie para obtener la telemetría<sup>16</sup>, y establecer las entradas de los canales de radio. En la tabla 3.3 se dispone de los mensajes implementados para la realización de este proyecto.

La información del *parser* proporciona la manera de decodificar la cadena de bytes devuelta por la FC al realizar una solicitud. Para ello se ha seguido la nomenclatura utilizada en el módulo *struct* de Python 3.5 [18], y descrita en la tabla 3.2

Elemento	Descripción
'<', '>'	Little o Big Endian respectivamente. Establece la ubicación del byte menos significativo, y por lo tanto determina la dirección de lectura de los grupos de bytes de la cadena recibida.
'B'	Carácter utilizado para representar un entero de un byte sin signo.
'H'	Carácter utilizado para representar un entero de dos bytes sin signo.
'c'	Carácter utilizado para representar un carácter de un byte.

Tabla 3.2: Nomenclatura del módulo Struct de la implementación 3.5 de Python

Los enteros sin signo representados en la tabla 3.2, tienen su equivalencia a enteros con signo mediante el mismo carácter en minúscula.

La implementación realizada sigue un paradigma funcional, de forma que la función utilizada para leer las respuestas de la FC puede ser utilizada con nuevas solicitudes de información, con tan solo pasar un nuevo parser en forma de cadena de texto.

## Secure SHell

Secure SHell o *SSH* [19] de aquí en adelante, es un protocolo de red cifrado, el cual se basa en el uso de claves públicas compartidas para crear un canal de comunicación seguro en una red no segura. Al aceptar una conexión, el servicio SSH presenta una shell sobre la que el cliente puede realizar las operaciones necesarias y propias de su nivel de privilegio. Además proporciona la posibilidad de redirigir el sistema de ventanas remoto al cliente, aunque en este caso se está haciendo uso de una versión reducida del SO, y por lo tanto no presenta

---

<sup>16</sup>Se define telemetría como un sistema de medición de magnitudes a distancia; en este contexto el término se ha desvirtuado y se entiende como la información que es transmitida de vuelta a la emisora de vuelo, siendo esta generalmente información de los sensores de la FC, GPS o voltaje disponible en la batería.

Mensaje	Código	Estructura	Parse	Descripción
MSP_MOTOR	104	8x UINT16	<8H	Devuelve la señal PWM que los ESC envían a los motores.
MSP_RC	105	18x UINT16	<18H	Devuelve la señal PWM disponible en cada canal.
MSP_ATTITUDE	108	3x INT16	<3h	Devuelve las inclinaciones de los ejes X e Y, así como la orientación.
MSP_ANALOG	110	1x UINT8 3x UINT16	<B3H	Devuelve el estado de los sensores incluidos en la FC: voltaje de la batería, RSSI, corriente instantánea y consumo
MSP_SET_RAW_RC	200	Nx UINT16	N/A	Establece el valor de los canales de radiocontrol.
MSP_SET_RAW_MOTOR	214	Nx UINT16	N/A	Establece la señal PWM a enviar a los motores.

Tabla 3.3: Mensajes MSP. Estructura de datos y representación

La columna *Mensaje* sigue esta nomenclatura por razones históricas, para mantener cierta coherencia con la tabla disponible en la descripción del protocolo MultiWii [17]. La columna *Código* establece el número de comando enviado. Los comandos que empiezan por 1, se corresponden con solicitudes de información, y aquellos que comienzan por 2 son órdenes. Este valor será utilizado por la FC para establecer el parseo de la información enviada, de ser alguna. La columna *Estructura* establece el tipo y la cantidad de cada dato que se envía o recibe. La columna *Parse* establece el tipo de parser que se aplicará a la respuesta recibida de la FC. Nótese que solo se define el *parser* para los comandos que comienzan por 1, es decir, para las solicitudes de información.

entorno de escritorio. El acceso a la Raspberry Pi que controla el *drone*, se lleva a cabo mediante el uso de este protocolo, siguiendo los siguientes pasos para su configuración:

- Generar el par de claves pública-privada en el cliente mediante el comando `ssh-keygen`
- Copiar la clave pública del cliente en el archivo `.ssh/authorized_keys` de la carpeta `home` del usuario a utilizar en el sistema remoto.
- Editar el archivo `/etc/ssh/sshd_config` para:
  - permitir únicamente acceso a usuarios que envíen una clave pública contenida en `authorized_keys`.
  - desactivar el acceso al usuario root, estableciendo la propiedad `PermitRootLogin` a `no`.
  - desactivar el acceso por contraseña, estableciendo la propiedad `PasswordAuthentication` a `no`.

De esta forma se logra dotar de acceso seguro a un cliente autorizado. Al tratar de conectar al sistema de control del *drone*, se muestra un banner en el que se advierte a usuarios malintencionados de la existencia de un log de conexiones recibidas. Para tratar de mitigar ciertos intentos de intrusión en el sistema, algo tan simple como cambiar el puerto en el que responde el servidor SSH, se ha mostrado tremadamente efectivo contra los ataques automatizados más simples y comunes.

## WebSocket

Se trata de un protocolo de comunicaciones *full-duplex* sobre una conexión TCP. Fueron estandarizados por el IETF como RFC 6455, [20, 21], en el año 2011.

La comunicación se realiza a través del puerto 80 identificando el recurso como `ws`, o del 443 si se utiliza una conexión cifrada identificando el recurso como `wss`, y es actualmente implementado por la mayoría de los navegadores.

En el caso de este proyecto se utilizarán WebSockets para realizar el proceso de *signaling* hacia el servidor WebRTC, ver subsección 3.3, que se estará ejecutando en la RaspberryPi, para que este inicie la transmisión de vídeo en tiempo real y así poder visualizarlo. Dicha conexión será cifrada mediante SSL/TLS haciendo uso del certificado generado y firmado por una CA<sup>17</sup> de confianza.

---

<sup>17</sup>Certificate Authority. Una entidad que proporciona certificados digitales.

Mediante WebSockets se establecerá el mecanismo para coordinar la comunicación y enviar mensajes de control al servidor de vídeo en tiempo real. Este proceso, denominado *signaling*, comporta tres tipos de información:

- Session Control Messages: Mensajes de Control de Sesión. Utilizados para iniciar o finalizar la comunicación, así como para reportar errores.
- Network Configuration: Configuración de Red. Utilizados para enviar la dirección IP y puerto de comunicación desde el cliente hacia el servidor.
- Media Capabilities: Capacidades de los dispositivos de Medios. Utilizados para establecer el tipo de *codec*, resolución, *bitrate*... etc.

Este intercambio de información deberá completarse de forma correcta antes de comenzar la transmisión entre el cliente y el servidor.

## WebRTC

Se trata de un proyecto de código abierto que proporciona comunicación en tiempo real mediante una API, para facilitar el intercambio de información entre pares, idealmente, aunque puede darse el caso de requerir el paso de esta información por un tercer servidor. Esta iniciativa involucra a Google, Mozilla y Opera entre otros, [22, 23].

Se apoya en ICE o Interactive Connectivity Establishment, [24] un framework utilizado para lograr que dos clientes se comuniquen entre sí, sin necesidad de un servidor intermedio que haga de negociador y de esta forma conseguir que la comunicación sea lo más ágil y rápida posible. Para lograrlo, pese al enmascaramiento de los clientes tras múltiples NAT<sup>18</sup>, hace uso de:

- Servidores STUN (*Session Traversal Utilities for NAT*), que se encuentran en el lado público de esta. Permiten detectar el uso de NAT, y encuentran el mapeo de IP y puerto realizado para la aplicación del cliente. De esta forma un cliente tras una NAT es capaz de conocer su IP pública y el puerto que utiliza para *salir* a internet, [25].
- Servidores TURN (*Traversal Using Relays around NAT*), como último recurso. En el caso de que no se pueda realizar la comunicación P2P,

---

<sup>18</sup> *Network Address Translation*. Con la masificación de internet se ha llegado a un punto en el que el protocolo IPv4 no dispone de más espacio de direccionamiento, es decir, no quedan direcciones IPv4. En lugar de migrar los diferentes elementos de una arquitectura de red (como switches, routers, servidores y demás) al protocolo de internet IPv6, que proporciona un espacio de direccionamiento *mucho* mayor ( $2^{128}$  direcciones), los proveedores de servicios de internet, crean redes de redes usando como *receptorista* el NAT. Lo cual dificulta *terriblemente* el encaminamiento de información entre pares. Sobre todo cuando crean NATs de dispositivos que usan NATs

el sistema pasa a utilizar un servidor que hace de repetidor, lo cual obviamente añade cierto retardo y supone una gran carga para estos. El protocolo TURN se encarga de proporcionar IP y puerto para obtener una conexión con estos servidores *repetidores* o *retransmisores*, [26].

El funcionamiento general de una comunicación entre pares (Alice y Bob, por supuesto) haciendo uso de la API WebRTC es como sigue:

- 1 Alice y Bob establecen comunicación haciendo uso del ICE, el cual trata de comunicarlos por un canal directo, con la menor latencia posible haciendo uso del protocolo UDP. Para ello es necesario utilizar servidores STUN. Google provee de un par de servidores STUN de uso libre. Si la comunicación por UDP falla, ICE trata de establecerla bajo TCP (primero HTTP y después HTTPS). Si la comunicación directa falla (generalmente debido a NAT transversales o firewalls), ICE hará uso de servidores TURN que harán de repetidores de los mensajes del cliente.
- 2 *Signaling*. La aplicación negocia el tipo de comunicación que espera, es decir, el codec de vídeo, la resolución, el *codec* de audio, *bitrate*... etc. Y establece los orígenes de vídeo/audio para ambos *peers*. En la documentación de WebRTC no se define la forma en que se debe transmitir la información, de manera que en este proyecto se ha escogido hacerlo a través de WebSockets, ver la subsección 3.3. La información a transmitir sigue el Session Description Protocol (SDP).
  - Alice ejecuta su lado constituyendo un *RTCPeerConnection*, con un manejador para el evento de llegada de un candidato a conexión.
  - Bob ejecuta su lado constituyendo un *RTCPeerConnection*, con un manejador para el evento de llegada de un candidato a conexión. La diferencia con Alice, es que es Bob quien trata de realizar la conexión mediante un protocolo arbitrario, WebSockets en este caso.
  - Cuando Alice recibe la conexión de Bob, esta añade a Bob como *ICECandidate* en su descripción de *peer* remoto. De igual manera actúa Bob en el momento en que establece comunicación con Alice. De esta forma, ambos establecen la descripción de la sesión.
- 3 Una vez que conocen la forma de comunicarse, Alice crea una oferta de vídeo que Bob recibe en un manejador, *onTrack* encargado de establecer la fuente de vídeo de la página que él visualiza al *stream* remoto que Alice ha ofrecido.
- 4 Bob inicia la *llamada* diciéndole a Alice que resolución quiere del vídeo.

El proceso completo viene detallado en el artículo [27]. En el caso de este proyecto, aquí termina el intercambio de streams. Alice es el servidor de vídeo y Bob es el cliente. La comunicación se establece como si fuesen pares, solo que Bob no llega a ofrecer nunca un stream de vídeo/audio, de forma que Alice nunca llega a establecer su fuente de vídeo local al stream remoto de Bob.

---

# Técnicas y herramientas

---

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Si se han estudiado diferentes alternativas de metodologías, herramientas, bibliotecas, se puede hacer un resumen de los aspectos más destacados de cada alternativa, incluyendo comparativas entre las distintas opciones y una justificación de las elecciones realizadas. No se pretende que este apartado se convierta en un capítulo de un libro dedicado a cada una de las alternativas, sino comentar los aspectos más destacados de cada opción, con un repaso somero a los fundamentos esenciales y referencias bibliográficas para que el lector pueda ampliar su conocimiento sobre el tema.

## 4.1. Metodologías de Desarrollo

### SCRUM

Scrum es un marco de desarrollo ágil que se caracteriza por realizar ciclos de desarrollo. Se basa en usar una estrategia incremental, frente al carácter más completo y planificado de las metodologías tradicionales, mediante iteraciones a las que denomina *sprint*.

En cada una de los sprint se revisa lo que se ha realizado durante la iteración anterior, y se determina que labores o tareas se pueden realizar en el nuevo ciclo. Esto aporta una gran cantidad de ventajas frente a los métodos tradicionales. [28].

### GitFlow

El flujo de trabajo de Git permite establecer una forma organizada y ágil de llevar a cabo todos las aportaciones del proyecto. [29].

Se basa en el uso de ramas para organizar el flujo de trabajo. La rama *master* contiene el estado actual del desarrollo estable, el cual está listo para ser desplegado en cualquier momento. Se suele crear una rama cuando se quiere añadir una nueva característica, o realizar una nueva versión del programa. En el caso de este proyecto se ha mantenido la rama master y se ha creado una rama para las nuevas versiones. Estas han sido unidas a la rama master una vez listas para su despliegue. Sin embargo, no se han creado ramas de la principal o de las versiones para corregir errores o incluir pequeñas modificaciones.

Dado que no se cuenta con un equipo de desarrollo completo, el flujo habitual de trabajo se ha desarrollado sobre la rama principal, master, con ramas de desarrollo a modo de *releases*.

## Kanban

Kanban es un sistema de organización y gestión de las tareas de un proyecto. [30] Viene del japonés *Kanban*, o *tarjeta de señal* en Castellano, llamado de esta manera por el uso que se hace de tarjetas que describen las tareas a realizar.

Su representación suele hacerse sobre una pizarra o panel en el que se van estableciendo las diferentes tareas a realizar en pequeñas tarjetas. Dichas tarjetas se organizan dentro del panel en diferentes zonas, como *en progreso* o *nueva tarea o finalizada*. De esta forma es sencillo comprobar el estado del desarrollo de un vistazo.

## 4.2. Herramientas de gestión de repositorio

### Git

Git es una herramienta utilizada para llevar a cabo control de versiones. Es posiblemente la más utilizada entre empresas y desarrolladores, y viene integrada en la mayoría de los sistemas basados en UNIX.

Para llevar a cabo la gestión de las versiones, establece un repositorio del cual se guarda un histórico de todas las modificaciones que se han llevado a cabo, permitiendo establecer quien hizo que modificación, así como hacer uso del sistema de ramas mencionado en 4.1.

### GitHub

GitHub es un repositorio de código remoto. Es posiblemente el servicio de hospedaje de código más extendido entre empresas y desarrolladores independientes [31].

Además de ser totalmente compatible con el sistema de ramas establecido en GitFlow 4.1, permite organizar el flujo de trabajo en forma de pequeñas tareas, que son fácilmente distribuibles en los diferentes sprints, ver 4.1.

De esta forma, unido a Git, se convierte en una herramienta de gran utilidad para llevar a cabo la correcta organización de un repositorio de código. Sobre todo si interviene más de un desarrollador.

### GitLab

En un principio, se valoró la posibilidad de hospedar un pequeño servidor GitLab que permitiese llevar a cabo el proyecto de forma privada de forma gratuita. Sin embargo, GitHub proporciona repositorios privados de forma gratuita a estudiantes, de forma que al final se eligió esta opción sobre GitLab.

### ZenHub

ZenHub es una extensión que se integra con GitHub. Se trata de una implementación del método Kanban, 4.1. [32].

Muestra en el repositorio de GitHub, un panel informativo organizado por columnas que describen el estado de las tareas que en ellas se encuentran. Permite visualizar rápidamente el estado del Sprint en el que se está trabajando, así como de las tareas que han quedado en espera.

### GitKraken

GitKraken es una aplicación de escritorio que permite hacer uso de Git desde una interfaz gráfica, [33]. Presenta el flujo de trabajo que se ha ido realizando, las diferentes ramas y *commits*, y permite realizar diferentes acciones relacionadas con Git, como *pull-request*, *commits*, creación y borrado de ramas, unión de ramas... etc.

Además provee de un sistema de resolución de conflictos entre archivos, que permite seleccionar los cambios a mantener de forma muy intuitiva y sencilla. La versión más reciente añade una implementación de Kanban que permite visualizar el mismo panel que muestra ZenHub en GitHub.

### GitHubDesktop

Se valoró la posibilidad de utilizar GitHubDesktop, pero GitKraken es superior en algunos sentidos, ya que permite realizar ciertas tareas complejas con extrema facilidad, como deshacer errores o guardar cambios para más adelante (*stash*), completamente integrado con GitFlow... etc.

### 4.3. Herramientas de desarrollo

#### PyCharm

PyCharm es un IDE, *Integrated Development Environment*, para el lenguaje de programación Python desarrollado por JetBrains, [34].

Se trata de un IDE que contiene todas las funcionalidades necesarias para el desarrollo de aplicaciones en Python, así como para la gestión de instalación de dependencias (vía `pip`, ver [35], y la adquisición de documentación necesaria para el desarrollo.

Es posiblemente el entorno de desarrollo más avanzado para Python. Se ha elegido porque permite hacer despliegue y *debug* de aplicaciones en remoto, de forma que es de gran utilidad al realizar tareas de *debug* en una RaspberryPi.

Cabe destacar que se está haciendo uso de una licencia de estudiante para acceder a todas las funcionalidades de PyCharm. Se puede obtener esta licencia a través de la página web, [34], creando una cuenta en la que se haga uso de un correo electrónico perteneciente a una institución educativa.

#### WebStorm

WebStorm es un IDE, *Integrated Development Environment*, para el desarrollo de aplicaciones web desarrollado por JetBrains, [36].

Se trata de un IDE que contiene todas las funcionalidades necesarias para el desarrollo de aplicaciones en HTML, JavaScript, Node.js, Angular, Electron... etc, así como para la gestión de instalación de dependencias (vía npm o Yarn), y la adquisición de documentación necesaria para el desarrollo.

Se trata de un entorno de desarrollo parecido a PyCharm, y su elección está un tanto condicionada por ello. Se ha utilizado para la programación del entorno web del que dispondrá el proyecto.

### 4.4. Herramientas de documentación

#### L<sup>A</sup>T<sub>E</sub>X

Para llevar a cabo la documentación del proyecto se ha hecho uso de L<sup>A</sup>T<sub>E</sub>X [37]. Se trata de un lenguaje de marcas que permite la redacción de textos que presentan alta calidad tipográfica. La filosofía de trabajo con L<sup>A</sup>T<sub>E</sub>X se basa en centrarse en el contenido y no en la forma. Es decir, el redactor de un documento no tiene porque centrarse en el formato, tipos de letra y demás, sino que su labor es redactar y por tanto se le deja dedicarse al contenido del mismo.

### TexMaker y TexLive

TexMaker es un editor multiplataforma de L<sup>A</sup>T<sub>E</sub>X. Se trata de un entorno parecido al habitual procesador de textos, que permite la redacción de textos haciendo uso del lenguaje de marcas T<sub>E</sub>X. Se caracteriza por implementar un corrector, disponer de auto-completado de marcas para L<sup>A</sup>T<sub>E</sub>X y un visor PDF del documento generado, [38].

TexLive es una distribución de L<sup>A</sup>T<sub>E</sub>X. Se trata de un compendio de herramientas, fuentes y archivos de configuración que permiten la compilación de código T<sub>E</sub>X a un documento legible, como un PDF, [39].

---

# **Aspectos relevantes del desarrollo del proyecto**

---

En este capítulo, se recogerán los aspectos que han determinado el desarrollo del proyecto. Tanto las ideas iniciales, como las finalmente implementadas, así como la motivación para tomar ciertas decisiones, y la manera en que se solucionaron los errores encontrados.

## **5.1. Inicio del proyecto**

La idea de este proyecto surgió al plantearme si quería pasar 12 créditos (300h) desarrollando algo que no me interesase ni motivase demasiado. Al fin y al cabo, 4 años de carrera deberían de acabar bien y, tal y como empezó, aprendiendo algo.

Así que se me ocurrió dar salida al *drone* que construí hace casi 4 años, haciéndolo formar parte de mi proyecto de final de carrera. En principio mi idea era que el *drone* recorriese un plano de un entorno abierto haciendo una ruta preestablecida. Grandes empresas del sector tecnológico (Amazon sobre todo) están jugueteando con la misma idea pero, generalmente por legislación, este proceso se está viendo muy lastrado.

Me gustaba la idea. ¿Qué mejor forma de finalizar una carrera de ingeniería que hacer que un *drone* se estrelle contra un obstáculo a alta velocidad, entre un amalgama de hélices y piezas hechas añicos? Al fin y al cabo, las pruebas son pruebas, y algo se iba a romper sí o sí. Mejor si era de una forma espectacular.

El Dr. César Ignacio García Osorio, uno de mis tutores, no parecía tan seguro, ya habían jugueteado con algo parecido en el pasado (en un simulador), y no le apetecía pasar de nuevo por lo mismo. El mismo Dr. César Ignacio García Osorio sugirió la posibilidad de que el *drone* actuase como un vigilante

nocturno en la universidad, y el Dr. José Francisco Díez Pastor añadió que podría hacer uso de un algoritmo de localización sin GPS.

Además, nos harían falta conocimientos fuertes sobre hardware y mecanismos de control, y un tercer tutor siempre viene bien, entra el Dr. Alejandro Merino Gómez.

Y así comencé con el desarrollo de este proyecto que, en algún momento, tendrá el potencial suficiente como para convertirse en el *sereno* de la Escuela Politécnica Superior de la Universidad de Burgos.

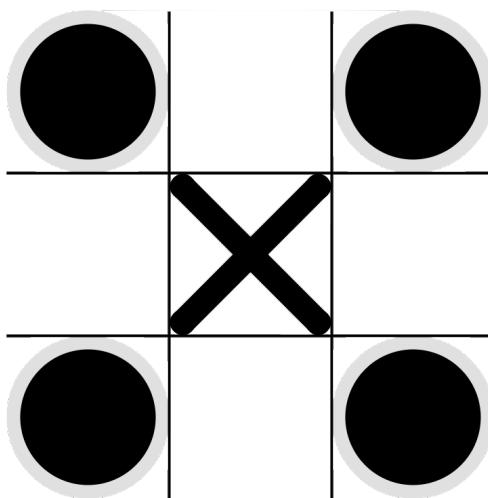


Figura 5.19: Logo de UBUDroneSereno.

## 5.2. Metodología

La gestión del proyecto se ha llevado a cabo a base de metodologías ágiles mediante *Scrum*. Sin embargo, cabe destacar que dado que el proyecto no se compone de varios integrantes, no es posible realizar en su totalidad el flujo de trabajo de Scrum, o este ha sido simplificado.

- Se han realizado iteraciones *Sprint* bisemanales, en las que se han discutido las tareas a completar de cara a la siguiente reunión.
- Dichas tareas, denominadas *Issues* han sido registradas haciendo uso de GitHub, que ha servido de plataforma de gestión de tareas y asignaciones. De nuevo, las asignaciones han sido realizadas por un único alumno.
- Se ha hecho uso de *Kanban* mediante la implementación que ofrece ZenHub de un tablero para tarjetas, y de la herramienta Glo, disponible en GitKraken.

- El flujo de trabajo se ha distribuido mediante Kanban, en diferentes pilas (En progreso, Cerrada, Backlog, IceBox... etc).
- Para comprobar el correcto ritmo de desarrollo del proyecto se ha hecho uso de los gráficos BurnDown que provee ZenHub, pero de nuevo, al no tratarse de un proyecto en el que intervienen múltiples desarrolladores, no es posible ajustarse al modelo presentado en los gráficos como ideal.

Para el desarrollo de todas las partes del proyecto se ha seguido el mismo proceso:

1. Se recaba toda la información necesaria para llevar a cabo la implementación.
2. Se estudia en profundidad la documentación.
3. Se comienza creando las clases de la implementación con una estrategia de diseño *bottom-up*, aunque las clases de más alto nivel siguen una estrategia *top-down* de forma que se requiere de una visión completa del proyecto para poder enlazarlas correctamente.

No se ha seguido un diseño dirigido por pruebas, *TDD*, ya que se ha considerado que genera una cantidad de código a refactorizar demasiado grande durante las etapas más tempranas, y dada la magnitud del proyecto solo conseguiría ralentizar el proceso de desarrollo. Se detallará más sobre las pruebas en la sección [5.8](#).

### 5.3. Formación

Para el desarrollo del proyecto se ha hecho uso de conocimientos adquiridos en la presente carrera, en muchos años de curiosidad y de búsqueda en internet.

El desarrollo ha sido, en su mayor parte, escrito en Python, y se ha hecho uso de librerías como Numpy, Matplotlib y SciPy que se estudian durante la carrera, y otras que no, como PySerial (para establecer una comunicación serie), Bluetin-Echo (para hacer uso de los sensores de distancia HC-SR04), el framework Flask (para el desarrollo de la web-app) y la API WebRTC (para el vídeo en tiempo real) entre otras.

Sin embargo, cabe destacar la lectura de ciertos artículos, y el acceso a ciertos recursos:

- *Artificial Intelligence for Robotics*. Un curso de Udacity impartido por Sebastian Thurn, disponible en [\[40\]](#). En él se explican componentes

habituales en múltiples proyectos, como el controlador PID, el filtro de Kalman, y otros no tan habituales, como el FIltro de Partículas utilizado en este proyecto para lograr un sistema de localización que no dependa de GPS, ver sección 3.1.

- MultiWiiSerialProtocol. Interfaz o definición del protocolo de comunicación MultiWii, utilizado en prácticamente todas las controladoras de vuelo actuales, como base de comunicación con ordenadores. Este protocolo es el utilizado para comunicar la RaspberryPi con la controladora de vuelo, y enviar/recibir comandos. Disponible en [17].
- *Getting Started with WebRTC*, un compendio de las funcionalidades que ofrece WebRTC para lograr comunicación en tiempo real. Utilizado para lograr vídeo sin apenas latencia entre un navegador y la RaspberryPi a bordo del *drone*. Disponible en [27].
- *The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots*. Se trata del artículo original de Borenstein y Koren, en el que detallan el funcionamiento del sistema de evasión de obstáculos implementado. Disponible en [8].
- *The Flask Megatutorial*. Miguel Grinberg da una clase magistral sobre como empezar, continuar y finalizar con Flask. Este mega-tutorial contiene todo lo necesario para preparar una web-app con Flask. Disponible en [41].

Por supuesto cabe destacar la ingente documentación disponible de Python, y todas las librerías que lo componen. Sin duda hay muchas otras fuentes de formación relacionada con este proyecto, y pueden ser consultadas en la bibliografía.

#### 5.4. Desarrollo de algoritmos

En el desarrollo de este proyecto, se ha llevado a cabo el desarrollo de tres algoritmos fundamentales:

- Un sistema de evasión de obstáculos. Llevado a cabo mediante la implementación de dos algoritmos: el algoritmo de Campos Potenciales, y el algoritmo VFH en sustitución del primero.
- Un sistema de localización sin depender de la red de satélites GPS/Glonass. Llevado a cabo mediante la implementación de un Filtro de Partículas basado en simulación de Montecarlo, para estimar la variable *posición*, en base a las variables *distancias a obstáculos*.

- Varios controladores automatizados para complementar la evasión de obstáculos, y lograr la autonomía del sistema. Llevados a cabo mediante implementaciones de controladores PID, alimentados por la salida del algoritmo VFH, y por las lecturas de los sensores de la controladora de vuelo (para la inclinación) y un sensor de ultrasonidos colocado para medir la altura del *drone*.

## Potential Fields

Durante el primer Sprint, y mientras decidíamos que algoritmos deberíamos usar para lograr el propósito del proyecto, el Dr. Alejandro Merino Gómez sugirió la utilización del algoritmo de Campos Potenciales para llevar a cabo la evasión de obstáculos.

El algoritmo de campos potenciales se basa en la idea de que existe una meta a la que se quiere llegar, y una serie de obstáculos.

La meta genera una fuerza de atracción sobre el *drone*, y los obstáculos fuerzas repulsoras. En primera instancia este algoritmo es relativamente sencillo de implementar, y no tiene un coste computacional muy alto. Toda la dimensionalidad del problema se acaba reduciendo a una suma de fuerzas, que determinan el vector resultante, con una dirección y un sentido que el *drone* deberá seguir.

Sin embargo, este método presenta ciertos inconvenientes:

- Si el *drone* entra en una zona con forma de ‘U’, podría atascarse en un mínimo local, entrando y saliendo de ella indefinidamente.
- Una zona estrecha es todo un desafío. Las paredes de un pasillo, por ejemplo, presentan fuerzas repulsoras sobre el *drone*, y la fuerza atrayente de la meta no es suficiente como para mantener la dirección del *drone* a través del pasillo, con lo que este empezará a oscilar (si es que ha conseguido entrar en él) o ni siquiera será capaz de pasar de la entrada del pasillo.

Por ello se ha hecho uso del Vector Field Histogram.

## Vector Field Histogram

El Vector Field Histogram se basa en el uso de una malla de certidumbre. Dicha malla es una representación del plano, el cual se divide en celdas (muy apropiado para usarlo con Numpy, y aprovechar la potencia del cálculo vectorizado de que provee).

Cada celda contiene un valor, dicho valor es la certeza de que en ella exista un obstáculo. Por tanto se define un tamaño para las celdas ( $5 \times 5$  cm es una

buenas medida teniendo en cuenta el rango de detección de nuestros sensores de ultrasonidos), y se realizan medidas mediante los sensores de ultrasonidos.

Encontrar un obstáculo en cierta posición supone incrementar en 1 la cuenta que existe en la celda que representa esa distancia del agente. Computacionalmente, este método supera al algoritmo de Campos Potenciales, ya que no es necesario tener en cuenta en qué posición del rayo que dispara el sensor (recordemos de la documentación teórica que el HC-SR04 tiene una amplitud de unos  $30^\circ$ ), creando una distribución Gaussiana alrededor de la bisectriz del rayo, sino que únicamente se incrementa la posición central.

Puede parecer una sobresimplificación del problema, pero lo cierto es que tras múltiples medidas de los sensores disponibles, la malla de certidumbre representa con bastante fidelidad el estado del entorno cercano al *drone*.

Una vez obtenida la malla, esta se convierte en un histograma polar. Para ello se divide esta en sectores de una amplitud determinada ( $5^\circ$  se ha mostrado como una cantidad aceptable, generando así 72 sectores que cubrirán los  $360^\circ$ ). De esta forma, ya se dispone de una representación aproximada de la ubicación de los obstáculos.

Aún así se somete el histograma a una función de suavizado. En el artículo presentado por Borenstein y Koren en [8], la función utilizada contiene una errata. Tras discutirlo con mis tutores, decidimos utilizar cualquiera de las funciones de suavizado o filtrado disponibles en SciPy, tal y como sugirió el Dr. José Francisco Díez Pastor, y me decanté por la señal de Hann. De esta forma, los obstáculos parecen *estirarse*, ya que la señal de Hann permite establecer la simetría de los valores a suavizar. Así que la certidumbre sobre la existencia de obstáculos en un punto contiguo a los obtenidos en la malla de certidumbre se verá incrementada, creando un histograma todavía más conservador sobre la posición de los obstáculos.

Por si esto pudiera parecer poco, se define un umbral. El umbral establece cuál es el nivel máximo de certidumbre de obstáculos en una zona, como para considerarla segura para navegar. El histograma presenta una serie de subidas y bajadas, como valles y colinas. Los valles, son precisamente las zonas que, umbral mediante, serán seguras para navegar.

Una vez obtenidos los valles (compuestos de varios sectores seguros), el algoritmo establecerá cuál es el sector elegido para dirigirse a él, en base a la ubicación de la meta a la que se quiere llegar. Precisamente en este paso se determina la fortaleza de este método frente a los Campos Potenciales. El VFH va a elegir siempre el centro del valle para navegar, de forma que puede navegar en zonas estrechas, siempre que el umbral lo permita.

Y precisamente en este paso es donde se encuentra otro problema en el artículo de Borenstein y Koren en [8], si siempre se elige el sector central para navegar, en el momento en el que se sale de una región estrecha (un pasillo

por ejemplo) y la meta no está en el centro, el *drone* empezará a ejecutar un movimiento circular aproximándose a la meta en un movimiento cada vez más cerrado pero, matemáticamente, puede que infinito. Por ello, se incluyó una modificación en el algoritmo original, que permite aproximarse al destino de forma directa, si este se encuentra a la vista del *drone*, esto es, en un sector seguro para su navegación.

La salida del algoritmo se ha codificado como la orientación que el *drone* debería seguir, es decir, presenta un valor en el intervalo  $[-179, 180]$ , el cual expresa cuantos grados debería rotar el *drone*, y dado el signo la dirección de rotación, para dirigirse a la meta a través de un sector seguro. Además, el VFH, proporciona una medida de la velocidad que debería alcanzar el *drone*, en base a como de ocupado está el sector en el que el *drone* navega. La velocidad será proporcionada en base a la certidumbre de obstáculos, de manera que se irá reduciendo para poder realizar un cambio de sentido.

Este valor, por si solo, no es de mucha utilidad ya que no es interpretable por la controladora de vuelo, sino que tiene que ser *convertido* a un valor entre  $[1000, 2000]\mu s$ . Para ello, se ha hecho uso de controladores PID, ver [5.4](#).

## Particle Filter

El Filtro de Partículas es una pieza clave en el funcionamiento del proyecto. Provee de información sobre la posición del agente de forma muy robusta, ya que está basado en un modelo probabilístico que obtiene sus valores mediante un método de Montecarlo.

Básicamente, el algoritmo crea *drones virtuales* (partículas) que obtienen medidas hasta los obstáculos. Conociendo esas medidas, y comparándolas con las del *drone* real, parece inmediato poder decir cuánto se parece un *drone virtual* al *drone* real, y establecer esa posición como la posición del *drone*.

Y por supuesto, en teoría casi todo es sencillo: los diferentes cursos, artículos, páginas web y tutoriales basados en teoría, siempre hablan de *landmarks*. Estos *landmarks*, son zonas reconocibles por el agente (una diana en una pared, una marca... etc), predisuestas sobre el entorno, de manera que tanto las partículas como el agente pueden medir sus distancias a ellos, y de esta forma calcular como de parecidas son las partículas al agente.

Sin embargo, a la hora de la práctica, no hay tales *landmarks*. Por el sencillo hecho de que si tu agente no tiene al alcance alguno de esos *landmarks*, no es nada probable que pueda medir su distancia a este. Pero sí tenemos paredes.

Así que lo que se hace es calcular las distancias teóricas de cada una de las partículas a los obstáculos dentro de su área de acción (basada en la distancia máxima alcanzable por los sensores HC-SR04), con la esperanza de que no

haya zonas simétricas ya que adquirirían la misma probabilidad de ser la zona en la que se encuentra el *drone*.

El *drone* por su parte toma medidas de las distancias, y se comparan las medidas reales con las de todas las partículas.

El *drone* se desplaza, y las partículas con él. Cada vez que el *drone* hace un movimiento las partículas deben ejecutar ese mismo movimiento. Sencillo. Sí, en teoría. El *drone* no gira o se desplaza la cantidad de grados o centímetros que se le ordena en cuanto se le ordena, el giro o el desplazamiento lleva tiempo. ¿Cuánto tiempo? No es nada sencillo de aproximar, depende del estado de la batería, de la temperatura (los ESC del *drone* pueden llegar a reducir la velocidad de giro del motor si notan demasiado calor, aunque no es habitual), del cálculo de los controladores PID... etc. Es decir, existe una incertidumbre MUY grande, sobre todo en el desplazamiento. El movimiento de rotación es más sencillo de aproximar dado que se dispone de un magnetómetro, aunque tampoco es fiable al 100% ya que puede estar alterado por la desviación magnética, diferente en cada punto del planeta, por la presencia de fuentes de magnetismo (motores, 4 en este caso), o fuentes de corriente elevadas (una batería alimentando 4 motores, por ejemplo).

De manera que se incluye cierta incertidumbre en cualquier movimiento realizado por las partículas, de esta forma la población de partículas va divergiendo, y alejándose poco a poco de la posición exacta del *drone*, en su mayoría.

Debido a ello, la población cada vez es menos útil al propósito de localización, así que se seleccionan individuos en base a la probabilidad de ser la posición real del agente. La selección se realiza con remplazo, es decir, una misma partícula puede aparecer varias veces. De esta forma la población vuelve a converger.

Y con esto ya se dispondría de una ubicación aproximada para el *drone*. Conocer la posición es fundamental para el funcionamiento del sistema de evasión de obstáculos, ver subsección 5.4, dado que el VFH trata de guiar el *drone* hacia la meta en una ruta segura, ¿cómo sabe hacia donde debe dirigir el *drone*, si no sabe dónde se encuentra?

De esto se puede extraer una lección de lo más filosófica: primero descubre donde estas, para luego poder dirigirte hacia tu meta.

## PID

Para llevar a cabo el control automatizado del *drone*, no es suficiente con proporcionar la salida de los diferentes algoritmos a la controladora de vuelo.

La controladora de vuelo entiende entradas en sus canales (Acelerador, Inclinación, Balanceo, Rotación, Armar/Desarmar) en el intervalo [1000, 2000] $\mu$ s,

de manera que recibiendo la información proveniente de algunos sensores, se han creado los siguientes controladores PID:

- Control de Altitud: Es fundamental disponer de un mecanismo de control constante de la altitud. Se alimenta de la información proveniente de un sensor de ultrasonidos que mira hacia el suelo, de forma que se puede establecer la altitud que debe alcanzar el *drone*, relativa al suelo sobre el que se encuentra. Se encarga del canal 1, o Acelerador.
- Control de Orientación: Este controlador se encarga de dirigir al *drone* por el entorno, mediante cambios en su orientación. Se alimenta de la salida del VFH, ver subsección 5.4, para establecer su objetivo, y del magnetómetro para establecer la medida real, y así crear una salida entre  $[1400, 1600]\mu s$ . Esta salida es debida a la posición central del canal,  $1500\mu s$ , estableciendo la *no* rotación del *drone* en ningún sentido. Se encarga del canal 4, o Rotación.
- Control de Velocidad: Este controlador se encarga de controlar la inclinación del *drone*. No es posible determinar una velocidad concreta para el desplazamiento, dado que es muy variable, pero si se puede establecer la inclinación que debe alcanzar el *drone*. Esta inclinación es generada en base a los obstáculos en el sector en que se navega, y provista por el VFH, ver subsección 5.4. De forma que tanto el objetivo, como el ajuste a realizar están automatizados. Genera una salida entre  $[1400, 1600]\mu s$ . La medida actual de la inclinación puede obtenerse de la controladora de vuelo. Se encarga del canal 2, o Elevación.

## 5.5. Comunicaciones

En el desarrollo de este proyecto, se ha llevado a cabo el desarrollo de tres sistemas de comunicación:

- Un sistema de comunicación entre un ordenador y un *drone*. Llevado a cabo mediante la librería de implementación propia *MSPio*.
- Un sistema de control remoto de un *drone* a través de un navegador web. Llevado a cabo mediante Flask, Python y la librería *MSPio*.
- Un sistema de vídeo en tiempo real. Llevado a cabo mediante la API de WebRTC, HTML, JavaScript, y en la parte del servidor (la RaspberryPi) se ha hecho uso de UV4L, un driver que accede a la cámara de la Raspberry, y provee de un servidor WebRTC al que conectarse, puede consultarse la web del proyecto en [42].

## MSPio

En este proyecto, la RaspberryPi no consta de un sistema de tiempo real y, por lo tanto, no es lo ideal para llevar a cabo el control de un sistema de tiempo real como lo es un *drone*. De esta manera, se ha relegado esta tarea en una controladora de vuelo.

Las controladoras de vuelo actuales, en su mayoría, implementan un mecanismo de comunicación con un ordenador para su configuración. Dicha configuración se realiza a través de una aplicación que muestra toda la parametrización que puede ser llevada a cabo (parámetros de PID, canales de radio, protocolos de radio, telemetría, alarmas, calibración... etc)

El protocolo que utilizan dichas controladoras de vuelo es generalmente el mismo: MultiWii Serial Protocol. Este protocolo se hereda de las primeras versiones de controladoras de vuelo, basadas en los mandos de la consola Wii de Nintendo<sup>19</sup>, y de ahí su nombre. La definición del protocolo es pública, así que cualquiera puede crear una implementación.

Para este proyecto, se ha creado una implementación para Python 3.x, a la que se ha llamado MSPio. Esta implementación es totalmente genérica, y permite el envío de órdenes a la controladora de vuelo, entre ellas la entrada de los canales que permiten mover el *drone*, así como la solicitud de información, como inclinación, estado de la batería, temperatura... etc.

Para mayor comodidad, se han creado varios métodos que permiten la obtención de información, o el envío de los canales de radio como una lista de valores, con tan solo llamarlos. Aún así, se ha determinado como *genérico*, ya que se ha definido el *parseo* de los valores de respuesta de la controladora de vuelo, como una cadena de texto. De forma que implementar la adquisición de nuevos datos es tan sencillo como pasar como parámetro una cadena de texto que representa la estructura de retorno. Ver tabla 3.3.

## Control Remoto

El mecanismo de control remoto se ha llevado a cabo mediante un navegador web, dado que proporciona un acceso sencillo y sin barreras a cualquier usuario.

Tan sencillo como iniciar sesión, el sistema establece que *drone* tiene asignado el usuario, y le conecta con él. La interfaz web permite la visualización en tiempo real de la emisión de vídeo disponible en el *drone*, a través de WebRTC, y su control con tan solo conectar una emisora al puerto USB del ordenador.

Una vez conectada la emisora, la web reconoce la misma como un joystick, y es capaz de leer los canales, y enviarlos al servidor remoto, el cual los convierte

---

<sup>19</sup>Wii y Nintendo son marcas registradas de Nintendo Co. LTD

en valores aceptables para el *drone*, y se los envía a este a través de una comunicación socket. Para activar el control remoto basta con activarlo y el *drone* ya estará listo para ser controlado.

De esta manera, el usuario final no tiene una comunicación directa con el *drone*, sino que el servidor intermedio es quien se encarga de esta tarea.

Esto libera a la RaspberryPi de hospedar un servidor web y recibir clientes, así como del uso de un canal de comunicación potencialmente no seguro entre el cliente y el *drone*.

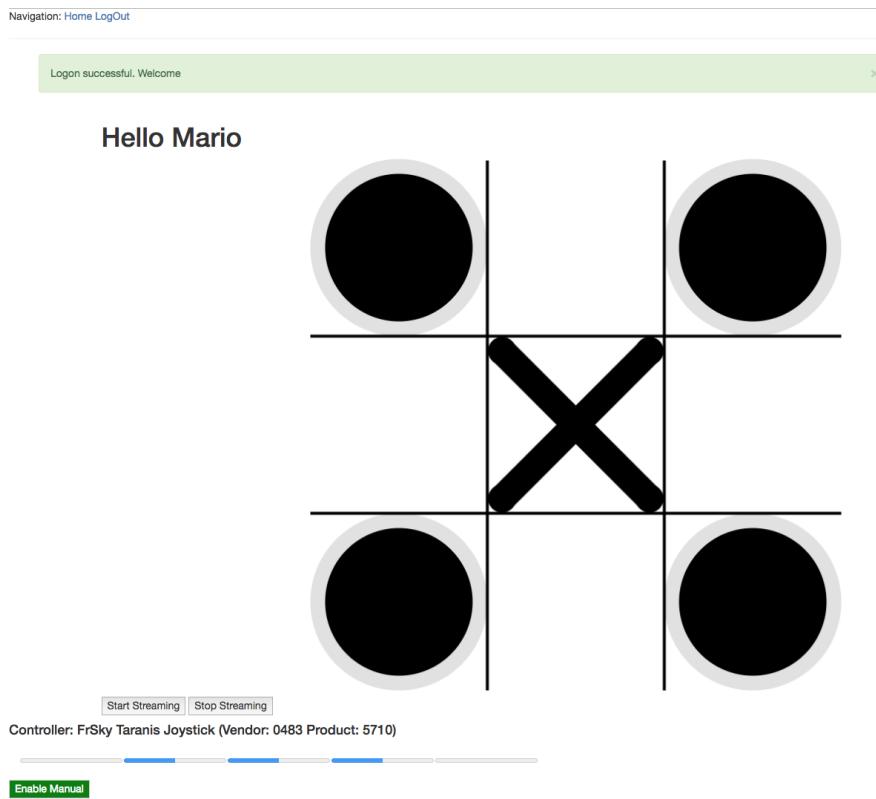


Figura 5.20: Página principal de la aplicación.

## Vídeo en Tiempo Real

Integrar vídeo en una página web puede parecer trivial, pero no lo es en absoluto. Los contenedores existentes en HTML requieren que el vídeo esté finalizado para poder reproducirlo (MP4/WebM deben definir la duración del vídeo), lo cual ha dado varios quebraderos de cabeza.

Se barajaron múltiples posibilidades, entre ellas:

- Hacer uso de MJPEG (Motion JPEG) consistente en tomar gran cantidad de fotografías e ir actualizando un canvas con cada nuevo fotograma. Sencillo, y simple de implementar, pero de un coste computacional inasumible dada la cantidad de trabajo que debe desempeñar la RaspberryPi.
- Hacer uso de MPEG-DASH (Dynamic Adaptive Streaming over HTTP), pero se mostró como una solución con un coste computacional elevado, dado que suponía instalar el servidor en la RaspberryPi, hacer que este realizase streaming y que el servidor principal actuase como un repetidor para los usuarios.

Finalmente nos decantamos por WebRTC, dado que permite establecer una comunicación directa entre dos navegadores, sin necesidad de hacer uso de HTTP. El problema que plantea es que se trata de una API para trabajar entre navegadores, y no es elegante ni fiable mantener una navegador constantemente abierto en la RaspberryPi, dado que requiere de intervención del usuario para permitir acceso a la cámara. La solución pasaba por utilizar, o implementar, un servidor WebRTC, que implementase el protocolo *Session Description Protocol* para comunicar un navegador cliente con la RaspberryPi. Por no añadir más carga de trabajo se ha hecho uso de UV4L, el cual es precisamente eso, un servidor que implementa las funcionalidades necesarias para hacer uso de WebRTC.

## 5.6. Plataforma de usuario

El manejo del *drone* por parte del usuario se ha diseñado para ser de tremenda simplicidad. Tan solo se muestra una página de login, tal y como se muestra en la Figura 5.21, que permite al usuario iniciar sesión.

Navigation: [Home](#) [LogIn](#)

# Sign in

Username

mbm0089

Password

.....

Remember me

Log in

Figura 5.21: Página de inicio de sesión para los usuarios.

Una vez registrado en el sistema, se le muestra una interfaz que permite iniciar o detener el streaming de vídeo por parte del *drone*. De esta forma, la RaspberryPi solo activa el servidor de vídeo cuando este se está visualizando, tal y como puede verse en la Figura 5.22.

Navigation: [Home](#) [LogOut](#)

Hello Mario



Figura 5.22: Página de del *drone* asignado para los usuarios. Los LED IR están activos dada la condición de baja luminosidad.

De conectar una emisora al equipo mientras el navegador está seleccionado, se reconoce ésta y permite controlar el *drone* con tan solo pulsar un botón, tal y como puede verse en la Figura 5.23.

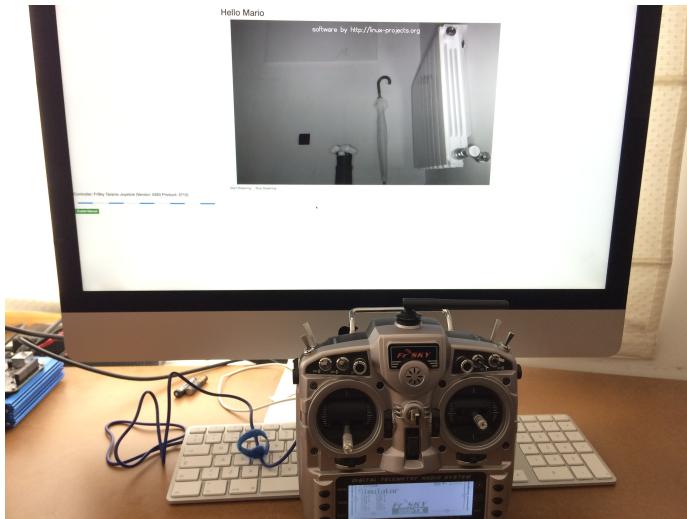


Figura 5.23: Página del *drone* asignado para los usuarios lista para activar el control.

La interfaz ha sido desarrollada en HTML5 y JavaScript, y la aplicación web se basa en el framework Flask, con la intención de aprender un nuevo medio de desarrollo de aplicaciones, y continuar trabajando con Python.

## 5.7. Hardware empleado

El equipamiento empleado en el *drone* es el siguiente:

- Un *drone* de 450mm de diagonal, distancia medida de motor a motor. Equipado con motores de 810KV (810rpm/V), controladores de velocidad (ESC) de 20A, una controladora de vuelo Flip32 con acelerómetro, giroscopio, magnetómetro y barómetro. Las hélices empleadas son de 9" de longitud y 4.5" de paso, generando un empuje máximo de 3.8Kg.
- Una batería LiPo de 3700mAh y 35C ( $C$  es la capacidad de descarga de la batería, en este caso sería  $35 \times 3700mAh = 129500mAh = 129,5A$  máximo).
- Seis sensores de ultrasonido HC-SR04. Cinco para las distancias a obstáculos y uno para controlar la altitud.
- Una RaspberryPi 3B.

- Una cámara PiNoIR. Sin filtro de infrarrojos, lo que permite el uso de LEDs infrarrojos para visión nocturna.
- Dos anillos de LED infrarrojos para visión nocturna.
- Seis divisores de voltaje para reducir el voltaje de salida de los sensores, ya que la RaspberryPi funciona a 3.3V y los sensores a 5V.
- Cables para conectar todo.

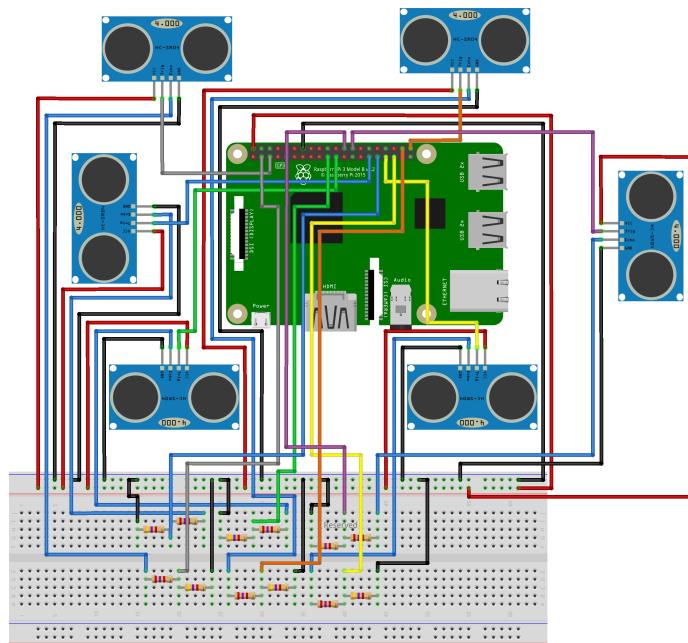


Figura 5.24: Vista conceptual del conexionado de los sensores a la RaspberryPi.

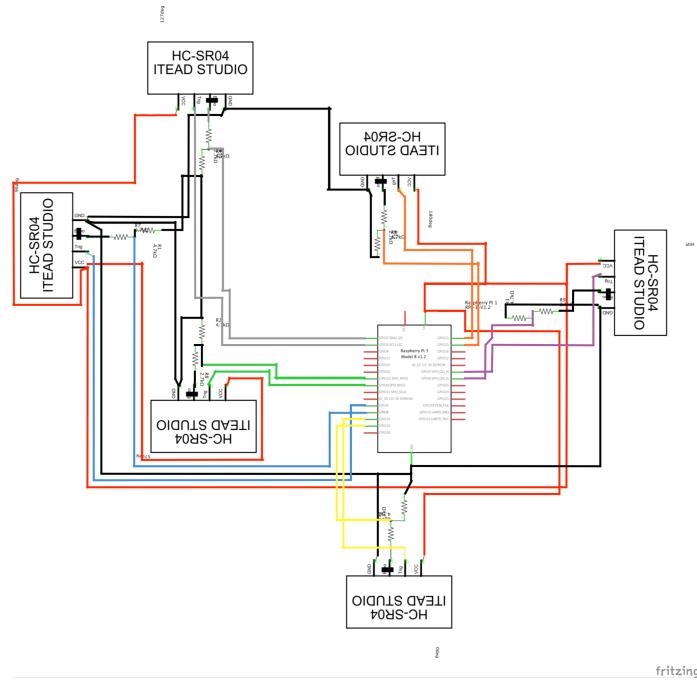


Figura 5.25: Vista esquemática del conexionado de los sensores a la Raspberry-Pi.

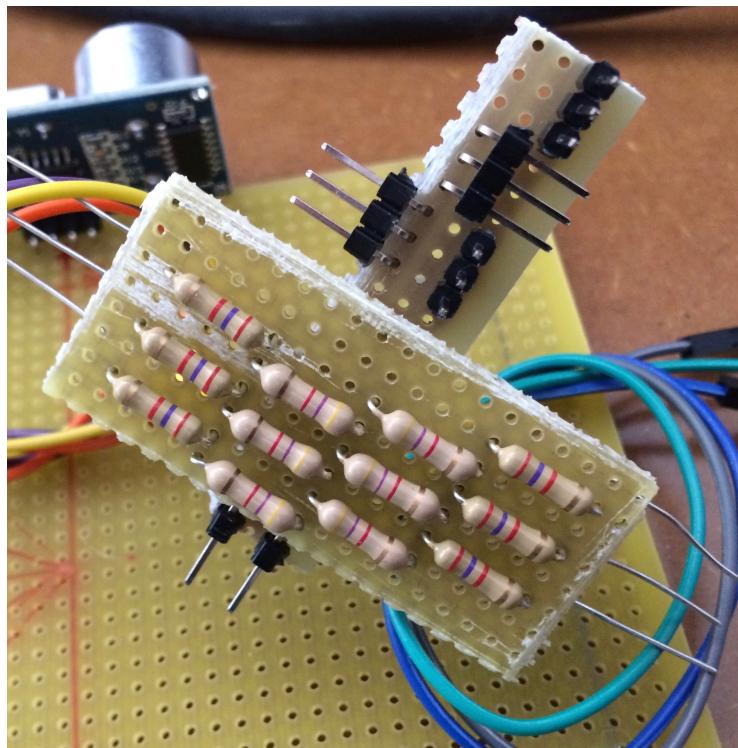


Figura 5.26: Vista imagen de los divisores de voltaje creados.

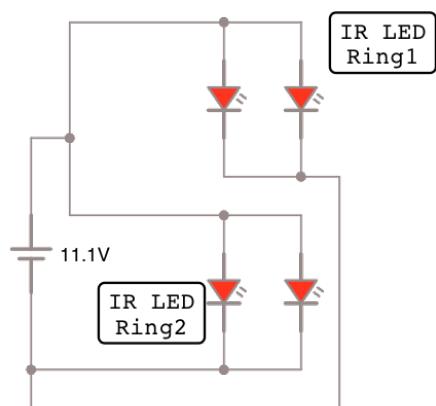


Figura 5.27: Vista esquemática del conexionado de los LED Infrarrojos.

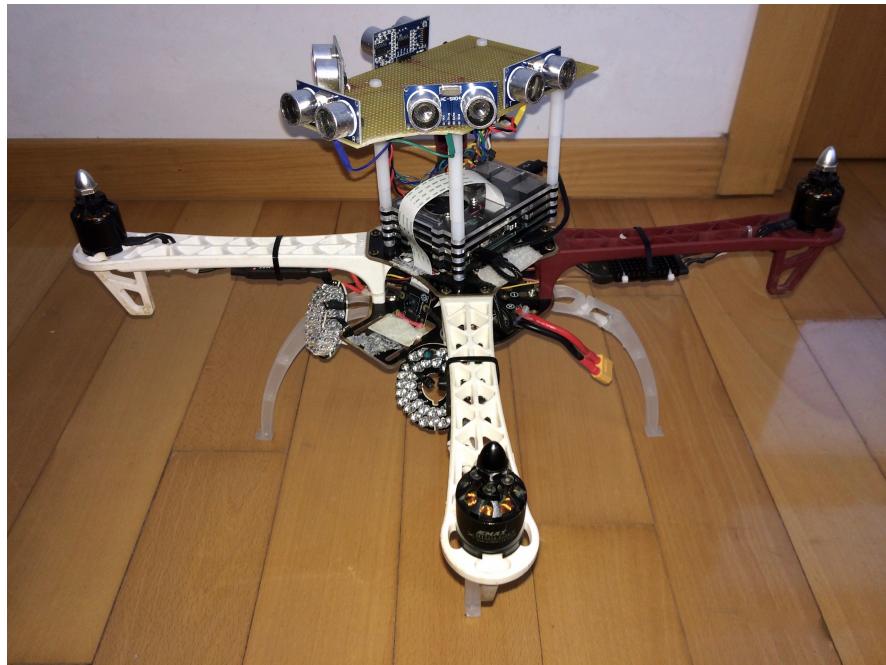


Figura 5.28: Detalle del *drone* completo. Lateral.

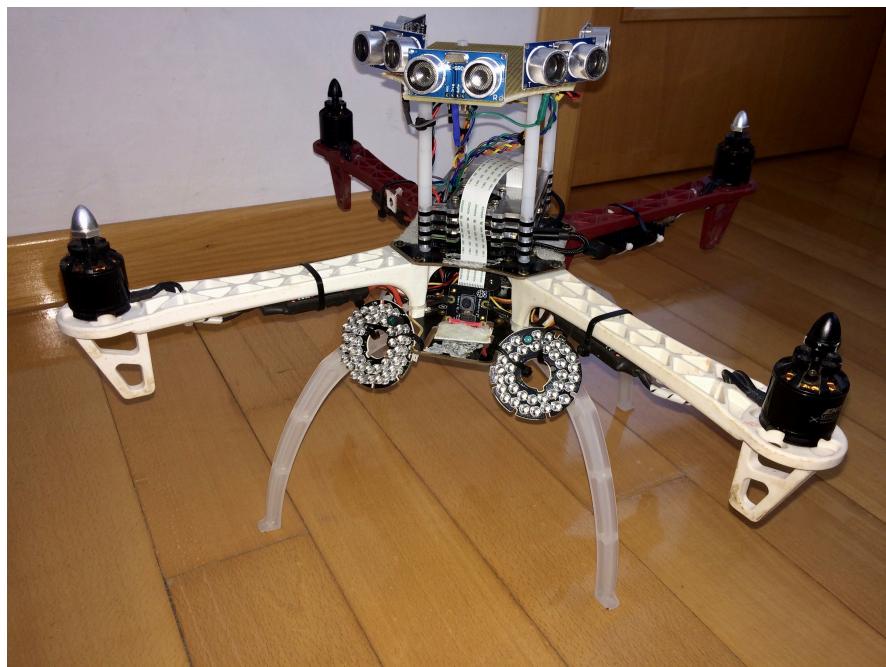


Figura 5.29: Detalle del *drone* completo. Frontal.

## 5.8. Pruebas

Sin duda la parte más relevante del proyecto es la implementación física del mismo. Las pruebas llevadas a cabo se han centrado en la componente práctica, y en pruebas de campo.

Las diferentes clases desarrolladas han sido testadas de forma interna, y no se dispone de pruebas como tal, que puedan ser desplegadas en un sistema de integración continua. La motivación de no realizar este tipo de pruebas es la siguiente:

- Las pruebas relacionadas con la comunicación entre el *drone* y la RaspberryPi requieren de una controladora de vuelo, y hacer un *mock* de este componente, supondría tener que estudiar la implementación completa que ofrece una controladora de vuelo al protocolo MultiWiiSerialProtocol, así como de todos los sensores. De no hacerlo, no existen sistemas de integración que pongan a nuestra disposición una controladora, un equipo que abra un puerto serie y que ejecute una serie de comandos/solicitudes interpretables por dicha controladora. Se incluye una pequeña clase de prueba que permite comprobar si se adquieren correctamente los datos de la controladora de vuelo, dibujando un plot de los ejes de inclinación y orientación, tal y como puede verse en la Figura 5.30, y en el vídeo [V#1](#).

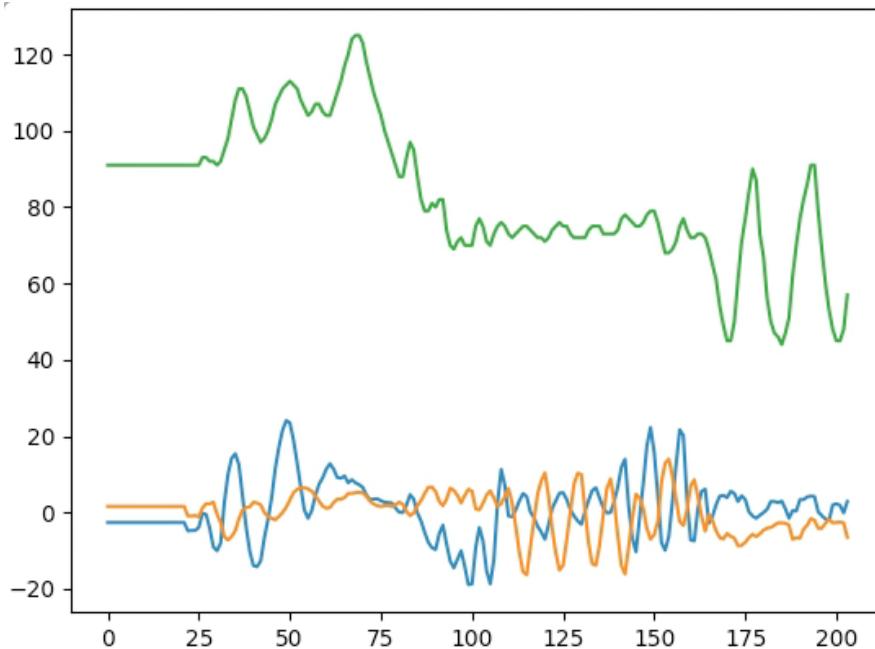


Figura 5.30: Salida de los sensores de la controladora de vuelo. En azul la elevación, en naranja el balanceo y en verde la orientación.

- De igual manera sucede con el sistema de control Remoto, que depende de la controladora de vuelo para comprobar su completo funcionamiento. De testearlo de forma parcial, solo se conseguiría comprobar que los paquetes llegan de forma correcta hasta el destino, y esta prueba se ha realizado como parte de las pruebas de campo, ver subsección 5.8, de forma satisfactoria, ya que ayudó a corregir un *bug* de difícil detección.
- Las pruebas relacionadas con el sistema de evasión de obstáculos, requieren de los diferentes sensores de distancia, de nuevo no existe ningún sistema de integración que proporcione dicho hardware. Realizar un mock de dichos sensores supone perder por completo la dinámica del sistema, i.e., los sensores no son perfectos, dan medidas con ruido, el *drone* no siempre se moverá con perfección milimétrica, los sensores no están perfectamente alineados... etc. Se realizaron pruebas durante su desarrollo mediante mock de los sensores, la cual puede ejecutarse si se utiliza la clase del VFH como `__main__`.
- Las pruebas relacionadas con el sistema de localización están basadas en los mismos sensores anteriores. El mock de dichos sensores para esta clase caería en una sobreimplementación del sistema, tal y como ocurre en el curso de UdaCity, ver [40], que solo daría una sensación de falsa funcionalidad del sistema. Realizar pruebas sobre el funcionamiento base del Filtro de

Partículas no es relevante, ya que no existe la dinámica encontrada en un entorno real. Pese a ello, en la clase que representa el Filtro de Partículas, se encuentra una pequeña prueba de su funcionamiento para un pequeño mapa en el que existen tres obstáculos. La dimensionalidad que alcanza el filtro es tal, que durante las pruebas no es nada sencillo apreciar el recorrido del algoritmo. De igual manera existe una pequeña prueba en la clase que representa los cálculos geométricos requeridos para el Filtro de Partículas.

## Pruebas de Campo

Las pruebas de campo han permitido demostrar el funcionamiento del *drone*, así como encontrar errores en la programación. Se han mostrado de gran utilidad, no solo en la búsqueda de errores, sino como fuente de información para futuras mejoras, así como para proyectos derivados o complementarios a este.

Esta sección se ha dividido en dos subsecciones, cada una establecida en base a la *release* a la que hacen referencia las pruebas.

### Primera Release: v0.1.0-alpha

Las primeras pruebas de campo llevadas a cabo, se realizaron en el polígono industrial de Villalonquejar, en Burgos. Dado que la primera *release* no incluye ningún elemento de automatización, sino que comprende el apartado relacionado con el control remoto del *drone*, pudieron ser llevadas a cabo sin demasiados inconvenientes en una zona apartada, pero no cerrada.



Figura 5.31: Drone despegando durante la primera prueba de campo.

La comunicación con el *drone* se estableció creando un punto de acceso desde un ordenador portátil. Seguidamente, se inició el servidor web en el portátil, el sistema de control del *drone*, y servidor WebRTC ambos en la RaspberryPi. El resultado de las pruebas puede verse en el vídeo [V#3](#).

Las pruebas comprenden los siguientes apartados, puntuados según la calidad del resultado en una escala de (0, 10]:

- Comprobación del correcto despliegue del sistema en una situación real: El despliegue se efectúa sin problema, la RaspberryPi responde correctamente a la comunicación vía SSH. Se realiza el despliegue perfectamente haciendo uso de la herramienta de *deployment* de que provee el IDE PyCharm. 10/10.
- Comprobación del sistema de vídeo en tiempo real: La comunicación con el servidor WebRTC es perfecta en este sentido. No existen retardos alarmantes en el feed de vídeo con una resolución de  $1280 \times 720$  a  $15\text{f/s}$ . Reducir la resolución y aumentar los fps podría ser una solución mejor, aunque el sistema no se moverá a una velocidad alta. La calidad del vídeo, sin embargo, es cuestionable. Al tratarse de una cámara sin filtro de luz infrarroja, los colores parecen algo saturados en ocasiones. 9/10.
- Comprobación del sistema de login, web y comunicación con el *drone*: El login se realiza de forma correcta, se reconoce el *drone* asignado al usuario y se activa la comunicación con este sin problema. 10/10.
- Comprobación del sistema de control remoto en tiempo real: El sistema de control remoto se activa sin problema, la comunicación es relativamente fluida, el *drone* responde correctamente a los comandos enviados, aunque con cierta latencia. Cabe destacar que los mensajes viajan hasta el servidor web que posteriormente se encarga de enviarlos al *drone*. Se encuentra un *bug*, referenciado en la *issue #53*, existente a la hora de *parsear* el paquete JSON recibido por el servidor de control remoto existente en el *drone*. La recreación del *bug*, puede verse en el vídeo [V#3](#) a partir del minuto 3:32. El *bug* detectado se debía a la debilidad de la señal al alejarse el *drone* del punto de acceso, ocasionando la pérdida de paquetes.

La realización de las pruebas ha dado como resultado la valoración de la release *v0.1.0-alpha* como lista para ser publicada. Sin embargo se mantiene como versión *alpha* dada la necesidad de realizar todavía más pruebas. Gracias al *bug* detectado, y solucionado, se valora incluir en la RaspberryPi una antena wifi mejor que la existente, o proporcionar a la RaspberryPi un modem 4G conectado vía USB para mantener siempre comunicación con el servidor. De esta forma se reducen las probabilidades de perder paquetes.

### Segunda Release: v0.2.0-alpha

La segunda release del proyecto, traerá las siguientes mejoras y características sobre la anterior versión:

- Grabación del feed de vídeo en el ordenador cliente, haciendo uso de WebRTC.
- Inclusión de diferentes sistemas de control automatizado, establecidos mediante un sistema de prioridades.
- Automatización del control de altitud.
- Automatización del control de velocidad.
- Automatización del control de dirección.
- Localización sin GPS.
- Evasión de obstáculos.
- Seguimiento de rutas.

Las pruebas realizadas comprenden los siguientes apartados, puntuados según la calidad del resultado en una escala de (0, 10]. Dada la peligrosidad de algunas de las pruebas, se ha llevado el *drone* a una nave industrial, ver agraciamientos en [5.9](#), dado que provee de un entorno seguro y cerrado en el que probar mecanismos automatizados:



Figura 5.32: Drone sobre superficie blanda durante el segundo field test

- Calibrado del sistema de control de altitud: Se trata de un controlador PID que debe ser parametrizado. Dada la dinámica del sistema, tan dependiente del estado de la batería, de los cambios de contexto realizados

por el sistema (recordemos que no es un RTOS<sup>20</sup>), de los errores de medida proporcionados por el sensor... etc, la parametrización del PID ha sido bastante compleja. De hecho durante el transcurso de las pruebas, se rompió una hélice y una de las patas que hacían de apoyo para el *drone*, dificultando esto último, todavía más, el proceso de estabilización durante el despegue.



Figura 5.33: Drone perdiendo una pata.

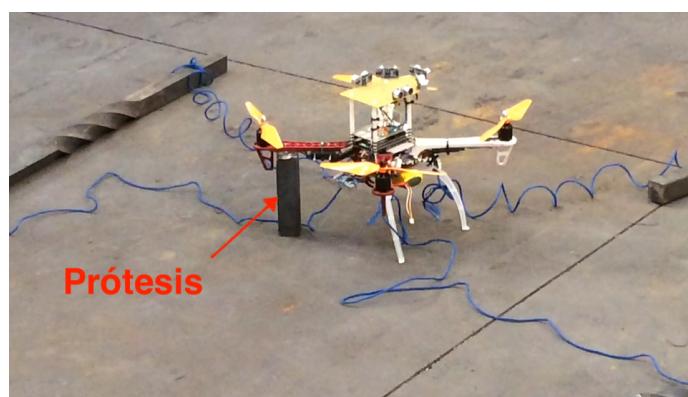


Figura 5.34: Drone con prótesis.

La motivación para retirar el material blando que puede verse en la

<sup>20</sup>Real Time Operative System

Figura 5.32, es que es tan absorbente que falseaba las medidas tomadas por el sensor de ultrasonidos.

En el vídeo V#4 puede verse el proceso llevado a cabo.

Se detectó un *bug* existente en el cálculo del PID que llevaba a la pérdida de precisión al realizar una conversión a entero, antes de realizar redondeo. Esto podía causar que el error acumulado se disparase, y por lo tanto dar a la componente Integral del sistema una importancia no deseada. Dado que se trata de un *bug* menor, se solucionó *in situ* por lo que no se generó una issue para ello.

Se detectó un *bug* existente en el sistema de control que no permitía que un controlador automatizado controlase cuando armar el *drone*. Esto produciría que el sistema de control de altitud, no pudiese actuar. Se trata de un *bug* menor, basado en añadir una pequeña espera en la que se mantiene la entrada del canal 5 (armar/desarmar) a un valor mínimo durante unos segundos, a la espera de que el sistema calibre el acelerómetro y el giróscopo. Solucionado *in situ*, no se ha generado una issue para ello. El resultado de las pruebas de altitud no ha sido el deseado. No se ha conseguido que el *drone* mantenga la altitud de forma correcta. Si que la alcanza sin problema, pero requiere de ajustar más los parámetros del controlador PID.



Figura 5.35: El *drone* alcanza con relativa suavidad la altura, pero no la mantiene correctamente.

Sin embargo ha arrojado claridad sobre ciertos problemas, y mejoras que se deberían probar. 5/10.

- Calibrado del sistema de control de velocidad: La velocidad a adquirir por el *drone* se basa en la inclinación del mismo. Se trata, de nuevo, de un controlador PID que debe ser parametrizado. Dada la imposibilidad de mantener la altura de forma correcta, y la indudable peligrosidad de este test, se ha preferido postergar la realización de estas pruebas hasta que se haya dado con una parametrización correcta del control de altitud. 0/10.
- Calibrado del sistema de control de dirección: La dirección a seguir se basa en la orientación del *drone*. Se trata, de nuevo, de un controlador PID que debe ser parametrizado. Dada la imposibilidad de mantener la altura de forma correcta, y la indudable peligrosidad de este test, se ha preferido postergar la realización de estas pruebas hasta que se haya dado con una parametrización correcta del control de altitud. 0/10.

La realización de las pruebas ha dado como resultado la valoración de la release *v0.2.0-alpha* como **no** lista para ser publicada. Sin lugar a dudas, requiere de muchas más pruebas en un entorno controlado antes de poder plantear la posibilidad de liberar esta versión.

Dados los resultados de las pruebas, convendría valorar los siguientes cambios en la estructura del *drone*:

- Reducir el peso total del dispositivo. Puede utilizarse una estructura de carbono, o similar.
- Mejorar el control de estabilidad del *drone*. Parece que la controladora de vuelo Flip32 no es lo ideal para este tipo de *drone* tan pesado y grande. Podría valorarse la posibilidad de cambiar a una controladora más moderna, y realizar una calibración de los PID mucho más acertada. De hecho la controladora de vuelo es, en términos tecnológicos, antigua.
- Cambiar los sensores de altitud, por uno con una tasa de adquisición mucho mayor. Este es uno de los principales problemas del cálculo del PID. El cuello de botella del sistema no está en el hecho de usar Python o de que no se trate de un sistema operativo de tiempo real, sino en la lentitud de los sensores. Los HC-SR04 requieren de un tiempo para tomar sus medidas demasiado elevado, unos  $0,06s$ , lo cual quiere decir que si el número de veces por segundo que se calcula el PID es muy superior al número de veces de que se dispone de información actualizada de los sensores, el cálculo del PID no es totalmente fiable, dado que trabaja con información antigua constantemente, y durante una cantidad de ciclos elevada. Por ello el uso de la componente integral queda muy limitado, ya que se dispara al no existir, aparentemente, cambios en los resultados provistos por el sensor. Podría limitarse la frecuencia del cálculo de PID, aunque esto tampoco es panacea.

- Para evitar ecos, el ‘disparo’ de ultrasonidos se realiza de forma secuencial, no enviando el siguiente disparo hasta que el anterior sensor haya recibido su medida. Dada la lentitud de estos sensores y que existen 5 de ellos en el cálculo de distancia a obstáculos, tomar una sola medida llevaría  $0,06s * 5 = 0,3s$ . Esta cantidad de tiempo, puede tornarse inaceptable en el momento en el que el sistema se desplace a una velocidad mayor, o requiera de mayor precisión. Por tanto se sugiere cambiar estos sensores por un sensor de distancia mucho más preciso y rápido, una solución basada en LIDAR debería ser valorada, o incluso en cámaras, dado el aumento de la capacidad de cómputo de las RaspberryPi.

## 5.9. Agradecimientos

Casi 90 páginas de memoria, sin contar anexos, y un montón de horas de lectura y búsqueda de información no podían pasar sin una, breve, sección de agradecimientos:

- En primer lugar a Laura, que me animó a comenzar esta carrera hace cuatro años, y que nunca ha dejado de animarme.
- A mi padre, que también es uno de mis mejores amigos, y ha sabido animarme (y aguantarme) en cualquier momento de frustración.
- A mis tres tutores Alejandro, Jose y César, que han gestionado este proyecto de forma impecable, y me han asesorado ante cualquier eventualidad. Me gustaría haber podido hacer más y dejarlo completado.
- A David, mi mejor amigo, que siempre me anima a inventarme cosas nuevas, y que siempre tiene unas palabras de ánimo para mí.
- A Isaac, otro gran amigo, y mi cámara durante la primera sesión de pruebas, ver subsección 5.8.
- A Cerrajería Artística Gutiérrez, por dejarme usar un espacio en su nave en Aranda de Duero para la segunda sesión de pruebas, ver subsección 5.8. De no ser por ellos, hoy seguiría tratando de bajar el *drone* de algún árbol.

---

# Trabajos relacionados

---

El desarrollo de este proyecto se encuentra en un ámbito bastante novedoso. La integración de *drones* es cada vez más amplia en múltiples sectores, sin embargo la navegación en entornos cerrados no es tan habitual en el sector civil.

## 6.1. Empresas

### TAISEI Co. LTD

Cabe destacar el uso que está dando TAISEI Co. LTD, una compañía Japonesa, dedicada a la seguridad y limpieza, a un *drone* de desarrollo propio, que pretende molestar a los trabajadores de una empresa para que no hagan horas extra. El artículo de El Mundo, disponible en [43], y la página de la compañía disponible en [44].

### Erle-Robotics

Erle Robotics es una compañía con sede en Vitoria, que se dedica a la creación de herramientas para desarrolladores de robótica. Entre sus productos se encuentra el Erle-Brain, el cual es precisamente una RaspberryPi que se encarga de controlar una controladora de vuelo, o el Erle-Copter, un *drone* con características muy parecidas al desarrollado. La página de la compañía está disponible en [45].

### FlyPulse

FlyPulse es una empresa basada en Suecia, y dedicada a la fabricación de *drones* con propósito de transporte de sistemas o elementos de asistencia médica. La página de la compañía está disponible en [46].

## TUDelft

TUDelft es una empresa Holandesa que se hizo famosa por el desarrollo de un *drone*-ambulancia capaz de transportar un desfibrilador de forma rápida y segura hasta su destino. La página del proyecto está disponible en [47].

## 6.2. Militar

Por falta de fuentes fiables, no se citarán en este proyecto implementaciones militares dadas a la navegación autónoma en entornos cerrados.

---

# Conclusiones y Líneas de trabajo futuras

---

En el siguiente apartado se detallan las conclusiones obtenidas del desarrollo del proyecto. A partir de estas conclusiones, se sugerirán algunas líneas de trabajo futuras que podrían complementar el proyecto, o ser proyectos completamente diferentes a este, pero con cierta relación.

## 7.1. Conclusiones

- El objetivo del proyecto no se ha cumplido completamente. Dada la magnitud del mismo, ha sido imposible llevar a cabo todas las pruebas que habrían supuesto su finalización. Relacionado con una cuestión de tiempo y de acceso a una instalación cerrada y en la que se puedan realizar con total seguridad.
- En múltiples asignaturas de la carrera se enuncia el principio *divide y vencerás*. Sin duda, seguir este principio en el desarrollo de software se ha mostrado fundamental a la hora de escribir código fácilmente mantenable.
- Establecer una arquitectura genérica, que permita ir *bajando* por la arquitectura desde una visión más abstracta a una visión más concreta de los diferentes problemas, puede parecer sobredimensionar y dificultar el problema, pero definitivamente es rentable cuando se quiere evitar repetir código.
- Hacer uso de PyCharm ha facilitado mucho el desarrollo del sistema. Presenta una integración casi perfecta con despliegue en remoto, lo que ha facilitado mucho la ejecución, debug y despliegue final de la arquitectura en la RaspberryPi.

- En teoría, todo funciona. En la práctica es cuando se descubren los verdaderos problemas. La componente física del proyecto ha aportado una visión muy crítica sobre los cursos y artículos publicados en internet. Haciendo ver que estos en muchos casos están *ajustados* para que el proceso quede perfecto para su presentación.
- Estimar la duración de las tareas es de gran dificultad, aunque se ha seguido una estrategia más bien conservadora, haciéndolas más largas de lo que en principio se preveía.
- Los servicios de integración continua, sobre todo aquellos relacionados con plataformas de testing, habrían sido de utilidad para evitar cometer algunos errores, sin embargo, no proveen de una forma de someter a pruebas el hardware necesario. Además, no proveen de acceso gratuito a repositorios privados, como es el caso de este proyecto.
- Los sensores utilizados para el proyecto puede que no sean los ideales para el mismo. Si bien son baratos, y fáciles de instalar y utilizar, no son lo suficientemente rápidos en la adquisición de datos.
- Python es «lento», pero cuando se vectoriza el cálculo haciendo uso de Numpy es casi equiparable a C.
- Las pruebas son parte fundamental de un proyecto, sobre todo si incluye una componente física como este. De haber realizado un proyecto más sencillo, se habría dispuesto de más tiempo para llevarlas a cabo correctamente.

## 7.2. Líneas de trabajo futuras

El proyecto realizado ha dado como fruto algunas líneas de trabajo por las que sería interesante continuar. No obstante, lejos de dar por finalizado (sin estarlo) este proyecto, se continuará con su desarrollo a título personal, y además se provee de una lista de proyectos derivados o relacionados con este:

- ZenHub provee de un *pipeline* que establece ciertas funcionalidades en pausa. Es conocido como *IceBox*, y en el se irán reflejando algunas de las funcionalidades que se han dejado a la espera, y que es posible que se implementen con el tiempo. Entre las mejoras a realizar se encuentran:
  - Implementar SLAM para no depender de un mapa prefijado.
  - Hacer uso de sensores más rápidos, o incluso cámaras para el cálculo de distancias.
  - Añadir detección de intrusos mediante diversas técnicas.

- Desarrollar un servidor propio de WebRTC para no depender de UV4L.
- Mejorar la interfaz web. Sin duda esta es la parte más floja del proyecto.
- Finalizar con la etapa de pruebas, llevándolas a cabo con un *drone* mejorado (sensores más rápidos, piezas a medida mediante impresión 3D... etc).
- Implementar un framework de testing que permita hacer uso de mocks de sensores y dispositivos hardware, sin necesidad de crearlos cada vez que se quiera hacer un test.
- Implementar un sistema de detección de intrusos y alertas al panel de control haciendo uso de la cámara disponible en la RaspberryPi.
- Implementar una arquitectura de enjambre de *drones*. Esta arquitectura multi-agente permitiría el uso de múltiples *drones* para llevar a cabo diferentes tareas. La comunicación entre los diferentes agentes para evaluar estrategias, o modificar el comportamiento de los mismos sería de gran utilidad.
- Implementar la creación de diferentes rutas en un equipo más potente. Si el mapa es grande y complejo, la RaspberryPi puede no ser lo suficientemente potente como para hacer esta labor de forma eficiente, de forma que sería interesante que el servidor que proporciona acceso a los *drones*, pueda enviar las rutas que deben seguir estableciendo las metas a lograr.
- Migrar la arquitectura a un RTOS.
- Implementar una controladora de vuelo dedicada al propósito de este proyecto, en lugar de depender de las existentes en el mercado.
- Implementar un sistema de evasión de obstáculos no basado en un algoritmo como el VFH, sino en una CNN<sup>21</sup> haciendo uso de cámaras.

---

<sup>21</sup>Convolutional Neural Network. Este tipo de redes neuronales son muy utilizadas en procesamiento de imágenes.

---

## Bibliografía

---

- [1] S. N. Gordon, J. Salmond, “Novel approach to nonlinear/non-gaussian bayesian state estimation,” 1993. [Online]. Available: <http://ieeexplore.ieee.org/document/210672/>
- [2] I. M. Rekleitis, “A particle filter tutorial for mobile robot localization,” 2003. [Online]. Available: <http://www.cim.mcgill.ca/~yiannis/particletutorial.pdf>
- [3] D. Sabinasz, “Robot localization IV: The particle filter,” 2017, [Internet; descargado 9-abril-2018]. [Online]. Available: <http://www.deepideas.net/robot-localization-particle-filter/>
- [4] D. Gallardo López, “Localización basada en filtros de partículas,” 2010. [Online]. Available: [https://rua.ua.es/dspace/bitstream/10045/9887/7/Gallardo-Lopez-Domingo\\_6.pdf](https://rua.ua.es/dspace/bitstream/10045/9887/7/Gallardo-Lopez-Domingo_6.pdf)
- [5] O. Khatib, “Real time obstacle avoidance for manipulators and mobile robots,” 1986. [Online]. Available: [https://cs.stanford.edu/group/manips/publications/pdfs/Khatib\\_1986\\_IJRR.pdf](https://cs.stanford.edu/group/manips/publications/pdfs/Khatib_1986_IJRR.pdf)
- [6] Y. K. J. Borenstein, “Potential field methods and their inherent limitations for mobile robot navigation,” 1991. [Online]. Available: [https://pdfs.semanticscholar.org/738a/a7b9fda536e7d603e9d84725f38f96b50150.pdf?\\_\\_ga=2.22475512.2130287822.1523287256-1927681115.1523287256](https://pdfs.semanticscholar.org/738a/a7b9fda536e7d603e9d84725f38f96b50150.pdf?__ga=2.22475512.2130287822.1523287256-1927681115.1523287256)
- [7] ——, “The virtual force field (vff) and the vector field histogram (vfh) methods – fast obstacle avoidance for mobile robots,” 1991. [Online]. Available: <http://www-personal.umich.edu/~johannb/vff&vfh.htm>
- [8] ——, “The vector field histogram - fast obstacle avoidance for mobile robots,” 1991. [Online]. Available: <http://www-personal.umich.edu/~johannb/Papers/paper16.pdf>

- [9] ——, “Real-time obstacle avoidance for fast mobile robots in cluttered environments,” 1990. [Online]. Available: <http://www-personal.umich.edu/~johannb/Papers/paper17.pdf>
- [10] H. Moravec and A. E. Elfes, “High resolution maps from wide angle sonar,” in *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, March 1985, pp. 116 – 121. [Online]. Available: [https://www.ri.cmu.edu/pub\\_files/pub4/moravec\\_hans\\_1985\\_1/moravec\\_hans\\_1985\\_1.pdf](https://www.ri.cmu.edu/pub_files/pub4/moravec_hans_1985_1/moravec_hans_1985_1.pdf)
- [11] J. Geerling, “Raspberry pi zero - conserve power and reduce draw to 80ma,” 2015, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.jeffgeerling.com/blogs/jeff-geerling/raspberry-pi-zero-conserve-energy>
- [12] ——, “How to overclock the microsd card reader on a raspberry pi 3,” 2016, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.jeffgeerling.com/blog/2016/how-overclock-microsd-card-reader-on-raspberry-pi-3>
- [13] PhilE, “Pi3 wifi stopped working,” 2016, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.raspberrypi.org/forums/viewtopic.php?p=928516#p921981>
- [14] ElecFreaks, “Ultrasonic ranging module hc - sr04,” 2015, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>
- [15] Wikipedia, “Pulse width modulation,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: [https://en.wikipedia.org/wiki/Pulse-width\\_modulation](https://en.wikipedia.org/wiki/Pulse-width_modulation)
- [16] e. a. Trollcop, Fiendie, “Multiwii,” 2012, [Internet; descargado 9-abril-2018]. [Online]. Available: <http://www.multiwii.com>
- [17] MultiWii, “Multiwii serial protocol,” 2015, [Internet; descargado 9-abril-2018]. [Online]. Available: [http://www.multiwii.com/wiki/index.php?title=Multiwii\\_Serial\\_Protocol](http://www.multiwii.com/wiki/index.php?title=Multiwii_Serial_Protocol)
- [18] P. S. Foundation, “Python struct module,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://docs.python.org/3/library/struct.html>
- [19] e. a. T. Ylonen, Cisco, “The secure shell transport layer protocol,” 2006, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://tools.ietf.org/html/rfc4253>
- [20] A. M. I. Fette, “Websocket,” 2011, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://tools.ietf.org/html/rfc6455>

- [21] Wikipedia, “Websocket,” 2017, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://es.wikipedia.org/wiki/WebSocket>
- [22] F. e. a. Google, “Webrtc,” 2012, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://webrtc.org>
- [23] ———, “Webrtc,” 2012, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://developer.mozilla.org/es/docs/WebRTC>
- [24] J. Rosenberg, “Interactive connectivity establishment,” 2010, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://tools.ietf.org/html/rfc5245>
- [25] e. a. J. Rosenberg, Cisco, “Stun,” 2008, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://tools.ietf.org/html/rfc5389>
- [26] e. a. R. Mahy, P. Matthews, “Turn,” 2008, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://tools.ietf.org/html/rfc5766>
- [27] S. Dutton, “Getting started with webrtc,” 2014, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.html5rocks.com/en/tutorials/webrtc/basics/>
- [28] Wikipedia, “Scrum (desarrollo de software),” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: [https://es.wikipedia.org/wiki/Scrum\\_\(desarrollo\\_de\\_software\)](https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software))
- [29] V. Driessen, “Gitflow - a successful Git branching model,” 2010, [Internet; descargado 9-abril-2018]. [Online]. Available: <http://nvie.com/posts/a-successful-git-branching-model/>
- [30] Wikipedia, “Kanban (desarrollo),” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: [https://es.wikipedia.org/wiki/Kanban\\_\(desarrollo\)](https://es.wikipedia.org/wiki/Kanban_(desarrollo))
- [31] GitHub, “How developers work,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://github.com/features#documentation>
- [32] ZenHub, “Prioritize your projects and release better software,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.zenhub.com/product>
- [33] GitKraken, “The legendary git gui client for windows, mac and linux,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.gitkraken.com>
- [34] JetBrains, “Python ide for professional developers,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.jetbrains.com/pycharm/>

- [35] P. S. Foundation, “The pypa recommended tool for installing python packages,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://pypi.python.org/pypi/pip>
- [36] JetBrains, “The smartest javascript ide,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.jetbrains.com/webstorm/>
- [37] Wikipedia, “Latex,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://es.wikipedia.org/wiki/LaTeX>
- [38] TexMaker, “Texmaker. free cross-platform latex editor since 2003,” 2003, [Internet; descargado 9-abril-2018]. [Online]. Available: <http://www.xm1math.net/texmaker/>
- [39] T. user groups, “Tex live,” 1996, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.tug.org/texlive/>
- [40] T. Sebastian, “Artificial intelligence for robotics,” 2016, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://classroom.udacity.com/courses/cs373>
- [41] T. F. Mega-Tutorial, “Miguel grinberg,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>
- [42] L. Projects, “User space video 4 linux,” 2017, [Internet; descargado 1-junio-2018]. [Online]. Available: <https://www.linux-projects.org/uv4l/>
- [43] E. Mundo, “Este dron te vigila para que eno hagas horas extra,” 2017, [Internet; descargado 9-abril-2018]. [Online]. Available: <http://www.elmundo.es/tecnologia/2017/12/11/5a2e550ee2704e184f8b458b.html>
- [44] T. C. LTD, “T-frend,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <http://taisei-bm.co.jp/beyond/t-frend/>
- [45] ErleRobotics, “Erle robotics,” 2012, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://erlerobotics.com/blog/>
- [46] FlyPulse, “Flypulse airborne aid with drones,” 2015, [Internet; descargado 9-abril-2018]. [Online]. Available: <http://www.flypulse.se>
- [47] TU Delft, “Ambulance drone,” 2014, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.tudelft.nl/en/ide/research/research-labs/applied-labs/ambulance-drone/>