



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 — 6 de mayo 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 del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 6 de mayo de 2018

Vº. Bº. del Tutor:

Vº. Bº. del Tutor:

Vº. Bº. del Tutor:

D. Dr. Alejandro
Merino Gómez

D. Dr. César Ignacio
García Osorio

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.

Descriptores

drone, UAV, semi-autónomo, semi-automático, vigilancia, navegación, interior, vídeo, filtro de partículas, campos potenciales, Histograma de Campos Vectoriales, VFH, búsqueda de ruta, MSP, raspberry pi

Abstract

The point 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 it to the security guard in charge.

Keywords

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

Índice general

Índice general	III
Índice de figuras	V
Índice de tablas	VI
Introducción	1
Objetivos del proyecto	3
2.1. Objetivos marcados por Requisitos Funcionales	3
2.2. Objetivos técnicos derivados	3
2.3. Objetivos personales	4
Conceptos teóricos	5
3.1. Algoritmia	5
Particle Filter localization	5
Campos Potenciales	9
Vector Field Histogram	15
3.2. Dispositivos Físicos	25
Raspberry Pi	25
Controladora de Vuelo	28
3.3. Protocolos	31
Pulse Width Modulation	31
MultiWii Serial Protocol	32
Secure SHell	37
WebSocket	38
WebRTC	39
Técnicas y herramientas	41
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 ^A T _E X	44
Aspectos relevantes del desarrollo del proyecto	46
Trabajos relacionados	47
Conclusiones y Líneas de trabajo futuras	48
Bibliografía	49

Índice de figuras

3.1. Gradiente de Potencial de Atracción.	11
3.2. Gradiente de Potencial de Repulsión.	12
3.3. Mínimo local en zonas convexas.	13
3.4. Movimiento oscilatorio en corredores estrechos.	14
3.5. Distribución Histográfica de Probabilidades	17
3.6. Angulos relativos al VCP del drone	18
3.7. En azul Polar Obstacle Density(POD). En naranja el suavizado de este (sPOD).	20
3.8. Valle ancho encontrado entre el obstáculo, a la derecha, y un espacio abierto, a la izquierda.	21
3.9. Movimiento circular al encontrarse k_{targ} en un sector seguro por ecuación 15	22
3.10. Valle estrecho encontrado entre dos obstáculos. La ecuación 15 proporciona un sector seguro ubicado en la zona central del valle.	23
3.11. Con un umbral muy elevado, puede darse el caso de que todo sean valles hasta estar demasiado cerca del obstáculo.	24
3.12. Con un umbral muy bajo, puede darse el caso de que los valles sean muy estrechos, o incluso inexistentes	24
3.13. Raspberry Pi 3.	26
3.14. Benchmark de lector microSD OC.	27
3.15. Controladora de Vuelo Flip32.	28
3.16. Modulación en Ancho de Pulso. Arriba 100 % del ciclo usado. Centro 25 % del ciclo usado. Abajo 50 % del ciclo usado	32
3.17. Composición de un mensaje MSP	34

Índice de tablas

3.1. Componentes de una Raspberry Pi 3 Model B	25
3.3. Mensajes MSP. Estructura de datos y representación	36
3.2. Nomenclatura del módulo Struct de la implementación 3.5 de Python	37

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 ocupar crece 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 al 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 ~~revisión~~ 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.



Este proyecto propone una **implementación** 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, ó uso** dado a, 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, papers 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 plantea a la hora de llevar a la práctica el proyecto.

2.1. Objetivos marcados por Requisitos Funcionales


- Diseñar un drone capaz de recorrer un espacio, en el que existan obstáculos, de forma segura.
- Diseñar un sistema de acceso al drone de forma segura.
- 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, 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 dron 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*.
- Hacer 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.

2.3. Objetivos personales

- Crear un **nuevo** sistema de navegación semiautónoma haciendo uso de drones.
- Hacer uso  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.
- 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

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.



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.



El Filtro de Partículas para localización, o Localización Monte Carlo, es un algoritmo que estima la posición y orientación de un agente en un entorno, haciendo uso de múltiples hipótesis ponderadas.

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 ??, 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 es posible incrementar el número de variables que se deben tener en cuenta para definir cómo de parecidas son N partículas y el agente.

Habitualmente, se comienza con una distribución de partículas aleatoriamente dispersas por el mapa, obviamente no se establecen partículas dentro de zonas ocupadas por obstáculos, con una orientación definida. 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 moverse las partículas se mueven en la dirección y orientación que determina el agente, i.e: Si el agente ha rotado 10° y se ha desplazado hacia delante 10cm, todas las partículas deberán rotar 10° 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 tupla, donde $n = 3$ en este caso, tal que: $[x, y, \theta]$ siendo x e y las coordenadas y θ la orientación.

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. En el *Filtro de Partículas* la variable de interés, la posición del agente, es representada por un set de M partículas en el instante $t = k$, tal que $S^k = [x_j^k, w_j^k] : j = 1..M$. 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*¹, 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.

Actualización de movimiento

Durante la actualización de movimiento, 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° 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° para dirigirse hacia la meta por un camino seguro, el agente *tratará* de rotar n° pero es muy probable que infra/sobrecorrija. 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 $t = k$:

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

¹En un proceso estocástico que cumple la propiedad de Markov, el estado futuro depende únicamente del estado presente, y no de los estados pasados.

Donde:

$$\begin{aligned}
 [x, y, \hat{\theta}]^T &= \text{Posición inicial del agente} \\
 \hat{\theta} &= \arctan\left(\frac{\delta y}{\delta x}\right) \\
 [x', y', \hat{\theta}']^T &= \text{Posición final del agente} \\
 \rho &= \sqrt{\Delta x^2 + \Delta y^2}; \text{ la distancia a recorrer}
 \end{aligned}$$

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

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

Donde:

$$\begin{aligned}
 [x, y, \hat{\theta}]^T &= \text{Posición inicial del agente} \\
 \hat{\theta} &= \arctan\left(\frac{\delta y}{\delta x}\right) \\
 [x', y', \hat{\theta}']^T &= \text{Posición final del agente} \\
 \rho &= \sqrt{\Delta x^2 + \Delta y^2} * f(\text{rand}|\mu, \sigma^2); \text{ la distancia a recorrer } \mathbf{añadiendo ruido} \\
 &\quad \text{dado por una distribución normal} \\
 \mu &= 0,0; \text{ se desea un valor aleatorio dentro de una gaussiana con centro en 0.0}
 \end{aligned}$$

σ = Errores de los sensores o sistemas de movimiento

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

De forma inevitable, las partículas que componen el filtro divergirán como consecuencia de la actualización de movimiento aplicando 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 quedándose únicamente con aquellas que más afinidad presentan en sus mediciones. Es decir, se mantienen aquellas que representan de forma más precisa la ubicación real del agente.

Para ello, se calcula para cada partícula la probabilidad 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:

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

Donde:

μ = Distancias de la partícula a los diferentes obstáculos.

σ = Ruido del sistema de sensores.

x = Distancias reales obtenidas por los sensores del agente.

Se le asignará a cada partícula un peso, proporcional a la probabilidad de tratarse de la ubicación del agente. Dicho peso será normalizado de la forma habitual mediante:

$$w_j^k = \frac{\tilde{w}_j^k}{\sum_j^M \tilde{w}_j^k} \quad (4)$$

Donde \tilde{w}_j^k es la medida de peso de una partícula j en el instante de tiempo k sin normalizar.

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 [2], 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 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 $U(q) : R^2 \rightarrow R$, 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 sobrereactúe:

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

Donde δ es la distancia euclidiana desde la posición del agente q a la meta $meta$, y ϵ 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:

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

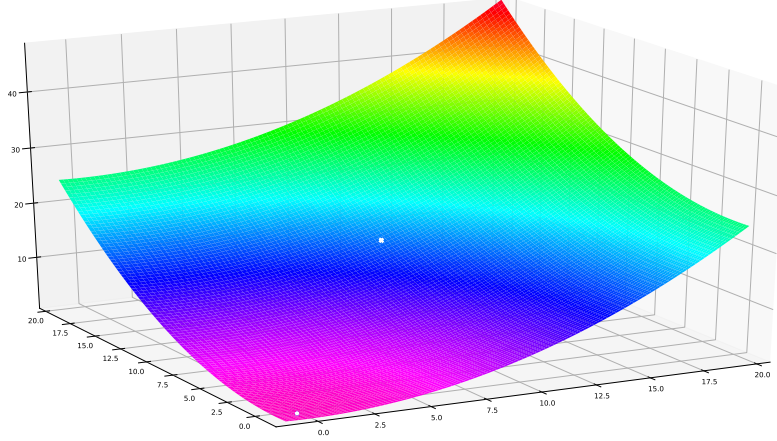


Figura 3.1: Gradiente de Potencial de Atracción.

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:

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

Donde η es un factor de escalado positivo, que permite establecer como de fuerte se ve afectado el agente por el campo repulsor, δ es la distancia euclidiana desde la posición del agente q al obstáculo obs , y ρ es un factor de escalado positivo, que permite establecer la distancia de influencia de forma proporcional. Y su gradiente

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

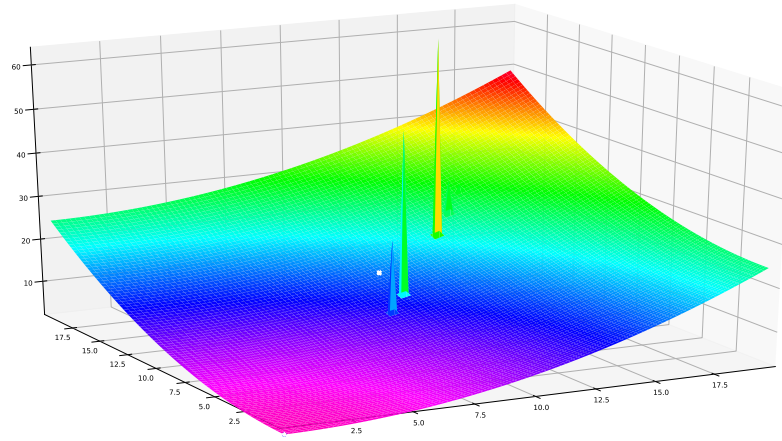



Figura 3.2: Gradiente de Potencial de Repulsión.

En la figura 3.2 se ha creado una serie de obstáculos artificialmente para ilustrar el comportamiento del algoritmo. 

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, [3] 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 [3].

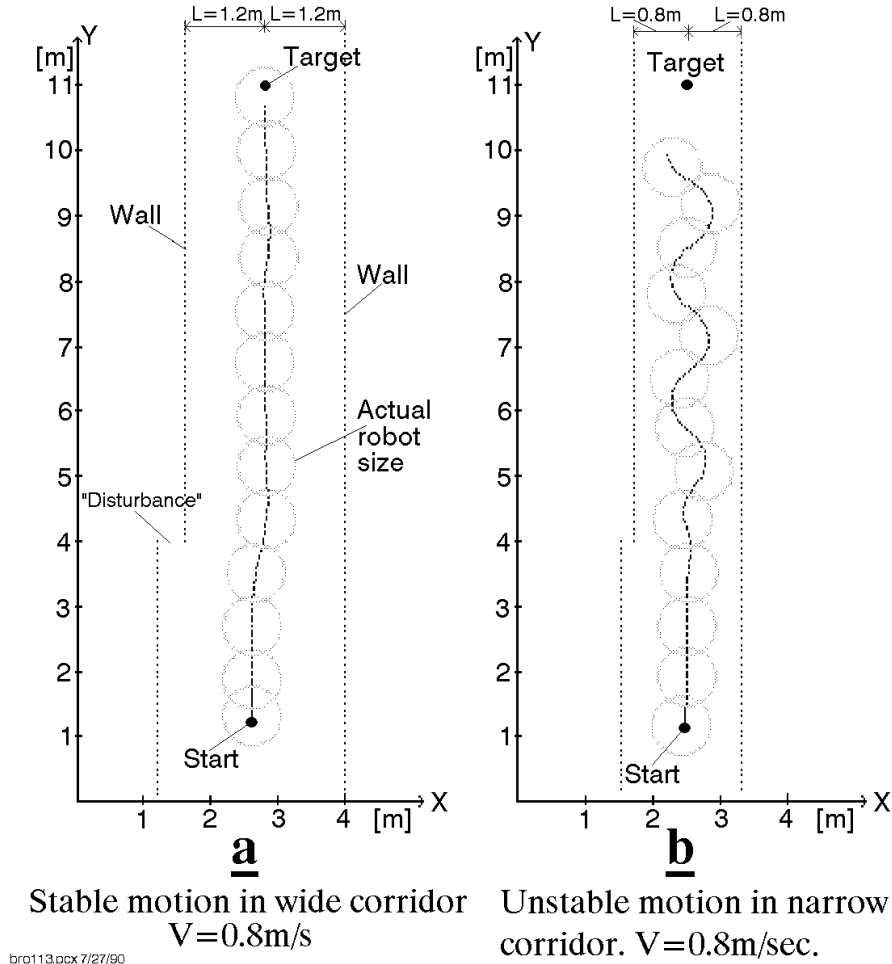


Figura 3.4: Movimiento oscilatorio en corredores estrechos.

- Alta complejidad computacional pese a poder modelarse como calculo 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 desechado el 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, [4] y [5]. 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 3.1.

Se compone de tres partes:

1. Red Cartesiana de Histogramas: Derivado del concepto de *mall de certidumbre* propuesto por Moravec y Elfes en 1985 [6] 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 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 α , de forma que $n * \alpha = 360$; $n \in \mathbb{Z}$. 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 3.1.
3. Capa de salida: Representa el resultado del algoritmo. A partir del POD, se obtienen los posibles *valles*² candidatos existentes entre obstáculos, y se elige el que más se aproxime a la dirección de la meta establecida.

²Así denominados por su representación en forma de histograma

Finalmente, entrega los valores de cambio de dirección necesario en cada momento. Ver 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 , i.e: Si se dispone de un sensor con un rango de medidas de hasta 400cm se puede crear una malla de 81x81, suponiendo que cada celda tenga un tamaño de $c = 5$, 5x5cm, en cuyo centro se encuentra el drone, dejando 40 filas y columnas a cada lado. Y estableciendo así, un perímetro de medidas de 200cm 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 ???. 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 esta 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, extraída de [4]

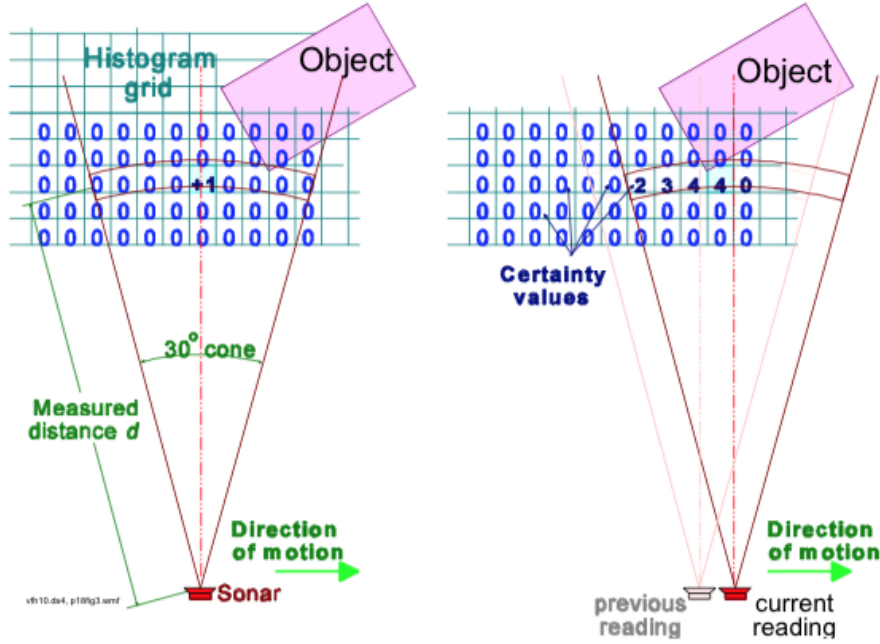


Figura 3.5: Distribución Histórica de Probabilidades

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 β de la celda en cuestión al centro del agente VCP^3 , 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 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

³El 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


-135			-90			-45
	-135		-90		-45	
		-135	-90	-45		
±180	±180	±180		0	0	0
		135	90	45		
	135		90		45	
135			90			45

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 1º y 4º 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.⁴

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

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

⁴Obviamente es lo deseable, no sería eficiente realizar un giro de 225° en el sentido de las agujas del reloj, cuando uno de 135° 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 dron

b = Representa la distancia desde la que un obstáculo afecta al dron

son positivos

El plano es convertido en una secuencia de sectores que cubren los 360° . Se definen, por tanto, $n \in \mathbb{Z}$ sectores k de amplitud α , por ejemplo $n = 72; \alpha = 5$. Y se distribuyen las medidas tomadas anteriormente en los diferentes ángulos relativos al VCP del dron, en los sectores k_i correspondientes, véase [10](#)

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

con un valor h_k , véase [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 [\[4\]](#) 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 asutriaco 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

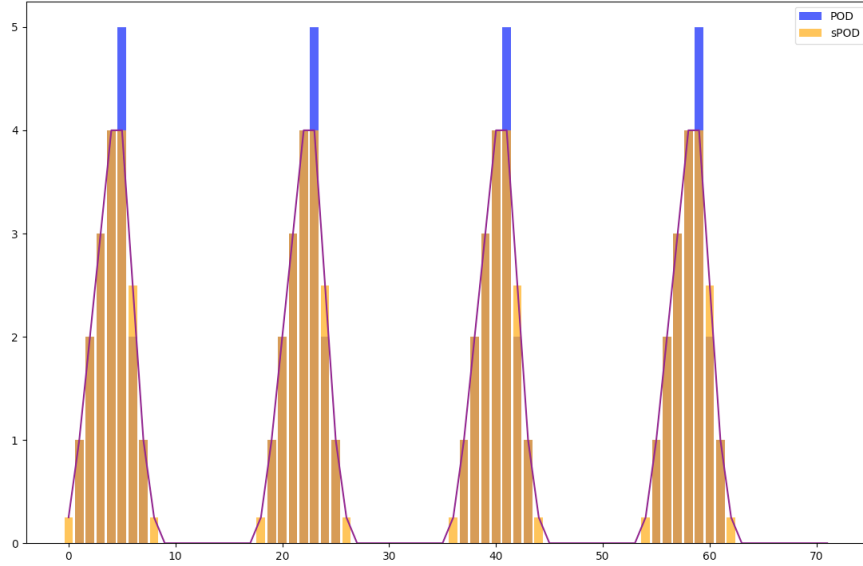


Figura 3.7: En azul Polar Obstacle Density(POD). En naranja el suavizado de este (sPOD).

Output Layer

En el último estadio del algoritmo VFH, se computa el ángulo de giro θ que se debe aplicar al dron 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 que 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 esta 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 imagen 3.8.

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

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

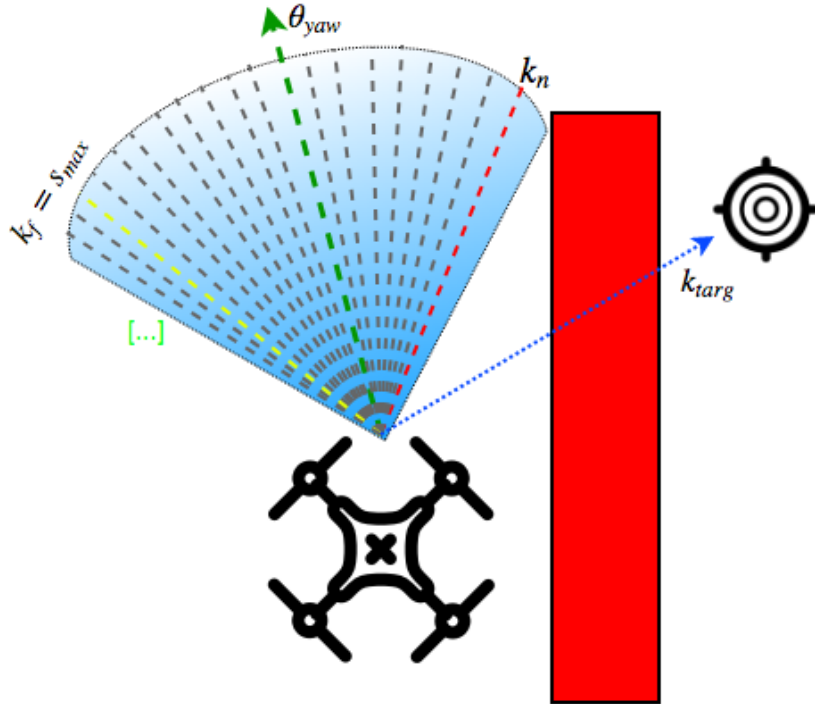


Figura 3.8: Valle ancho encontrado entre el obstáculo, a la derecha, y un espacio abierto, a la izquierda.

Sin embargo, la ecuación 15 presenta un comportamiento errático cuando el destino se encuentra dentro de un valle, dado que se establece θ_{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 imagen 3.9

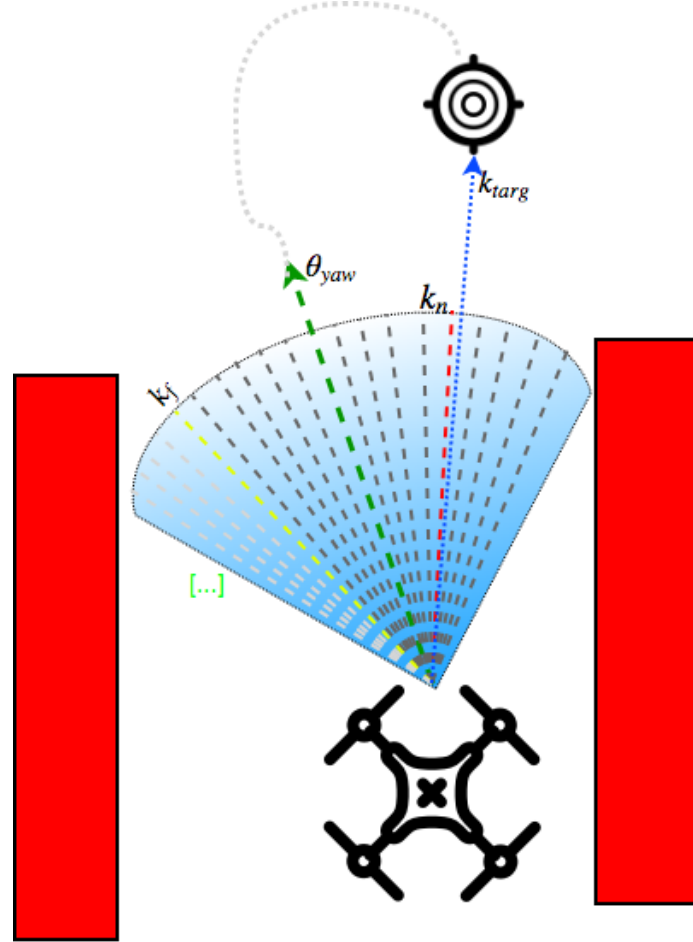


Figura 3.9: Movimiento circular al encontrarse k_{targ} en un sector seguro por ecuación 15

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 segura⁵. Además en el momento en que la posición del dron sea intermedia entre dos obstáculos, la

⁵Una cualidad de la que el algoritmo de campos potenciales adolecía, generando movimientos oscilatorios en espacios cerrados, como ya se vió en 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 θ_{yaw} , siempre centrado entre los dos obstáculos, ilustrado en la imagen 3.10.

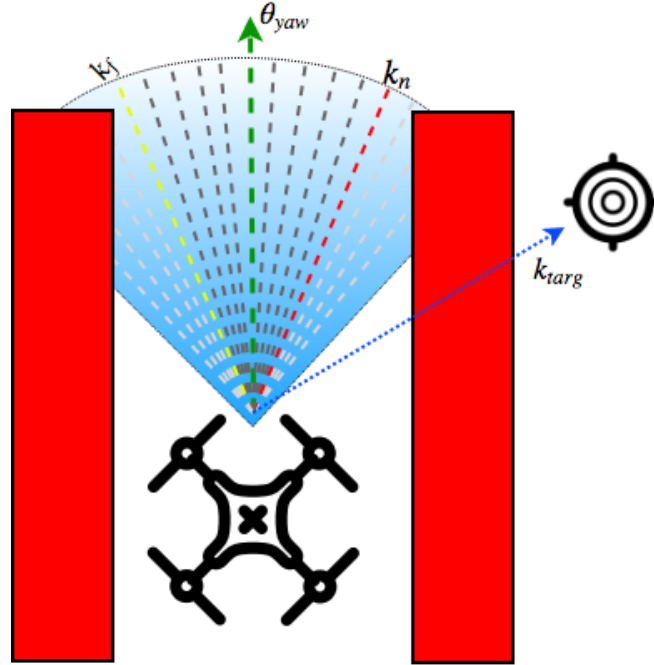


Figura 3.10: Valle estrecho encontrado entre dos obstáculos. La ecuación 15 proporciona un sector seguro ubicado en la zona central del valle.

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 dron.

Por ejemplo, establecer un umbral muy alto, ver imagen 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.

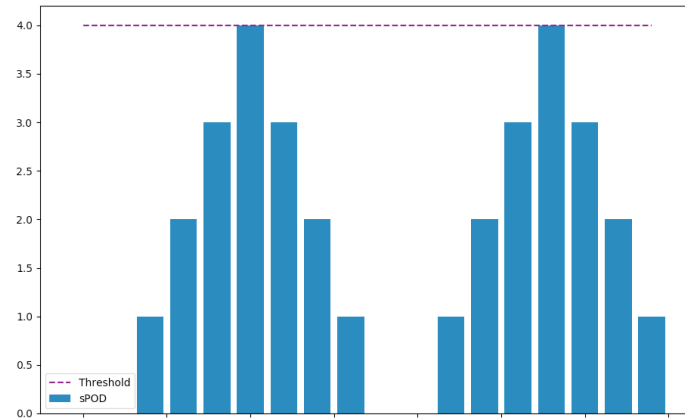


Figura 3.11: Con un umbral muy elevado, puede darse el caso de que todo sean valles hasta estar demasiado cerca del obstáculo.

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

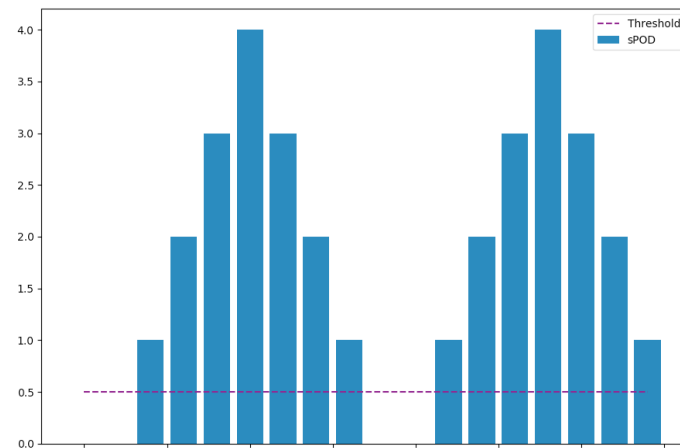


Figura 3.12: Con un umbral muy bajo, puede darse el caso de que los valles sean muy estrechos, o incluso inexistentes

3.2. Dispositivos Físicos

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 UK 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 x 56mm (LxA), 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 Raspbian⁶, 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 micro SD mediante el siguiente procedimiento por consola⁷:

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

⁶Descargable desde: <https://www.raspberrypi.org/downloads/raspbian>

⁷Realizado en OSX, aunque en Linux es muy similar

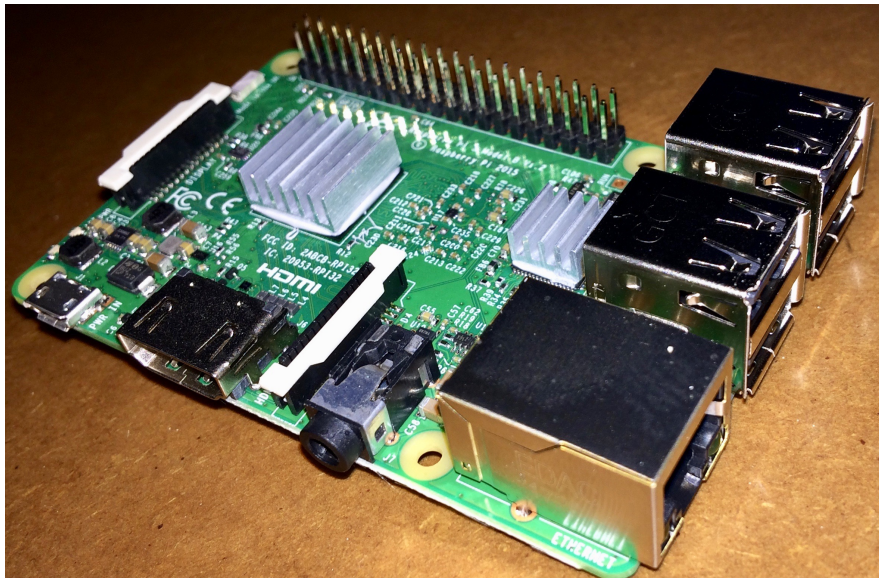


Figura 3.13: Raspberry Pi 3.

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 directa⁸, 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 2,5A⁹ para funcionar al máximo nivel de estrés para el procesador y alimentar dispositivos USB.

Sin embargo, este no es estrictamente nuestro caso, véase 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.

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

⁸Véase `man hdiutil`, sección *DEVICE SPECIAL FILES*

⁹Véase <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 0_DIRECT disk reads: 108 MB in 3.02 seconds = 35.78 MB/sec
Timing 0_DIRECT disk reads: 108 MB in 3.02 seconds = 35.72 MB/sec
Timing 0_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.

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 [7].

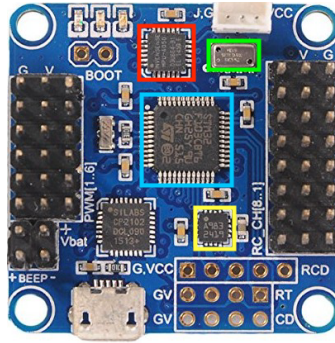


Figura 3.15: Controladora de Vuelo Flip32.

- 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 imagen 3.14. Descrito en [8] y [9].
- 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* 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 dron, tomando medidas de los sensores de que 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 imagen 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 imagen [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 imagen [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:


- IMU¹⁰ 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.¹¹
- 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.

¹⁰Unidad de Medición Inercial.

¹¹Disponible en: <http://magnetic-declination.com/>

3.3. Protocolos

Pulse Width Modulation

La modulación por ancho de pulso [10], 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 imagen 3.16. 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 2000µs. Cualquier valor por debajo, o por encima, haría entrar la controladora en FailSafe¹². En  caso de este proyecto, se ha determinado que no se hará uso de este protocolo para la comunicación entre la Raspberry Pi y la controladora de vuelo, véase 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.

¹²Al 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

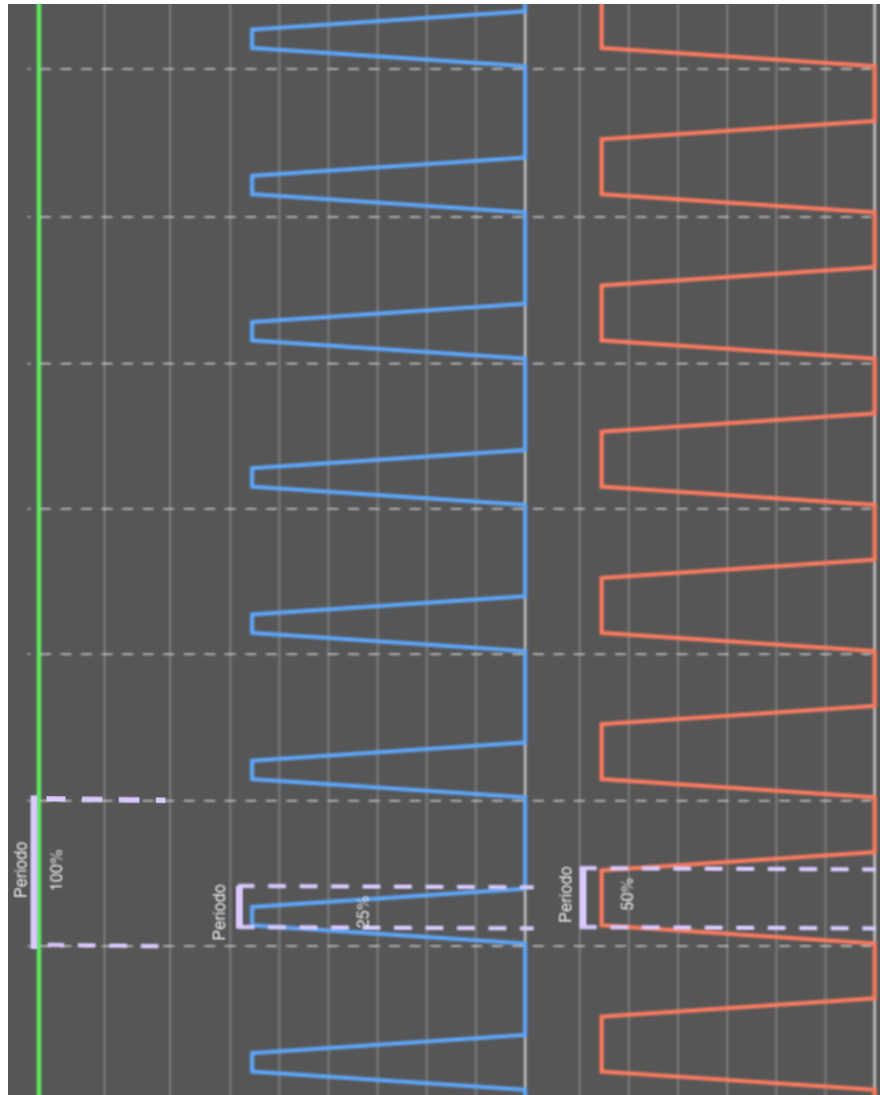


Figura 3.16: Modulación en Ancho de Pulso. Arriba 100 % del ciclo usado. Centro 25 % del ciclo usado. Abajo 50 % del ciclo usado

MultiWii Serial Protocol

MultiWii es un software de control de multirrotores 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, [11]. 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, [12]. 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 imagen [3.17](#)

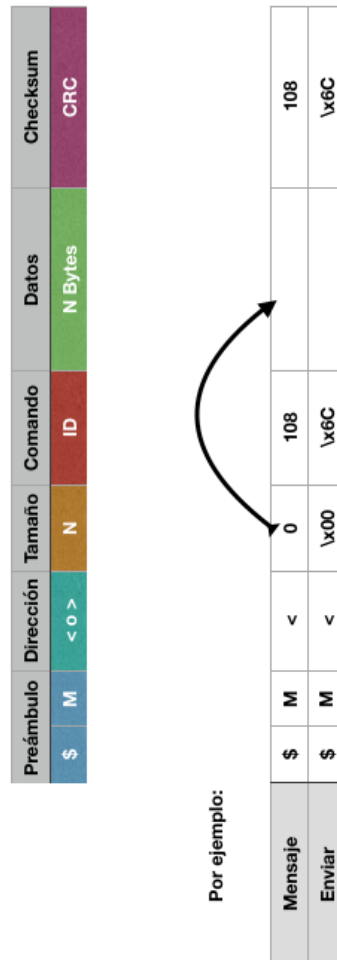


Figura 3.17: Composición de un mensaje MSP

- 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 caracter 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 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 byte de control. 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 dron a la FC. Es decir, en lugar de utilizar un receptor de radio, se utilizará un puerto serie para obtener la telemetría¹³, 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, [13] y descrita en la tabla 3.2

¹³Se 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 radio-control.
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, [12]. 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.

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'	Caracter utilizado para representar un entero de un byte sin signo.
'H'	Caracter utilizado para representar un entero de dos bytes sin signo.
'c'	Caracter utilizado para representar un caracter 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 caracter 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* [14] 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 entorno de escritorio. El acceso a la Raspberry Pi que controla el dron, 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 tremendamente 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, [15] y [16], 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 sección 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 CA¹⁴ de confianza.

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.

¹⁴Certificate Authority. Una entidad que proporciona certificados digitales.

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, [17], [18].

Se apoya en ICE o Interactive Connectivity Establishment, [19] 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 NAT¹⁵, 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, [20].
- Servidores TURN (Traversal Using Relays around NAT), como último recurso. En el caso de que no se pueda realizar la comunicación P2P, 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*, [21]. 

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.

¹⁵Network 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

- 2 Signaling. La aplicación negocia el tipo de comunicación que espera, es decir, el codec de video, 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 [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 el 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 [\[22\]](#). 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. Ver [23]

GitFlow

El flujo de trabajo de Git permite establecer una forma organizada y ágil de llevar a cabo todas las aportaciones del proyecto. [24]

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 arreglar bugs 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. Ver [25] 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. [26]

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. Ver [27]

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, [28]. 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, [29].

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 [30], 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.

WebStorm

WebStorm es un IDE, *Integrated Development Environment*, para el desarrollo de aplicaciones web desarrollado por JetBrains, [31].

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^AT_EX

Para llevar a cabo la documentación del proyecto se ha hecho uso de L^AT_EX[32]. Se trata de un lenguaje de marcas que permite la redacción de textos que presentan alta calidad tipográfica. La filosofía de trabajo con L^AT_EX se basa en centrarse en el contenido y no en la forma. Es decir, el redactor de un documento no tiene que 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^AT_EX. Se trata de un entorno parecido al habitual procesador de textos, que permite la redacción de textos haciendo uso del lenguaje de marcas *Tex*. Se caracteriza por implementar un

corrector, disponer de auto-completado de marcas para \LaTeX y un visor PDF del documento generado. [33]

TeXLive es una distribución de \LaTeX . Se trata de un compendio de herramientas, fuentes y archivos de configuración que permiten la compilación de código *TeX* a un documento legible, como un PDF. [34]

Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros³, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

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] 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
- [3] 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
- [4] —, “The vector field histogram - fast obstacle avoidance for mobile robots,” 1991. [Online]. Available: <http://www-personal.umich.edu/~johannb/Papers/paper16.pdf>
- [5] —, “Real-time obstacle avoidance for fast mobile robots in cluttered environments,” 1990. [Online]. Available: <http://www-personal.umich.edu/~johannb/Papers/paper17.pdf>
- [6] 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
- [7] 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>

- [8] —, “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>
- [9] PhilE, “Pi3 wifi stopped working,” 2016, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.raspberrypi.org/forums/viewtopic.php?p=928516#p921981>
- [10] Wikipedia, “Pulse width modulation,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: https://en.wikipedia.org/wiki/Pulse-width_modulation
- [11] e. a. Trollcop, Fiendie, “Multiwii,” 2012, [Internet; descargado 9-abril-2018]. [Online]. Available: <http://www.multiwii.com>
- [12] MultiWii, “Multiwii serial protocol,” 2015, [Internet; descargado 9-abril-2018]. [Online]. Available: http://www.multiwii.com/wiki/index.php?title=Multiwii_Serial_Protocol
- [13] P. S. Foundation, “Python struct module,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://docs.python.org/3/library/struct.html>
- [14] 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>
- [15] A. M. I. Fette, “Websocket,” 2011, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://tools.ietf.org/html/rfc6455>
- [16] Wikipedia, “Websocket,” 2017, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://es.wikipedia.org/wiki/WebSocket>
- [17] F. e. a. Google, “WebRTC,” 2012, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://webrtc.org>
- [18] —, “WebRTC,” 2012, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://developer.mozilla.org/es/docs/WebRTC>
- [19] J. Rosenberg, “Interactive connectivity establishment,” 2010, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://tools.ietf.org/html/rfc5245>
- [20] e. a. J. Rosenberg, Cisco, “Stun,” 2008, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://tools.ietf.org/html/rfc5389>
- [21] e. a. R. Mahy, P. Matthews, “Turn,” 2008, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://tools.ietf.org/html/rfc5766>

- [22] S. Dutton, “Getting started with webrtc,” 2014, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.html5rocks.com/en/tutorials/webrtc/basics/>
- [23] 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))
- [24] 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/>
- [25] 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))
- [26] GitHub, “How developers work,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://github.com/features#documentation>
- [27] ZenHub, “Prioritize your projects and release better software,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.zenhub.com/product>
- [28] GitKraken, “The legendary git gui client for windows, mac and linux,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.gitkraken.com>
- [29] JetBrains, “The smartest javascript ide,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.jetbrains.com/webstorm/>
- [30] 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>
- [31] JetBrains, “Python ide for professional developers,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.jetbrains.com/pycharm/>
- [32] Wikipedia, “Latex,” 2018, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://es.wikipedia.org/wiki/LaTeX>
- [33] TexMaker, “Texmaker. free cross-platform latex editor since 2003,” 2003, [Internet; descargado 9-abril-2018]. [Online]. Available: <http://www.xmlmath.net/texmaker/>
- [34] T. user groups, “Tex live,” 1996, [Internet; descargado 9-abril-2018]. [Online]. Available: <https://www.tug.org/texlive/>