



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE  
TELECOMUNICACIÓN

GRADO EN INGENIERÍA AEREOESPACIAL EN AERONAVEGACIÓN

**TRABAJO FIN DE GRADO**

**Drones**

Autor: Jesús Saiz Colomina

Tutor: José María Cañas Plaza

Curso académico 2017/2018



# Agradecimientos

Hola

# Resumen

Cada día podemos observar el gran crecimiento que se está llevando a cabo con los drones, tanto a nivel de ventas de robots aéreos como a nivel de software y capacidades que pueden llevar a cabo los mismos. No por esto quiere decir que sea un nuevo robot ya que estos tienen mucha historia, se empezaron a utilizar con fines bélicos, sus primeras misiones fueron de localización y destrucción remota y luego como medio de espionaje contra el enemigo en la primera guerra mundial (1916 primera aparición).

En este Trabajo de Fin de Grado se abordan diferentes capacidades de los drones implementándolas todas ellas en un solo uso para crear un programa que permita tanto el uso sencillo de softwares de navegación y guiado utilizando autolocalización visual como la implementación de herramientas de aterrizaje y despegue guiado mediante algoritmos de navegación visual. Con ello se crea una navegación y guiado de un dron completamente autónomo, únicamente utilizando la ayuda de balizas de aterrizaje y despegue, y localizadores (AprilTags). Todas estas funcionalidades se implementan en un único programa con la herramienta Visual States, la cual permite dividir el conjunto de capacidades en diferentes apartados.

Con este trabajo quiero adentrarme en este nuevo mundo y llegar a conocer de lo que son capaces los robots aéreos, gracias a los avances de las nuevas tecnologías y a la posibilidad de adaptación en los drones. Para ello se ha implementado una nueva funcionalidad para los drones mediante un nuevo software y la utilización de funciones ya creadas en JdeRobot. El componente final se ha escrito con el lenguaje de programación Python, en la versión JdeRobot 5.6 y probado con el simulador Gazebo, en el que se han realizado las simulaciones.

# Índice general

<b>Índice de figuras</b>	<b>VI</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Robótica actual . . . . .	1
1.1.1. Clasificación . . . . .	2
1.1.2. Aplicaciones . . . . .	3
1.2. Robótica Aérea . . . . .	7
1.2.1. Componentes . . . . .	9
1.2.2. Movimiento de cuadricópteros . . . . .	10
1.2.3. Marco legislativo español para drones . . . . .	11
1.3. Visión Artificial y Autolocalización . . . . .	15
1.4. Robótica aérea con JdeRobot . . . . .	16
<b>2. Objetivos</b>	<b>19</b>
2.1. Problemas a abordar . . . . .	19
2.2. Requisitos . . . . .	21
2.3. Metodología . . . . .	21
2.4. Plan de trabajo . . . . .	23
<b>3. Infraestructura</b>	<b>25</b>
3.1. Entorno JdeRobot . . . . .	25
3.1.1. Herramienta ColorTuner . . . . .	26
3.1.2. Interfaz Pose3D . . . . .	26

<b>ÍNDICE GENERAL</b>	<b>IV</b>
3.1.3. Plugin ArDrone2 en Gazebo . . . . .	26
3.1.4. Práctica Navegación por control de posición . . . . .	27
3.1.5. Práctica drones Gato-Ratón . . . . .	27
3.2. Gazebo . . . . .	27
3.3. Visual States . . . . .	28
3.4. Balizas visuales AprilTags . . . . .	30
3.5. Biblioteca OpenCV . . . . .	30
3.6. Slam-Visualmakers . . . . .	31
3.7. NumPy . . . . .	33
<b>4. Desarrollo</b>	<b>34</b>
4.1. Diseño . . . . .	34
4.2. Módulo de Control . . . . .	36
4.2.1. Componente de control de pilotaje . . . . .	37
4.2.2. Componente de control de aterrizaje y despegue . . . . .	41
4.3. Módulo de Autolocalización . . . . .	43
4.4. Integración en el Algoritmo Final . . . . .	45
<b>5. Experimentos</b>	<b>46</b>
5.1. A-Controlador . . . . .	46
5.2. B-Autolocalización . . . . .	46
5.3. C-Algoritmo completo . . . . .	46
<b>6. Conclusiones</b>	<b>47</b>
6.1. Conclusiones . . . . .	47
6.2. Trabajos futuros . . . . .	47
<b>7. Bibliografía</b>	<b>48</b>

# Índice de figuras

1.1.	Ejemplos de funcionalidades de Robots . . . . .	2
1.2.	Robot ASIMO realizando acciones cotidianas. . . . .	3
1.3.	Robot PEPPER en un aula como complemento educador. . . . .	4
1.4.	Robot DaVinci utilizado en hospitales. . . . .	5
1.5.	AAI RQ-2 Pioneer - Uno de los primeros Robots Aéreos. . . . .	8
1.6.	Drone ala fija Vs Drone ala móvil . . . . .	9
1.7.	Movimiento del Drone según sus rotores. . . . .	11
1.8.	Dron contra incendios. . . . .	14
1.9.	Visión artificial de un automóvil. . . . .	15
1.10.	TFM Alberto Martín. . . . .	17
1.11.	TFG Manuel Zafra. . . . .	18
2.1.	Representación del desarrollo en espiral. . . . .	22
3.1.	Ejemplo de Pose3DData . . . . .	26
3.2.	Entorno de Simulación Gazebo. . . . .	28
3.3.	Herramienta Visual States sobre Gazebo . . . . .	29
3.4.	Ejemplos de AprilTags. . . . .	30
3.5.	Ejemplo de algoritmo con OpenCV. . . . .	31
3.6.	GUI de Slam-Visualmarkers. . . . .	32
3.7.	Interfaces Slam-Visualmarkers. . . . .	33
4.1.	Esquema representativo de la aplicación. . . . .	35

## *ÍNDICE DE FIGURAS*

VI

4.2. Esquema de los componentes de Visual States. . . . .	37
4.3. Baliza utilizada en Gazebo. . . . .	42
4.4. Ejemplo de los Módulos en Visual States . . . . .	43
4.5. Entorno de la aplicación Slam VisualMarkers . . . . .	44

# Capítulo 1

## Introducción

En este primer capítulo, antes de adentrarnos en la parte mas técnica, se va a introducir al lector de forma breve en qué es la robótica y más concretamente en los robots aéreos, para así poder conocer el estado actual de estos robots y cómo ha evolucionado en los últimos años este sector, creando una gran rama dentro del mundo de la aviación. Este TFG presenta un robot aéreo que utiliza visión para navegar por lo que destacaremos el contexto en el que se encuadra.

### 1.1. Robótica actual

La Robótica es una rama multidisciplinaria de la ingeniería y la ciencia la cual incluye partes de la ingeniería mecánica, ingeniería eléctrica, ciencias de la computación y otras. Estas tecnologías permiten desarrollar máquinas que puedan sustituir a los humanos en sus acciones más cotidianas pero sobretodo están pensadas para ser usadas en ambientes peligrosos (detección y desactivación de bombas), procesos de manufactura (montaje en serie de coches) e incluso en ambientes donde los humanos no pueden sobrevivir (otros planetas como Marte).

Los principios básicos que se plantearon para el correcto funcionamiento de los robots desde que esta palabra fue inventada, ambos hechos realizados por Isaac Asimov, fueron: primero, que ningún robot puede hacer daño a un ser humano, o permitir

que se le haga daño por no actuar; segundo, que un robot debe obedecer las órdenes dadas por un ser humano, excepto si éstas órdenes entran en conflicto con la primera ley; y tercero, que un robot debe proteger su propia existencia en la medida en que está protección no sea incompatible con las leyes anteriores. Todo esto actualmente, como se ha visto en la sociedad actual es muy difícil de que se cumpla y dista mucho de la realidad y, aunque si que se exigen unos mínimos a la hora de crear robots, cualquier parecido es mera casualidad.



(a) Robot Artificiero      (b) Robots en cadena de montaje      (c) Rover Curiosity en Marte

Figura 1.1: Ejemplos de funcionalidades de Robots

### 1.1.1. Clasificación

Hay muchos tipos de clasificación de los robots, uno bastante utilizado y reconocido es según su cronología:

- **1<sup>a</sup> Generación:** Robots manipuladores. Sistemas mecánicos de varias funcionalidades con un sistema de control sencillo, el cual puede ser manual, de secuencia fija o de secuencia variable.
- **2<sup>a</sup> Generación:** Robots de aprendizaje. Son capaces de repetir una secuencia de movimiento que ha sido previamente ejecutada por una persona. El operador realiza los movimientos requeridos mientras el robot los memoriza y le va siguiendo.

- **3<sup>a</sup> Generación:** Robots con control sensorizado. Llevan incorporados controladores, pequeñas computadoras que ejecutan las órdenes de un programa y mediante el manipulado es capaz de realizar los movimientos ordenados.
- **4<sup>a</sup> Generación:** Robots inteligentes. Similares a la generación anterior pero incluyen una gran mejora y es que están equipados con sensores que mediante la comunicación con la computadora de control sobre la realización de las órdenes permite una toma inteligente de decisiones y un control de procesos en tiempo real.

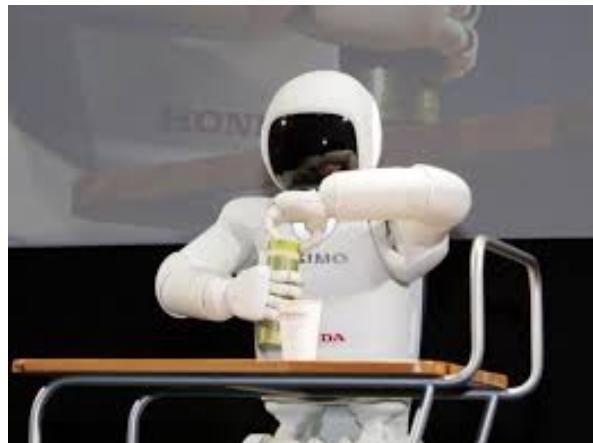


Figura 1.2: Robot ASIMO realizando acciones cotidianas.

### 1.1.2. Aplicaciones

La robótica es una ciencia que está en constante desarrollo, esto ligado al incremento de los procesadores, así como en dispositivos hardware y desarrollo de software ha permitido que hoy en día la robótica esté presente en prácticamente todas las industrias actuales y cotidianas de nuestra vida:

- **Agricultura:** Hoy en día es muy común ver robots que controlen y monitorizan por si solos las cosechas y a medida que pasa el tiempo se vuelven más y más populares. La tecnología robótica aplicada al sector agrícola se encuentra en un

estado de desarrollo avanzado debido a la necesidad de aumentar la producción sin aumentar los recursos al mismo tiempo que se minimiza el impacto ambiental.

- **Educación:** La robótica ha surgido como un recurso didáctico innovador que favorece la enseñanza de conceptos y conocimientos de distintas disciplinas, no únicamente las tecnológicas o científicas. Además esta tecnología se utiliza como factor de motivación, a partir del interés de los niños, para llevar al alumno al desarrollo de su propio conocimiento.



Figura 1.3: Robot PEPPER en un aula como complemento educador.

- **Industria:** Se utilizan tanto para realizar trabajos peligrosos o de gran dificultad para un humano como puede ser la aplicación de sustancias nocivas, el moldeado de materiales o el transporte pesado; como para tareas de inspección y control de calidad mediante visión artificial y sistemas mecánicos. Además, el uso de robots conlleva una mejora de calidad y un gran aumento de la productividad, como por ejemplo ayudando a la logística y el procesado final para los envíos.
- **Automovilismo:** En las fábricas se utilizan para ayudar en la fabricación, como

los brazos robóticos de montaje o para el transporte de materiales, como los AGVs de transporte que son capaces de guiarse por la fábrica autónomamente. Fuera de estas podemos observar vehículos con conducción autónoma que poco a poco son cada vez más utilizados como Tesla, BMW y los sistemas de aparcamiento autónomo. También en la investigación espacial se hace un gran uso de la robótica, lo que permite investigar entornos que para un ser humano serían prácticamente imposibles, como el caso del robot Curiosity mencionado anteriormente.

- **Medicina:** Se han desarrollado dispositivos que permiten realizar desde trabajos quirúrgicos guiados por imágenes hasta cirugía mínimamente invasiva realizada mecánicamente por un robot como es el caso del robot DaVinci capaz de realizar operaciones por sí solo con un nivel de precisión imposible de alcanzar por un humano y una gran velocidad. También podemos encontrar robots asistenciales para personas que necesitan una supervisión y cuidado continuo, prótesis robóticas que van desde la sustitución parcial de alguna parte dañada del cuerpo hasta exoesqueletos. Otro ejemplo estaría en la robótica terapéutica, utilizada como medio de rehabilitación fisiológica y en tratamientos para enfermedades como el Alzheimer.



Figura 1.4: Robot DaVinci utilizado en hospitales.

- **Militar:** En muchas ocasiones los mayores avances en materia de tecnología se han realizado durante períodos de guerra. Actualmente existe una gran gama de vehículos terrestres sin piloto humano con funciones de reconocimiento e incluso algunos vehículos armados, un ejemplo de esto sería el robot PackBot que según el medio en el que se encuentre se puede equipar con diferentes instrumentos para adaptarse a él. El mayor desarrollo en cuanto a robótica militar está siendo llevada a cabo en la robótica aérea, donde estos sistemas han pasado de ser unidades de apoyo a unidades primarias de ataque.
- **Ocio y tiempo libre:** En los últimos años se ha producido una integración de esta tecnología en eventos de cultura, deporte y ocio e incluso en las casas. Podemos ver ejemplos de esto en eventos deportivos en los que compiten en disciplinas solo robots, en las nuevas generaciones de consolas y videojuegos, en los que se hace amplio uso de la visión por computador o en las casas donde podemos ver robots autónomos que se encargan de las tareas domésticas como la aspiradora Roomba. Aquí también podríamos incluir los robots para niños que son capaces de divertir y entretenir a la vez, y están llegando a todos los hogares los más vendidos son los robots de LEGO donde los niños pueden aprender a crear robots y programarlos de una forma muy sencilla.
- **Seguridad:** La robótica ha dado al mundo de la seguridad y la vigilancia una nueva perspectiva, siendo la visión artificial el eje en torno al que giran estas aplicaciones. La automatización de estas tareas permite una mayor facilidad y eficiencia a la hora de ejecutar esta labor, podemos encontrar drones que vigilan grandes concentraciones de personas, cámaras en seguridad vial que analizan el tráfico o el uso doméstico de estas para la prevención de accidentes.

## 1.2. Robótica Aérea

Los robots aéreos o más comúnmente conocidos como UAV (*Unmanned Aerial Vehicle*), son vehículos aéreos no tripulados (VANT) controlados remotamente desde una estación de control en tierra y/o mar, o por un programa previamente implementado, estos son los llamados UAV autónomos, programados para que respondan ante el entorno e incluso interactúen con él.

Inicialmente estos robots se pensaron únicamente para uso militar y fueron investigados y desarrollados por este sector, empezaron a crearse entre los años 1914 y 1918, durante la I Guerra Mundial, cuando estaban pensados para utilizarse como blancos aéreos de entrenamiento y defensa contra los Zeppelins, se continuaron desarrollando durante la II Guerra Mundial también para entrenar a los operarios de los cañones antiaéreos pero pronto se vio el potencial que tenían estos robots aéreos y se decidió dejar de utilizar únicamente como blanco aéreo para utilizarlo con fines mas productivos como la vigilancia, iniciada durante la guerra de Vietnam, y la obtención, manejo y transmisión de información ya sea propia u obtenida por los mismos UAV y así poder proteger la misma de la guerra electrónica y la criptografía ya que las comunicaciones son mucho mas seguras y difíciles de detectar.

Hasta este último siglo se pensaba que serían investigados únicamente para la industria militar, pero visto el gran potencial que tenían, los nuevos usos que se les estaban dando y los avances en la industria aeronáutica e informática permitieran que se convirtiese en una herramienta útil para la sociedad civil. En la actualidad los UAV son útiles en diferentes industrias con diferentes objetivos y así una posible clasificación de los mismos sería:

- **Blanco:** Fue el primer uso que se le dio a los UAV y que permitió el posterior avance y desarrollo que ha obtenido. Servían como simulación de aviones y ataques enemigos para poder entrenar las defensas de los ejércitos.

- **Reconocimiento:** Uso que reveló el gran potencial de estos robots, servían para enviar información militar recopilada durante el vuelo, normalmente en las zonas enemigas.



Figura 1.5: AAI RQ-2 Pioneer - Uno de los primeros Robots Aéreos.

- **Combate:** estos son los llamados UCAV(*unmanned combat air vehicle*) usados para llevar a cabo misiones de combate que suelen ser peligrosas.
- **Logística:** o también llamados de carga, utilizados sobretodo para transportar mercancías peligrosas o sobre zonas en conflicto sin el riesgo de perder vidas humanas.
- **Investigación y Desarrollo:** en éstos se prueban y se mejoran los sistemas que están en desarrollo para comprobar su correcto funcionamiento.
- **Comercial y Civil:** última utilización que se ha llevado a cabo y que se encuentra en mayor crecimiento tanto de innovaciones como de ventas ya que son diseñados para propósitos civiles como pueden ser: realizar filmaciones , inspección y reparaciones, sondas de investigación, rescates, vigilancia, detección de incendios y agricultura entre otros.

### 1.2.1. Componentes

La mayoría de los UAV actuales cuentan con una serie de partes, sensores y actuadores que les permiten realizar los propósitos para los que han sido creados, entre ellos encontramos:

- **Motor:** En esta parte diferenciaremos los de ala fija los cuales se rigen por el mismo principio de vuelo que los aviones en los cuales la sustentación se produce gracias a la forma de las alas y el enfrentamiento de estas frente al viento, y los de ala rotatoria, estos se rigen por el principio de vuelo de los helicópteros en los cuales las propias hélices del motor son las que por su forma y giro producen la sustentación y por tanto el vuelo del drone.



Figura 1.6: Drone ala fija Vs Drone ala móvil

- **Chasis:** Estructura principal sobre la que se sitúan el resto de los elementos. Cambiará su forma dependiendo del tipo de UAV, variando la longitud del cuerpo de aterrizaje o el número de soportes para los motores o hélices. Gracias a los últimos avances en la ingeniería de materiales se ha conseguido que el chasis sea mucho más ligero y con ello incrementar la carga útil(*pay-load*) para mejorar la funcionalidad de los mismos.
- **Batería:** La parte más crítica por el momento de los UAV y en la que más se está investigando, la autonomía media de los UAV no supera la hora de vuelo y esto es debido a que los motores necesitan mucha potencia para poder volar los drones. Cuanto más grandes más potencia es necesaria y por tanto mayor batería

se necesita, por lo que hay que encontrar el término justo que se necesita para cada drone en cuanto a peso, carga útil y autonomía.

- **Equipo transmisor y receptor:** Parte encargada la transformación de la información y si fuese el caso de la comunicación simultánea con el equipo en tierra. Es una parte muy importante para los UAV que están teledirigidos o que transmiten información simultanea de lo que están haciendo, normalmente mediante radiofrecuencia o Wi-Fi.
- **Controlador:** Encargado de recoger toda la información tanto de la previamente cargada como de los sensores para procesarla y ejecutar las órdenes adecuadas para completar el objetivo que se le ha establecido.
- **Cámara:** Aunque no todos los UAV la llevan incorporada, es una parte imprescindible en el desarrollo de los mismos ya que permite saber en todo momento lo que está realizando el drone y poder analizar estas imágenes para comprobar el correcto funcionamiento.
- **Altímetro:** Es el sensor que mide los cambios en la presión atmosférica de forma que puede establecer la altura a la que se encuentra el drone.
- **IMU:** es un dispositivo electrónico que mediante una combinación de acelerómetros y giróscopos determina la velocidad, orientación y las fuerzas gravitacionales del vehículo.
- **Magnetómetro:** es un dispositivo que mide la dirección del campo gravitacional de la tierra y de esta forma calcula la orientación del dispositivo con respecto a ésta, lo cual es muy útil para poder direccionar el drone sobre rutas terrestres.

### 1.2.2. Movimiento de cuadricópteros

En este TFG nos centraremos sobretodo en los Robots Aéreos de ala móvil, concretamente en los cuadricópteros que son los que están formados por cuatro

hélices las cuales permiten el movimiento en todas las direcciones posibles mediante la variación de potencia de los motores que hacen girar estas hélices. En la imagen 1.7 se pueden observan cómo se producen los principales movimientos del drone. Las flechas indican el sentido de rotación de las hélices y el color la velocidad del giro, el rojo significa mayor velocidad. Como diferenciación de la forma de vuelo de los helicópteros destacar que en estos la torsión generada por el rotor principal se contrarresta con una hélice de apoyo perpendicular a ésta y en los drones cuadricópteros el problema de la torsión se solventa asignando el sentido de giro de las hélices de forma opuesta entre rotores que están situados de forma contigua.

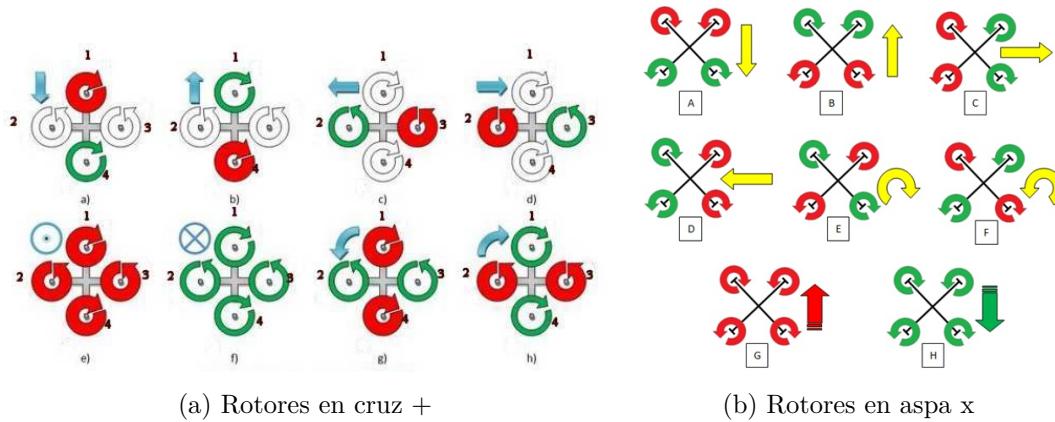


Figura 1.7: Movimiento del Drone según sus rotores.

### 1.2.3. Marco legislativo español para drones

En esta sección nos centraremos principalmente en el marco legislativo en España sobre los UAV, que se acordó el 15 de Diciembre de 2017 en el Consejo de Ministros y se publicó en el BOE [1], y en los avances más importantes que se han llevado a cabo en nuestro país.

La nueva ley sigue cumpliendo prácticamente la totalidad de la ley aprobada el 4 de Julio de 2014 en la cual se especificaba:

- Tipo de Drone: Se establecen dos categorías iniciales: Drones con peso inferior

a 2Kg. y drones con peso entre los 2Kg. y 25Kg. Para ambos es imprescindible disponer de un carnet de piloto de drones para poder operar en España. En caso de los drones de peso inferior a 2kg, no será necesario que estén inscritos en el registro de aeronaves ni disponer de un certificado de aeronavegabilidad. Para ambos tipos de drone, será necesario incluir obligatoriamente una placa identificativa con el nombre del fabricante del aparato así como los datos fiscales de la empresa que lleve a cabo dichas operaciones.

- Espacio aéreo: El espacio aéreo pertenece a AESA, y como tal, para poder realizar cualquier tipo de actividad comercial o civil con un drone, se deberá obtener un permiso oficial, como mínimo 5 días antes de llevar a cabo cualquier operación en el aire. Esta nueva legislación sigue manteniendo la prohibición de sobrevolar núcleos urbanos o espacios con una alta masificación de gente sin el consentimiento especial por parte de la Agencia Española de Seguridad Aérea.
- Seguridad: El pilar fundamental en el que se ha basado el Ministerio para la realización de la normativa de uso de drones civiles en España es la seguridad. Por ello cada empresa deberá disponer de un manual de operaciones cumplimentado siguiendo el estándar proporcionado por el Ministerio, así como un estudio de seguridad de cada una de las operaciones a realizar. Es decir, si alguien piensa en hacer volar un drone al margen de la ley, ya sea con un peso inferior a 2kg, o entre 2kg y 25kg, se expone a sanciones que van entre 3.000€ a 60.000€.
- Carnet de piloto de Drones en España: Para que las empresas puedan operar legalmente los pilotos designados deberán disponer de un carnet oficial para el manejo de drones. Si estos pilotos ya disponen de un título de piloto de avión, ultraligero u otro específico, no será necesario obtener dicha titulación. En caso contrario deberán cursar una serie de exámenes y pruebas oficiales para obtener el carnet oficial de piloto de drones. A día de hoy, no existen academias oficiales bajo la tutela del Gobierno que realicen estos cursos, por eso y mientras se empiezan a impartir estos cursos, será obligatorio demostrar que se dispone de los

conocimientos teóricos y algún tipo de carnet oficial o documento que acredite a los pilotos en el manejo de drones para poder llevar a cabo cualquier operación. Esta normativa temporal sobre drones en España considera los diferentes marcos en los que se podrán realizar los distintos trabajos aéreos y en función del peso de la aeronave. Además, el texto aprobado se completa con el régimen general de la Ley 48/1960, de 21 de julio, sobre Navegación Aérea, y no sólo marca las pautas de operación con este tipo de aeronaves, sino también otro tipo de obligaciones.

Las únicas novedades de la ley puesta en marcha a finales del año pasado son:

- Sobrevolar zonas pobladas: Ésta es una de las medidas más esperadas por el sector. Se podrán realizar vuelos sobre aglomeraciones, edificios y reuniones de personal al aire libre siempre y cuando la masa máxima al despegue de la aeronave no sobrepase los 10kg, mantengamos la aeronave dentro del alcance visual del piloto (VLOS) y no sobrepasemos los 120 metros de altura ni los 100 metros en horizontal con la posición del piloto.
- Vuelos nocturnos: Con la actual ley 18/2014 no se permite realizar vuelos nocturnos con RPAs. Los vuelos nocturnos están recogidos en el borrador de la nueva ley y parece que van a ser permitidos siempre y cuando tengamos la autorización de la Agencia Estatal de Seguridad Aérea. Esto supone además que es necesario contar con un estudio de seguridad.
- Vuelos en espacio aéreo controlado: Por el momento solamente está permitido volar en zonas de espacio aéreo no controlado. Con la nueva ley se podrá volar en espacio aéreo controlado siempre y cuando presentemos los estudios de seguridad correspondiente y tengamos la autorización de AESA.
- Operaciones EVLOS: Con la normativa vigente solamente podemos alejar nuestra aeronave a una distancia máxima de 500 metros en horizontal respecto a la posición del piloto. Con la nueva normativa estarán permitidos los vuelos dentro del alcance visual aumentado (EVLOS). Es decir, estos 500 metros pueden ser

ampliados, siempre y cuando existan observadores intermedios coordinados entre si. En todo momento, al menos uno de ellos debe tener visión directa del vehículo.

En cuanto a los inventos de drones o de usos de estos nos basaremos en los dos certámenes más importantes. A nivel internacional tenemos el Premio *UAE Drones For Good Award* en los Emiratos Árabes en el cual los siguientes proyectos españoles han llegado a ser finalistas: transferir rápidamente órganos de trasplantes desde los centros de donantes, vigilar mejor las zonas verdes para combatir la caza furtiva, controlar la vida salvaje y reducir el riesgo de incendios, ofrecer mejor detección de campo de minas y combatir la propagación de la enfermedad del sueño (Tse-Tse) mediante el control de mosquitos.

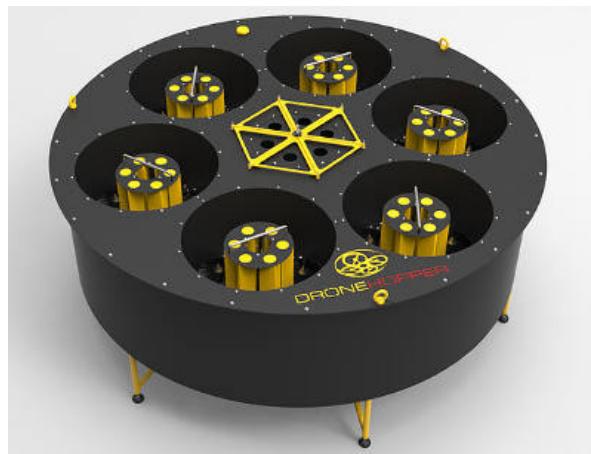


Figura 1.8: Dron contra incendios.

A nivel nacional tenemos el Premio que se ha creado este año a la Innovación Aeronáutica por el COIAE(Colegio Oficial de Ingenieros Aeronáuticos de España) y que ha ganado, dron que extingue incendios forestales, el cual tiene la capacidad de adaptarse a las condiciones de un fuego para apagarlo<sup>1</sup>.

<sup>1</sup><https://www.drone-hopper.com>

### 1.3. Visión Artificial y Autolocalización

Para muchos científicos el sentido del cual el ser humano obtiene más información del medio es la visión. De hecho, debido a esta gran cantidad de datos se calcula que el 70 % de las tareas del cerebro se emplean en analizar esta información visual. La Visión Artificial o Visión por Computador busca conseguir la información visual para extraer las características que le interesan y así poder utilizar procedimientos automáticos. Para conseguir que todo esto fuese viable y no se tardara días en enviar imágenes se emplearon técnicas de procesamiento de imágenes las cuales han avanzado a niveles de que el desfase entre la visión artificial y la realidad sea prácticamente nulo.



Figura 1.9: Visión artificial de un automóvil.

El drone manejado en este TFG incorpora un sistema de autolocalización visual, para entender el contexto en cuanto a visión artificial esta se basa en la naturaleza de la luz, la luz es la parte de la radiación electromagnética que puede ser percibida por el ojo humano y la luz visible corresponde a la radiación del espectro visible, ambas están formadas por fotones cuyas propiedades en relación con la dualidad de onda explican las características de su comportamiento. Una vez conocida

la naturaleza de la luz, la Visión Artificial se basa en la formación de imágenes y, como hemos dicho antes, en el sistema de procesamiento de éstas. El primer apartado estaría constituido por el subsistema de iluminación, de captación de la imagen y de adquisición de la señal en el computador, una vez se ha conseguido introducir la señal en el computador se procesa mediante algoritmos para transformarla en información útil para los robots.

En estos últimos años y gracias a la privación del GNSS y la utilización de este sistema para la sociedad guerra de la industria militar ha permitido que hoy podamos utilizar la localización en cualquier dispositivo con una simple antena, ha su vez y visto el potencial de este sistema de autolocalización todos los países han intentado crear su red satelital de posicionamiento, el primero y promotor de todo fue GPS (EE.UU.), Galileo (UE), Beidu (India), Glonass (Rusia)... Con toda esta red de satélites y la posibilidad de los nuevos softwares que permiten la utilización de varios sistemas conjuntamente, prácticamente todos los dispositivos utilizan un localizador GNSS para posicionarse, el problema viene cuando la posición que necesitamos es tridimensional y tiene que ser muy precisa, ya que este sistema aun cuenta con fallos de metros en cuanto al posicionamiento tridimensional. Por ello, en nuestro proyecto, no podremos utilizar este sistema de posicionamiento y lo cambiaremos por la autolocalización mediante balizas visuales, la cual nos dará un error mucho menor y nos permitirá realizar el enrutamiento y guiado mucho más preciso.

## 1.4. Robótica aérea con JdeRobot

Este TFG se enmarca en el proyecto de software libre para robots JdeRobot. Los antecedentes inmediatos a este TFG en los que me he basado y que me han ayudado para crear mi trabajo han sido:

Alberto Martín <sup>2</sup> [2] Navegación visual en un cuadricóptero para el segui-

---

<sup>2</sup><http://jderobot.org/Amartinflorido-tfm>

miento de objetos. En el abordaba la navegación visual autónoma de un cuadricóptero implementando algoritmos de navegación visual para el seguimiento de objetos de manera autónoma tanto con la cámara frontal como con la cámara inferior.

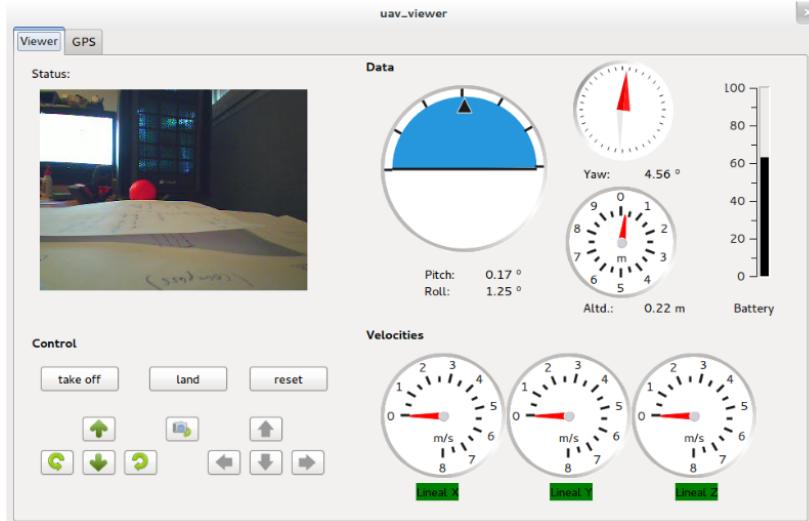


Figura 1.10: TFM Alberto Martín.

Daniel Yagüe<sup>3</sup> [3] Cuadricóptero AR.Drone en Gazebo y JdeRobot. En el cual su objetivo era proporcionar un soporte para robots aéreos en el entorno JdeRobot dentro del simulador Gazebo y crear varias aplicaciones de navegación para validarlos en entornos reales.

Alberto López-Cerón<sup>4</sup> [4] Basado en la autolocalización visual robusta basada en marcadores. Su objetivo principal era crear un algoritmo de autolocalización visual basado en marcadores, es decir, a partir de la detección de balizas. Fue creado tanto para el entorno de simulación como para entornos reales.

Arturo Vélez<sup>5</sup> [5] Dedicado al seguimiento de un objeto con textura desde un dron con cámara. Aquí trabajó en un seguimiento visual, esta vez sin filtro de color,

<sup>3</sup><http://jderobot.org/Daniyague-pfc>

<sup>4</sup><http://jderobot.org/Alopezceron-tfm>

<sup>5</sup><http://jderobot.org/Avelez-tfg>

donde el drone sigue una textura en movimiento detectando unos puntos de interés para reconocerla.

Manuel Zafra<sup>6</sup> [6] Centrado en el seguimiento de rutas 3D por un drone con autolocalización visual con balizas. Diseño un sistema de vuelo autónomo para drones en espacios interiores, basándose en la autolocalización mediante la visión artificial.

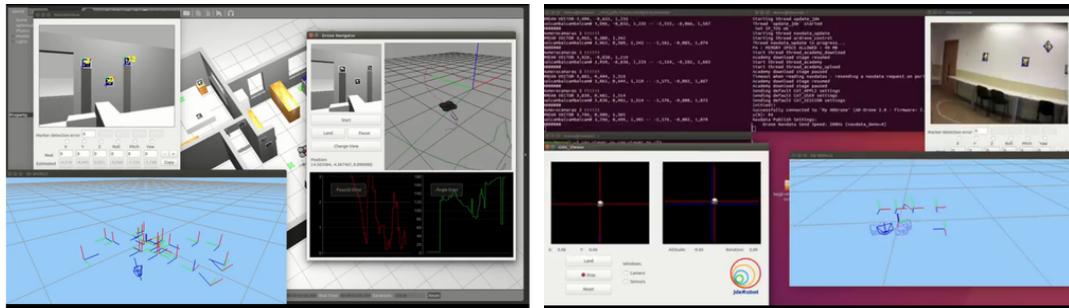


Figura 1.11: TFG Manuel Zafra.

Jorge Vela<sup>7</sup> [7] Aborda el despegue, navegación y aterrizaje visuales de un drone usando jderobot. Se centra en la localización y aterrizaje controlado de un drone mediante la detección visual de balizas.

Una vez expuesta la breve introducción, se va a explicar lo realizado en este TFG. En el capítulo 2 se redactan los objetivos propuestos y la metodología seguida para llegar hasta ellos. En el capítulo 3 se presenta la infraestructura que hemos utilizado. En el capítulo 4 se describe la implementación del algoritmo realizado, cómo se ha programado y cómo se ha integrado junto con los demás algoritmos en una sola aplicación final. En el capítulo 5 se detallan los distintos experimentos realizados, los resultados y los errores cuantificados que se han obtenido. Por último, en el capítulo 6 se incluyen las conclusiones a las que se han llegado tras finalizar este proyecto y los trabajos futuros que se podrían realizar.

<sup>6</sup><http://jderobot.org/MazafraV-pfc>

<sup>7</sup><http://jderobot.org/Jvela-tfg>

# Capítulo 2

## Objetivos

Una vez introducidas las motivaciones que me han llevado a hacer este tfg y expuestos los conceptos mas importantes que se deben saber para comprender la totalidad del trabajo, en este capitulo se van a exponer los diferentes problemas que se abordaran a lo largo de la memoria, los requisitos para poder abordar estos problemas y por ultimo la metodología y el plan de trabajo que se ha seguido.

### 2.1. Problemas a abordar

El objetivo principal de este trabajo es la creación de un sistema que permita el funcionamiento de un dron completamente autónomo con esto queremos decir que despegue de forma controlada, siga una ruta previamente establecida y aterrize tambien de forma controlada. Para la parte de enrutamiento el drone ha de conocer en todo momento su posición en el entorno mediante técnicas de visión por computador basadas en marcadores visuales artificiales, mientras que para el despegue y aterrizaje controlados debe reconocer las balizas por visión cuya posición es desconocida.

Conociendo el objetivo principal este se ha desglosado en varios subobjetivos para poder abordarlo correctamente:

1. **Utilización de la herramienta Visual States:** La cual nos permitirá crear la jerarquía necesaria para poder pasar de un estado a otro mediante transiciones y

así poder dividir nuestro programa e ir mejorándolo e implementando las nuevas creaciones sin tener que modificar las otras partes, además de esto creará una interfaz gráfica en la cual representa el comportamiento del robot gráficamente y así se puede distinguir el estado en el que se encuentra. Esta representación gráfica permite un mayor nivel de abstracción para el usuario, ya que solo tiene que preocuparse por programar las acciones actuales del robot y seleccionar qué componentes puede necesitar de la interfaz del robot.

2. **Adaptación e integración del componente Slam-Visualmarkers:** Como nadie había utilizado la herramienta de Visual States integrando el componente de slam-visualmakers hemos tenido que ajustar este componente e introducirlo por pasos para poder finalmente conseguir su correcto funcionamiento en la misma.
3. **Recodificación e integración del aterrizaje visual:** Ya que este estaba creado a partir de una posición dada en altura y contaba con temporizadores y una maquina virtual de estados. Por esto se ha modificado para poder integrarlo en nuestro sistema de navegación y además a partir de este se ha creado el propio sistema de despegue controlado.
4. **Desarrollo de dos componentes de navegación basados en la posición absoluta del drone:** Ambos serán controles de pilotaje aplicados sobre las posiciones obtenidas por el componente slam-visualmarkers, la diferencia radicara en que a uno le estableceremos una serie de puntos o balizas a los cuales debe de llegar mientras que el otro deberá seguir una ruta.
5. **Validación experimental en entorno simulado Gazebo:** Se creará un mundo tridimensional en el que se realizarán las pruebas pertinentes para determinar la robustez del sistema en conjunto. Se examinará para ambos componentes de navegación el error de la estimación de posición y cómo este puede afectar al sistema de pilotaje del cuadricóptero, ademas se comparara con anteriores sistemas de pilotajes y se verán las posibles mejoras. También

se calculará el error producido en la estimación de posición por el componenete Slam-Visualmakers.

## 2.2. Requisitos

Una vez descritos todos los objetivos, lo siguiente es nombrar los requisitos que va a satisfacer la aplicación que hemos desarrollado:

- El algoritmo funcionará en la plataforma de desarrollo JdeRobot-5.6.3.
- El sistema se ejecutara en el entorno GNU/Linux Ubuntu 16.04.
- La aplicación se ejecutara en el simulador Gazebo con la utilización del robot Ar.Drone.
- El sistema debe ser exportable a cualquier escenario que sea un espacio simulado controlado y contenga marcadores AprilTags con posiciones conocidas.
- Para la autolocalización, el sistema sólo puede depender de las imágenes servidas por la cámara del drone y utilizando el componente Slam-Visualmarkers.
- El control de navegación mediante el pilotaje debe ser suave para no perder los marcadores y balizas, pero tambien robusto, fluido y vivaz para que el drone se mueva de forma ágil y veloz por el entorno y sus rutas establecidas.
- Programado en Python 2.7.

## 2.3. Metodología

Al tratarse de un trabajo de integración y desarrollo software el modelo de trabajo que se ha seguido ha sido el de desarrollo en espiral. Este modelo nos ha permitido trabajar de forma progresiva, es decir, empezando por las partes mas sencillas y a medida que las íbamos logrando ir avanzando hasta las mas complejas. Para ello

la forma de conseguirlo fue mediante la marcación de hitos o pautas a alcanzar que se revisaban periódicamente mediante reuniones con el tutor, de esta forma se analizaban si se había conseguido llegar al resultado esperado en cada etapa y así poder pasar al siguiente objetivo analizando para ello las posibles rutas que se podían seguir según la toma de decisiones y la evaluación de riesgos. Cuando se alcanzaba un objetivo lo suficientemente importante se creaba una entrada en el cuaderno de bitácora de JdeRobot, el cual es público y nos ha servido para llevar un seguimiento de las áreas realizadas y como punto de comunicación con el tutor para saber las dificultades que teníamos y como poder solucionarlas.

En el siguiente enlace del mediawiki o cuaderno de bitácora mencionado anteriormente se pueden encontrar desde fotos y videos de la práctica final como resultados y pasos iniciales que se hicieron para adentrarse en el mundo de la programación de robots y drones <http://jderobot.org/Jsaize-tfg>. También se puede visitar el código de todas estas etapas que se observan en el mediawiki, este es de acceso público en GitHub <https://github.com/RoboticsURJC-students/2017-tfg-jesus-saiz>.

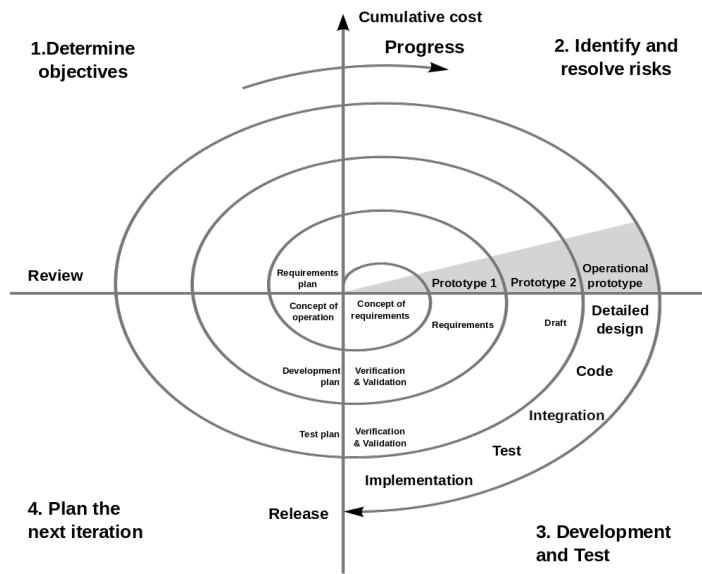


Figura 2.1: Representación del desarrollo en espiral.

## 2.4. Plan de trabajo

Para conseguir la metodología de trabajo expuesta anteriormente se ha seguido una planificación dividida en las siguientes fases:

- **Programación en Python:** Necesaria para poder entender el código anterior en el que me he basado y crear el mio propio. Con el conocimiento de c/c++ y una serie de guías de ayuda fue el primer paso de aprendizaje.
- **Formación en JdeRobot:** Había que comprender el funcionamiento de la plataforma y saber todas las herramientas con las que contaba para saber en que te podías basar y que era lo que tenias que desarrollar tu mismo. Para ello se realizaron una serie de programas simples en los cuales trabajabas con distintos robots y con diferentes herramientas, teniendo así los primeros contactos con programación de robots simulados y reales.
- **Familiarización con Gazebo:** El simulador utilizado preferentemente en JdeRobot, en el hubo que crear el entorno de simulación sobre el cual se basaría el programa de vuelo del drone.
- **Estudio y desarrollo del interfaz gráfico:** El programa se ha desarrollado sobre la aplicación Visual States, la cual he tenido que aprender para conseguir introducir todas las etapas y que funcionase correctamente.
- **Aprendizaje e integración del componente de autolocalización:** Necesario para conocer el funcionamiento de este componente y poder manejar los parámetros y las balizas con las que trabaja, para así poder conseguir una correcta autolocalización del drone y tambien obtener los errores de la herramienta y extraerlos de los errores de pilotaje.
- **Adaptación y desarrollo del control de aterrizaje:** Para poder adaptarlo a la aplicación de Visual States y a su vez crear un propio sistema de despegue del drone.

- **Desarrollo del algoritmo para el control de pilotaje:** De dos formas diferentes, por un lado el algoritmo de seguimiento de puntos y por otro lado el de seguimiento de trayectorias. Ambos capaces de gobernar el movimiento del drone para seguir las rutas en 3D.
- **Validación experimental en entornos simulados:** Para comprobar y validar el correcto funcionamiento de las fases anteriores se realizaron una serie de experimentos en entornos simulados, con estas pruebas se detectaron posibles comportamiento erróneos que corrigiéndolos se consiguió estabilidad en el sistema y una posible viabilidad en entornos reales.

# Capítulo 3

## Infraestructura

Una vez vistos los objetivos de este trabajo, en este capítulo se mostraran las diferentes infraestructuras sobre software en el que nos hemos apoyado para la creación de este TFG. También se explicara el funcionamiento de los distintos componentes que lo forman.

### 3.1. Entorno JdeRobot

JdeRobot<sup>1</sup> es un paquete de software libre para desarrollar aplicaciones de robótica y visión por computación. Estos dominios incluyen sensores como cámaras, actuadores y software inteligente en el medio. Está escrito principalmente en los lenguajes C++ y Python, y proporciona un entorno de programación basado en componentes. Pueden ejecutarse en diferentes ordenadores y se conectan mediante el middleware de comunicación ICE o los mensajes ROS. Los componentes interoperan a través de interfaces explícitas. Es el software principal con le que he trabajado y esta mantenido por el grupo de robótica de la Universidad Rey Juan Carlos.

---

<sup>1</sup>[http://jderobot.org/Main\\_Page](http://jderobot.org/Main_Page)

### 3.1.1. Herramienta ColorTuner

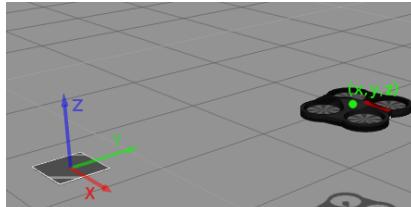
Es una aplicación para configurar filtros de color personalizados en espacios de color HSV, RGB o YUV, y así poder obtener la gama de color que nos interesa o realizar un filtro sobre una imagen o vídeo. Utiliza un interfaz ICE donde se representan dos imágenes, una la real y otra la filtrada, en esta se pueden variar los parámetros para ver que colores cumplen las características, quedándose los otros en negro.

### 3.1.2. Interfaz Pose3D

Pose3D es la interfaz que se emplea en JdeRobot para obtener la posición y orientación en un espacio 3D de un objeto, en nuestro caso un drone. Se implementa mediante la función *Pose3Ddata* y esta compuesta por un punto en 3D el cual indica la situación de un objeto mediante la posición en coordenadas cartesianas y la orientación mediante los cuaterniones, a partir de estos podremos obtener los ángulos de roll, pitch y yaw.

Pose3DData

```
{
    float x; /* x coord */
    float y; /* y coord */
    float z; /* z coord */
    float h; /* */
    float q0; /* qw */
    float q1; /* qx */
    float q2; /* qy */
    float q3; /* qz */
};
```



(a) Datos de Pose3DData

(b) Representación de los datos

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan \frac{2(q_0 q_1 + q_2 q_3)}{1 - 2(q_1^2 + q_2^2)} \\ \arcsin(2(q_0 q_2 - q_3 q_1)) \\ \arctan \frac{2(q_0 q_3 + q_1 q_2)}{1 - 2(q_2^2 + q_3^2)} \end{bmatrix}$$

(c) Matriz de Conversión Angular

Figura 3.1: Ejemplo de Pose3DData

### 3.1.3. Plugin ArDrone2 en Gazebo

Hemos empleado para el desarrollo del proyecto el ArDrone 2.0 o mas bien su modelo en el entorno de simulación Gazebo. Este plugin implementa un control realista y optimizado en simulación del drone, por lo que nos permite trabajar con los distintos sensores (cámara, IMU, sonar...).

### 3.1.4. Práctica Navegación por control de posición

De esta práctica se ha conocido el funcionamiento del drone en un entorno simulado y así saber como poder manejarlo mediante un controlador de velocidades simple. Este control es el interfaz CMDVel el cual nos permite enviar información de velocidades al drone para poder guiarlo dentro del entorno de simulación Gazebo.

Esta práctica se encuentra en el GitHub de Jderobot Academy [https://github.com/JdeRobot/Academy/tree/master/src/position\\_control](https://github.com/JdeRobot/Academy/tree/master/src/position_control)

### 3.1.5. Práctica drones Gato-Ratón

El objetivo de esta práctica era utilizar algunas de las capacidades del drone junto con las herramientas de JdeRobot. Desde filtros de color obtenidos mediante ColorTuner hasta interfaces como Pose3D o CMDVel. Con todo esto esto se puede control de forma remota el drone y ver el estado de sus sensores. Una vez realizada esta práctica se pudo observar las capacidades completas del drone y como podíamos desarrollar nuestro algoritmo.

Esta práctica se encuentra en el GitHub de Jderobot Academy [https://github.com/JdeRobot/Academy/tree/master/src/drone\\_cat\\_mouse](https://github.com/JdeRobot/Academy/tree/master/src/drone_cat_mouse)

## 3.2. Gazebo

El simulador Gazebo es un programa open source distribuido bajo la licencia Apache 2.0 que se utiliza sobretodo para la investigación en robótica e Inteligencia Artificial.

Este simulador ofrece la capacidad de simular de una forma eficiente y precisa cualquier tipo de robot en entornos complejos tanto de exterior como de interior. Sus características principales son sus motores de físicas, el motor de renderizado avanzado, el repositorio con la mayoría de robots comerciales y una gran gama de sensores y cámaras que permiten simular la mayoría de entornos reales.

Al tratarse de un programa OpenSource su comunidad crece a diario lo que permite que cada vez existan más plugins, esto unido a la fácil integración en ROS e ICE permite que podamos tener el software base para simular los robots reales.

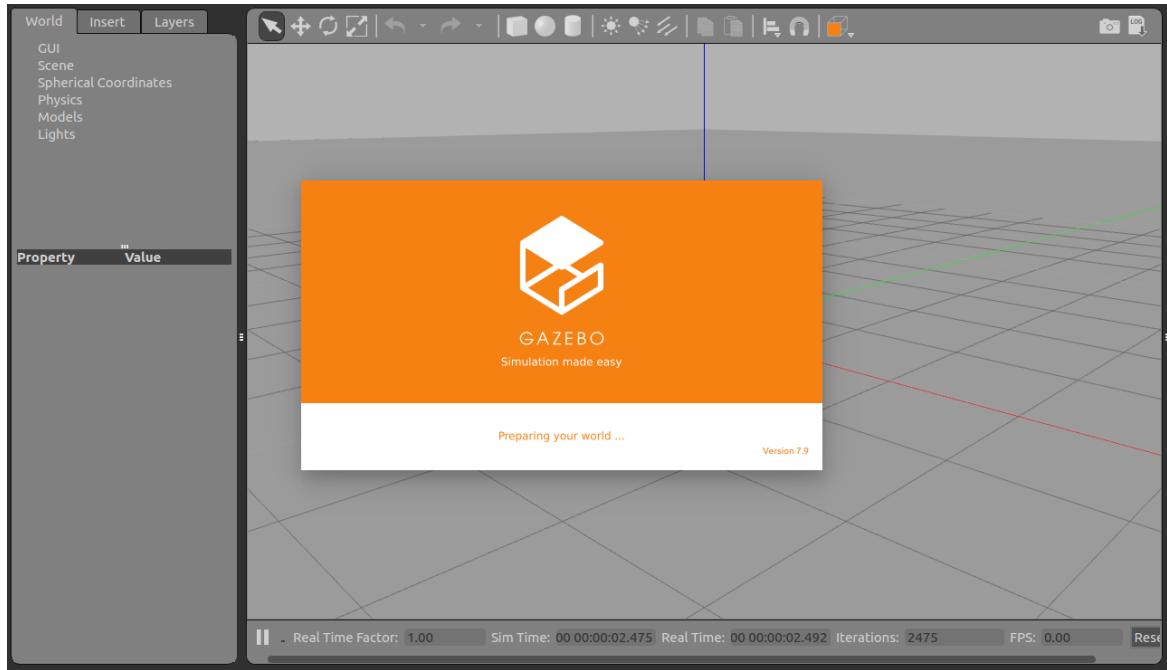


Figura 3.2: Entorno de Simulación Gazebo.

En nuestro TFG hemos trabajado con la versión Gazebo 7.9. Los mundos creados en esta aplicación se definen con la extensión '.world' y escritos mediante SDF (*Simulation Description Format*). Para este trabajo he creado el *ArDones2.world* el cual utiliza el plugin ArDones2, junto con la estructura de una casa y diversas balizas *AprilTags*.

### 3.3. Visual States

VisualStates es una herramienta para la programación de comportamientos de robots que utilizan máquinas de estados finitos jerárquicas (*HFSM - Hierarchical*

*Finite State Machine*). Representa el comportamiento del robot gráficamente en una hoja en blanco compuesta por estados y transiciones. Cuando el autómata está en un cierto estado, pasará a otro dependiendo de las condiciones establecidas en las transiciones. Esta representación gráfica permite un mayor nivel de abstracción para el usuario, ya que solo tiene que preocuparse por programar las acciones actuales del robot y seleccionar qué componentes puede necesitar de la interfaz del robot.

Esta herramienta cuenta con dos interfaces de usuario diferentes: la GUI visualStates y la GUI de tiempo de ejecución de python. La GUI de visualStates permite el diseño de autómatas y la GUI de tiempo de ejecución de python proporciona una visualización del autómata en ejecución.

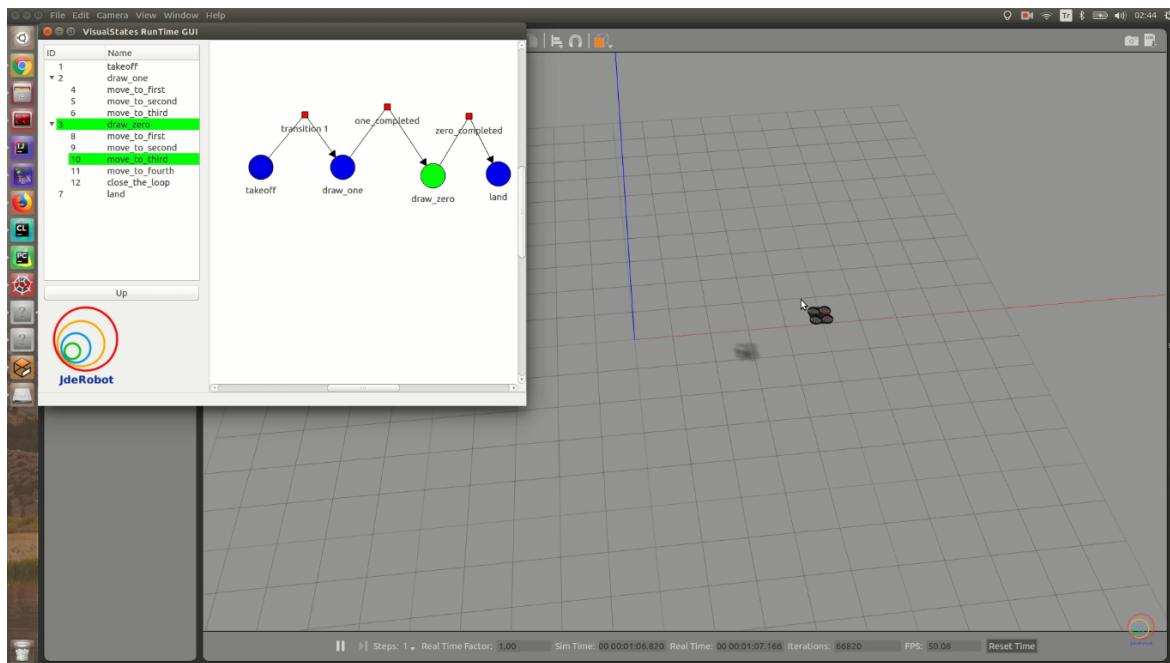


Figura 3.3: Herramienta Visual States sobre Gazebo

### 3.4. Balizas visuales AprilTags

AprilTags [8] es un sistema de visión por computador que permite detectar balizas visuales contenidas en una imagen. Es útil en una amplia variedad de tareas que incluyen la realidad aumentada, la robótica y la calibración de cámaras.

Las balizas se basan en el concepto de los códigos QR, aunque éstas están diseñadas para contener muchos menos bits de información(4-12 bits). Además, presenta un nuevo sistema de codificación que aborda problemas específicos de los códigos de barras 2D como es la robustez frente a la rotación y a los falsos positivos que pueden dar las imágenes naturales. EL software de detección AprilTag calcula la posición, orientación e identidad 3D precisa de las etiquetas en relación con la cámara, además de su correspondiente ID. Eso lo realiza mediante un algoritmo de segmentación basado en gradientes locales que consigue que las líneas se estimen con precisión, el cual se implementa en C sin dependencias externas. Esto provoca una tasa de falsos negativos muy bajos, aunque aumenta la probabilidad de falsos positivos. Sin embargo, gracias a la codificación de las balizas esta probabilidad es reducida hasta niveles aceptables.



Figura 3.4: Ejemplos de AprilTags.

### 3.5. Biblioteca OpenCV

OpenCV<sup>2</sup> es una biblioteca *open-source* de visión artificial que se desarrolló inicialmente por Intel, tiene un conjunto de funciones que van desde sistemas de seguridad con detección de movimiento hasta centros de procesamiento con

---

<sup>2</sup><https://opencv.org/>

reconocimiento de objetos. Está programada en C/C++ y en nuestro TFG para el procesamiento de imágenes nos hemos apoyado principalmente sobre ésta, trabajando en la versión 3.4.

Gracias a OpenCV, al obtener la imagen que transmite el drone se puede tanto detectar objetos como aplicar cambios sobre ella, para marcar las zonas de interés o diferentes objetos. Permite la realización de filtros de color, para eliminar objetos no deseados dependiendo el momento. Además permite el uso de operadores morfológicos (erosión y dilatación) gracias a los cuales se evitan imperfecciones en las imágenes como puede ser el ruido, que clasifica objetos inexistentes o de no interés como objetos de interés.

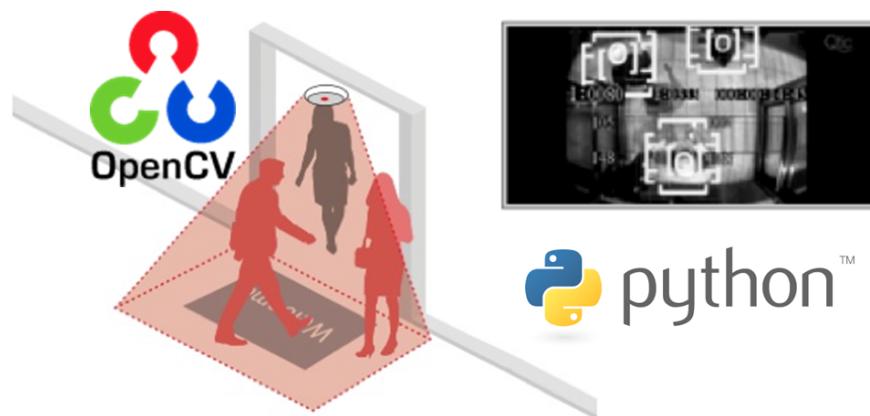


Figura 3.5: Ejemplo de algoritmo con OpenCV.

## 3.6. Slam-Visualmakers

Como su propio nombre indica SLAM (*Simultaneous Localization and Mapping*) algoritmos capaces de generar un mapa y localizar al sensor de forma simultánea a partir de balizas visuales, *Visualmarkers*. Es una aplicación desarrollada por Felipe Pérez Molina<sup>3</sup> en su Trabajo Fin de Máster que ha ido creando paralelamente

<sup>3</sup><http://jderobot.org/F1perez-tfm>

a este TFG. Su trabajo esta desarrollado en la plataforma JdeRobot y está programada en C++.

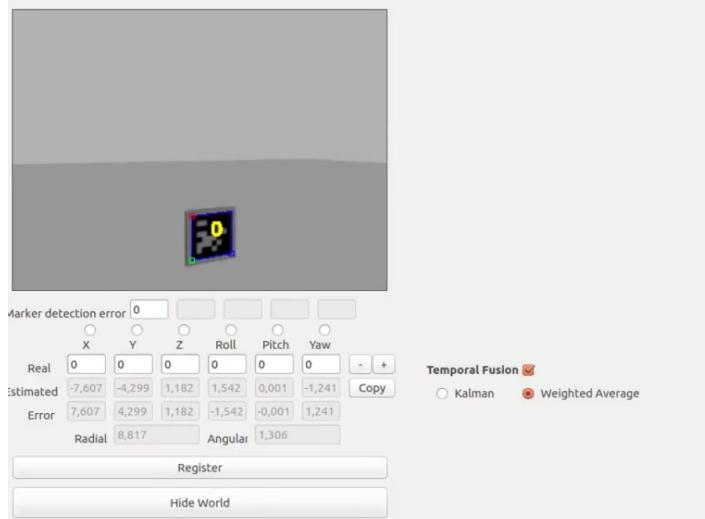


Figura 3.6: GUI de Slam-Visualmarkers.

Esta aplicación es un desarrollo de Cam-Autoloc, pero con un algoritmo de estimación de posición mas preciso, un interfaz gráfico propio, eliminando así la dependencia de QtCreator's, y la eliminación de la librería Aruco para su ejecución.

Para su correcto funcionamiento el algoritmo necesita tres entradas de datos: primero, se sirve de las imágenes recibidas a través de una interfaz creada con ICE y devuelve la posición estimada mediante un objeto Pose3D; segundo, un fichero de texto que contiene una lista donde figuran el identificador, posición y orientación 3D de cada baliza visual ubicada en el entorno; y tercero, otro fichero, este de configuración, que contiene toda la información de los parámetros de la cámara utilizada.

Para estimar la posición, el algoritmo comienza analizando la imagen recibida mediante las librerías OpenCV y AprilTags para explorar la imagen en 2D en busca de las balizas. Una vez localizadas, se hace uso de la librería Progeo para calcular la posición y orientación en tres dimensiones de la cámara con respecto a cada marcador. Finalmente, se realiza un proceso de fusión temporal y fusión espacial de las estimaciones obtenidas a partir de cada baliza. La aplicación permite elegir qué

clase de filtro temporal utilizar, pudiendo escoger entre un filtro por pesos o un filtro Kalman.

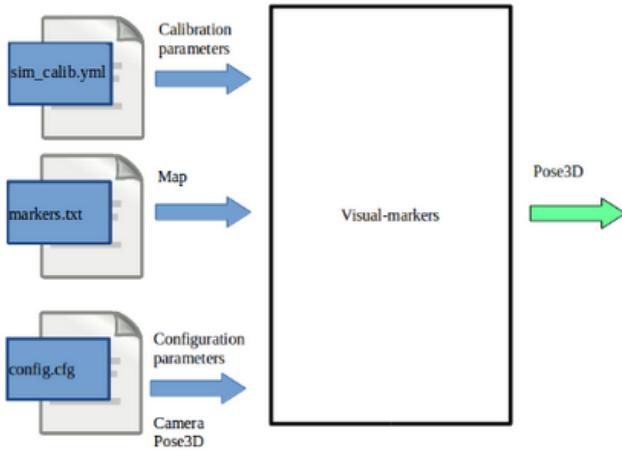


Figura 3.7: Interfaces Slam-Visualmarkers.

## 3.7. NumPy

Como hemos dicho este TFG está escrito en el lenguaje de programación Python y NumPy es una extensión *open-source* de este lenguaje. Es el paquete fundamental para la informática científica con Python. Contiene, entre otras cosas un poderoso objeto de matriz N-dimensional, herramientas para integrar el código C/C++ y Fortran, álgebra lineal útil, transformadas de Fourier y capacidad de trabajo sobre números aleatorios.

Además de sus usos científicos obvios, NumPy también se puede usar como un contenedor multidimensional de datos genéricos. A su vez se pueden definir tipos de datos arbitrarios, lo que permite a NumPy integrarse de manera rápida y sin problemas con una amplia variedad de bases de datos. NumPy está licenciado bajo la licencia BSD (*Berkeley Software Distribution*), lo que permite su reutilización con pocas restricciones.

# Capítulo 4

## Desarrollo

En este capítulo se describen los pasos seguidos para lograr una solución a los objetivos planteados, utilizando la infraestructura mencionada anteriormente y como se ha implementado.

La solución al problema ha sido un algoritmo de navegación sobre el cual se dará una visión global. A continuación se explicara en detalle el diseño y el funcionamiento de cada uno de los componentes utilizados. Por último, se detallara como ha sido el proceso de integración y como ha ido evolucionando este hasta llegar al objetivo final.

### 4.1. Diseño

El objetivo de este algoritmo es que permitir al drone realizar un comportamiento completamente autónomo desde el despegue, hasta el aterrizaje, ambos controlados, pasando por el seguimiento de una ruta previamente definida. Todo esto basándose únicamente en balizas de apoyo visual. Por lo que estaríamos hablando del vuelo completamente autónomo de un drone mediante visión artificial y control de posición.

En la aplicación final se diferencian dos partes principales: por un lado tenemos el componente encargado de estimar la posición mediante algoritmos de visión

por computador y por otro lado tenemos el componente que se encarga del control del drone tomando las decisiones del movimiento segun la etapa en la que se encuentre.

En el esquema 4.1 se puede ver una explicación de las entradas y salidas de flujos de información. También se pueden observar los ficheros que son imprescindibles en cada modulo para su correcto funcionamiento. Esta imagen nos da una idea del funcionamiento global de la aplicación y se puede observar como toda la comunicación entre procesos se lleva a cabo mediante ICE. A continuación vamos a explicar el comportamiento de cada módulo de una forma breve.

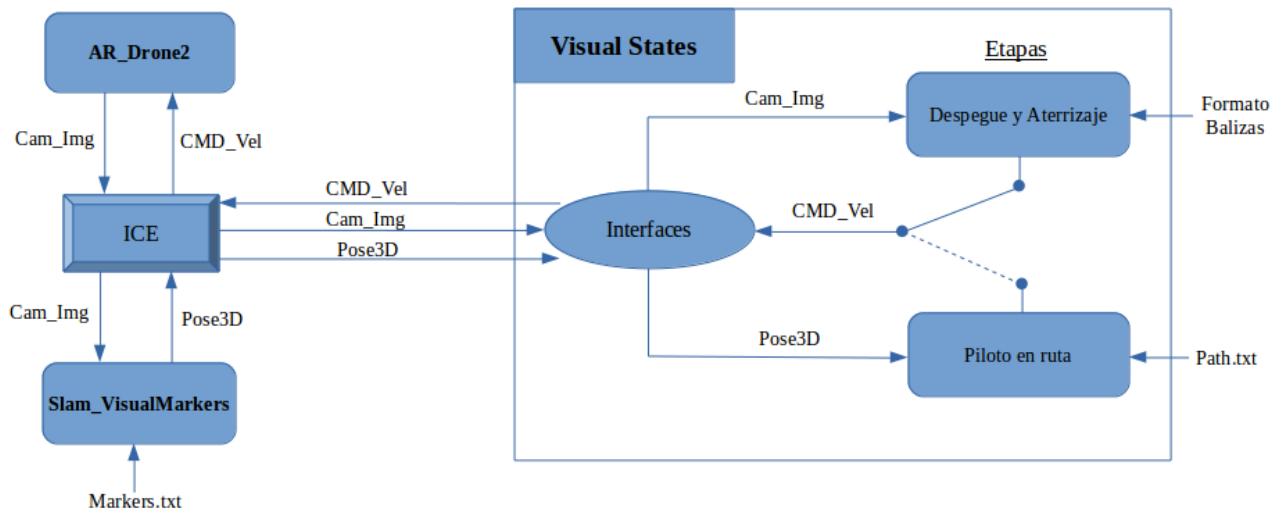


Figura 4.1: Esquema representativo de la aplicación.

El módulo Slam VisualMarkers recibe las imágenes que le proporciona el drone y envia los datos en forma de Pose3D( $x,y,z,q0,q1,q2,q3$ ) a VisualStates. Este componente comienza analizando la imagen recibida en busca de la presencia de marcadores AprilTags. Si no los encuentra devuelve el numero 0 como indicador de ello, en cambio si encuentra alguna envía cuantas ha encontrado y aplica cálculos de geometría proyectiva para estimar la posición de la cámara con respecto a cada una de las balizas. A continuación realiza una fusión espacial, aplicando un filtro basado en pesos y una fusion temporal, mediante un filtro de Kalman. Finalmente, la estimación

calculada se envía al componente de navegación mediante una interfaz ICE.

El módulo de Slam VisualMarkers recibe la imagen de la cámara del drone y las posiciones estimadas, envía estos datos a las etapas que lo necesiten y según en la etapa en la que se encuentre genera una serie de órdenes de velocidades que envía al drone vía interfaz ICE para conseguir el objetivo que según la etapa puede ser despegar, aterrizar o seguir una ruta.

A continuación vamos a explicar cada modulo con mayor profundidad, nuestro mayor desarrollo en este TFG fue la creación de un algoritmo de pilotaje que mejorase los anteriores tanto en precisión de seguimiento de rutas como en tiempo de realización de estas rutas, por lo que sera el apartado en el que mas nos centraremos. Sin embargo, al ser un TFG de integración también explicaremos los módulos en los que nos hemos basado y hemos resintonizado para su correcto funcionamiento en el algoritmo final.

## 4.2. Módulo de Control

Es la parte ne la que mas nos hemos centrado y que mas hemos conseguido desarrollar dentro del algoritmo, ya que para nosotros es la parte mas importante. Se podría decir que se trata de un piloto el cual realiza las tareas de pilotaje del drone. Hemos dividido el pilo segun los datos que recibe de la interface de VisualStates.

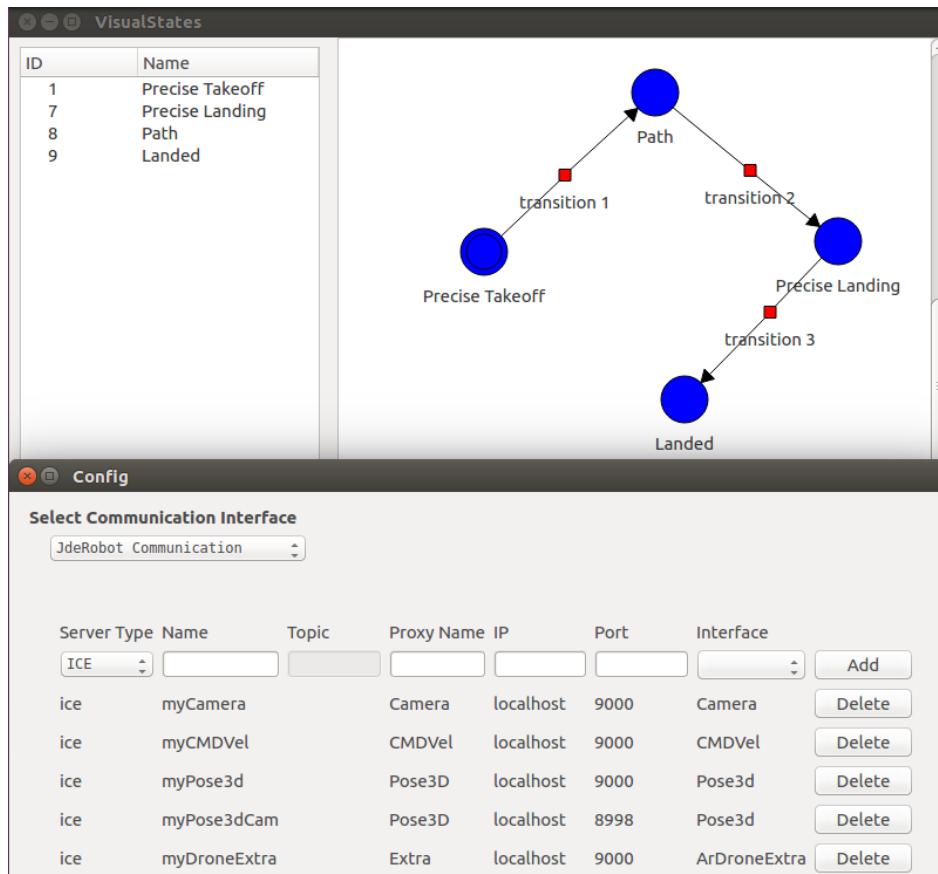


Figura 4.2: Esquema de los componentes de Visual States.

#### 4.2.1. Componente de control de pilotaje

Como se puede observar en la imagen 4.2 el algoritmo de este componente estaría en el estado de Path, el cual se encarga de seguir una trayectoria. Hemos desarrollado dos algoritmos diferentes según el tipo de trayectoria que se le introduzca, si es una trayectoria de puntos separados en la cual el drone solo tiene que alcanzar esos puntos se trataría del **Piloto1** si en cambio la trayectoria es una ruta de puntos prácticamente continuos y en la cual queremos que el drone mire hacia el siguiente punto al que se dirige entonces se trataría del **Piloto2**. Ambos algoritmos se basan en la posición tridimensional del drone para ello hemos utilizado los tipos de datos

de JdeRobot *Pose3D* los cuales nos devuelven tanto la posición del drone como su orientación en el espacio mediante los cuaterniones o los Ángulos de Euler según lo que necesitemos. Estos datos serán proporcionados por la estimación de posición de Slam VisualMarkers. Y nuestro algoritmo devolverá una serie de velocidades que irá enviando al cuadricóptero mediante la función *CMDVel()*.

Por su sencillez de pilotaje vamos a explicar primero el **Piloto1**. Este control de pilotaje se centra en el movimiento direccional únicamente, es decir, para que el drone alcance el punto al que debe llegar no variará prácticamente ninguno de sus ángulos de Euler, tan solo se moverá sobre los ejes x,y,z.

Como el movimiento será únicamente direccional en coordenadas cartesianas tan solo hay que calcular el vector que apunta desde la posición del cuadricóptero hasta el siguiente punto de ruta que se desea alcanzar:

$$\begin{aligned} Pose3D &= (P_x, P_y, P_z) \quad ; \quad Beacon = (B_x, B_y, B_z) \\ \vec{V}_x &= P_x - B_x \quad ; \quad \vec{V}_y = P_y - B_y \quad ; \quad \vec{V}_z = P_z - B_z \end{aligned}$$

Una vez calculado este vector entre las posiciones lo multiplicaremos por un coeficiente regulador que reducirá los valores hasta que estos se encuentren dentro del rango de velocidades del drone y una vez alcanzado este rango se pasaran los valores por una función que comprueve que las velocidades máximas no exceden las permisibles por Slam VisuaMarkers para la detección y análisis de balizas:

```
if math.fabs(xVel) > MaxVx :  
    xVel = MaxVx * np.sign(xVel)  
if math.fabs(yVel) > MaxVy :  
    yVel = MaxVx * np.sign(yVel)  
if math.fabs(zVel) > MaxVz :  
    zVel = MaxVx * np.sign(zVel)
```

Con este método de control de navegación por posición nos aseguraremos que el drone alcanza el punto que queremos adecuadamente y a medida que se va acercando a él, para que la aproximación sea exacta, al reducirse el vector se reducirá la velocidad

alcanzando la baliza con total exactitud. Una vez se alcanza la baliza se busca cual es la siguiente en la lista y se vuelve a realizar el proceso, así hasta alcanzar todas ellas.

En cuanto al **Piloto2** lo primero fue crear las rutas que posteriormente seguiría el drone, como estas debían ser largas con giros y con puntos muy consecutivos se creó en python la función que nos permitiese crear estas según la separación entre puntos de ruta que considerasemos necesaria:

```
i=0
a=0
if i == 0:
    archivo = open( ' pos.txt ', 'w' )
    archivo.write(' x      y      z      roll     pitch     yaw \n')
    i = i+1
b = time.clock()
pos_sim = self.pose.getPose3d()
if ((b - a) > time_write):
    archivo.write(str(pos_sim.x)+ ' ')
    archivo.write(str(pos_sim.y)+ ' ')
    archivo.write(str(pos_sim.z)+ ' ')
    archivo.write(str(pos_sim.roll)+ ' ')
    archivo.write(str(pos_sim.pitch)+ ' ')
    archivo.write(str(pos_sim.yaw)+ ' ')
    archivo.write(str(b)+'\n')
    a = b
```

Una vez obtenida la ruta y a diferencia del **Piloto1** este seguirá los puntos orientándose siempre en la dirección entre la posición y el siguiente punto, para ello tendremos que variar la velocidad angular en torno al eje Z, modificando por tanto el ángulo de yaw del drone y de esta forma conseguir orientarlo de la forma más rápida hasta el siguiente punto de la ruta. Las primeras versiones del piloto tan solo se iban a tener en cuenta las velocidades lineales correspondientes a los ejes X y Z del

drone, descartando la velocidad en Y debido a que modificando yaw no sería necesario el movimiento lateral. Pero con forme se fue investigando mejorando esta vesion se descubrió que permitir al drone realizar pequeñas variaciones en la velocidad lineal del eje Y permitía un aproximación a los puntos mucho mas efectiva y precisa, por lo que se optó por incluir también la variación de la velocidad en el eje Y. Con la ayuda del piloto anterior y teniendo en cuenta las mejoras anteriores se va a explicar el algoritmo final de pilotaje.

Para el calculo de las velocidades lineales se seguiran los mismos pasos que en el **piloto1** pero variaremos los coeficientes de corrección para que las velocidades lineales sean parejas a la nueva velocidad angular. Esta velocidad angular se calculara mediante las funciones de predicción de posición.

Para ello el primer paso es calcular la distancia horizontal que recorrerá el drone hasta la siguiente iteración del algoritmo. La distancia se obtiene multiplicando la velocidad horizontal obtenida anteriormente por el tiempo que transcurre entre dos iteraciones:  $d_\psi = v_x * \Delta_t$ .

Para predecir la posición del cuadricóptero debemos tener en cuenta el ángulo de giro con respecto al eje Z del drone en ese instante. Ya que en el estándar de Pose3D la orientación se indica en cuaterniones, haremos un convesión para conocer el angulo de Euler mediante la siguiente ecuación:

$$\Psi_z = \arctan^2 \left( \frac{2 * q_0 * q_3 + q_1 * q_2}{1 - 2 * (q_2^2 + q_3^2)} \right)$$

Una vez obtenido el ángulo  $\Psi_z$  sacamos la posición que predecimos solo en X e Y ya que la variación en Z no influye en cálculo del ángulo de giro de yaw.

$$X_f = d_\psi * \cos \Psi_z + x_{pose}$$

$$Y_f = d_\psi * \sin \Psi_z + y_{pose}$$

Ahora calcularemos el error de ángulo, que es la diferencia entre el ángulo actual y el ángulo sobre el eje Z existente entre el drone y el punto de ruta,  $\Psi_e$ . Teniendo

el ángulo se calcula una velocidad angular a partir de la distancia horizontal al punto,  $d_H$ , y la velocidad lineal actual,  $v_x$ .

$$\Psi_{path} = \arctan\left(\frac{V_y}{V_x}\right) \quad ; \quad \Psi_e = \Psi_{path} - \Psi_z$$

$$d_H = \sqrt{V_x^2 + V_y^2} \quad ; \quad \omega_e = \frac{\Psi_e}{d_H/v_x}$$

La velocidad angular dependerá entonces de la velocidad angular calculada y de un factor de corrección que viene dado por la relación entre el error lateral predicho  $L_{fe}$  y la velocidad horizontal. Este factor está multiplicado por una constante  $K_g$ , que es la ganancia de corrección del giro.

$$L_{fe} = \cos\Psi_z * (Y_{path} - Y_f) - \sin\Psi_z * (X_{path} - X_f)$$

$$\omega_z = \omega_e + K_g * (L_{fe}/v_x)$$

Por último, una vez obtenida la velocidad angular  $\omega_z$  ajustaremos la velocidad en el eje Y,  $v_y$ , a partir de esta. Una vez obtenidas las tres velocidades que se van a enviar al drone, las velocidades horizontales  $v_x$   $v_y$ , la velocidad vertical  $v_z$  y velocidad angular  $\omega_z$  se envían al drone mediante una llamada al método `SendCMDVel(vx,vy,vz,wz)` de Interfaces, que se ocupa de mandar las órdenes decididas al cuadricóptero.

#### 4.2.2. Componente de control de aterrizaje y despegue

Para este componente hemos utilizado el trabajo que realizó Jorge Vela en su TFG [7] refactorizandolo y acoplándolo dentro de nuestro módulo de pilotaje en Visual States. El diseño de este algoritmo es un proceso basado en adquisición-procesado-envío de ordenes. La adquisición de los datos se realizan mediante los sensores del drone. Estos datos sensoriales serán recogidos para su procesamiento y tras esto se enviarán las instrucciones al drone para que las ejecute. El sensor utilizado en este estado ha sido la cámara, y a diferencia del estado anterior, los movimientos del drone dependerán de lo que esta capte en cada momento.

El lugar de aterrizaje del drone es una baliza previamente definida 4.3. Esta baliza es un cuadrado que en su interior tiene cuatro cuadrantes, dos verdes, y dos azules. Esta baliza se diseñó así para que sea difícil confundirla con otro objeto, pues de ser una baliza simple se podrían confundir los colores, además lo que busca el software será la cruceta que forman estos cuatro cuadrados y el punto central de ésta.

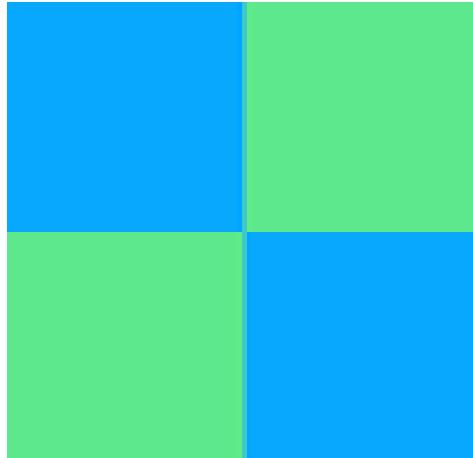


Figura 4.3: Baliza utilizada en Gazebo.

En cuanto al **despegue** se sitúa el drone sobre una baliza sobre la cual tiene que estabilizarse. De esta forma, al despegar detecta esta y trata de centrarse, evitando así que se desvíe por factores externos y quedándose en la situación correcta y pasando al siguiente módulo que sería el del piloto.

Una vez el cuadricóptero termina la ruta establecida, este procede a la siguiente etapa que es el **aterrizaje**, el cual se divide en dos partes, primero la búsqueda de la baliza de aterrizaje y luego el centrado de la baliza y la aproximación a esta.

En cuanto a la búsqueda, sigue una navegación en espiral de forma que irá rastreando la zona ampliando su giro de forma continua, hasta que detecte una baliza. Si no se detecta una baliza, continuará el algoritmo de búsqueda en el punto donde se había quedado, aumentando con la amplitud de las espirales a la que se había llegado. Cuando se detecte, dejará el movimiento en espiral y haciendo que coincida el centro de la baliza con el centro de la imagen de la cámara. Para realizar el movimiento de

centrarse en la baliza se ha utilizado un control PD (proporcional y derivativo).

por ultimo una vez coincide la cruceta de la baliza con el centro de la imagen, comienza a descender de forma constante a la vez que va centrando imagen por si el drone se desvía por algún factor externo. Cuando el área que detecta la baliza es prácticamente el área de la baliza, en ese momento el drone desciende hasta posarse en el suelo, entonces para los motores y el estado cambia a al estado final de "*Landed*".

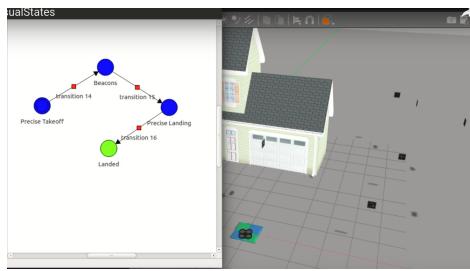


Figura 4.4: Ejemplo de los Módulos en Visual States

### 4.3. Módulo de Autolozalización

Este modulo lo desarrollo originalmente Alberto López Cerón, luego fue refactorizado por Samuel Martín para integrar una capa de comunicaciones mediante interfaces ICE, también añadió métodos para la conversión entre cuaterniones y ángulos de Euler, debido a que la aplicación original utilizaba los ángulos de Euler y la interfaz Pose3D, cuaterniones, posteriormente Manuel Zafra la actualizo para jderobot e hizo los primeros pasos de navegación de drones utilizando esta autolocalización. Por último Felipe Pérez ha cambiado los ficheros de configuración para que sea una herramienta que funcione utilizando únicamente librerías que vienen en JdeRobot por si sola sin depender de QtCreator y su compilador *qmake*, también esta creando la nueva infraestructura en ROS para que se pueda utilizar tanto en ICE como en esta, con la ventaja de que en ROS tendrá nuevos marcadores como el numero de balizas detectadas para poder hacer estudios de precision y marcas temporales para saber cuando entran las balizas en la aplicación y así caracterizar mejor el error.

En cuanto a la herramienta Slam-VisualMarkers está formada por un módulo principal *Main-Window* que interconecta el resto de módulos e implementa una interfaz gráfica de usuario. El método *ProcessImage* contenido en *CameraManager* se ocupa de procesar la imagen en 2D capturada por la cámara y buscar en ésta la presencia de marcadores además de estimar la posición 3D con respecto a éstos. Es imprescindible el fichero de *Markers.txt* que contiene toda la información necesaria de cada marcador: id, tamaño y posición. Para localizar los marcadores en la imagen se hace uso del método de detección ofrecido por la biblioteca AprilTags. Este método se aplica a una versión en escala de grises de la imagen obtenida y tiene como salida un array en el que figuran todos los marcadores encontrados. Una vez el array de marcadores detectados es generado y aplican una serie de operaciones geométricas. Estas operaciones devuelven la posición relativa de una cámara dado un sistema de referencia compuesto por la correspondencia entre los puntos 2D de la imagen y los correspondientes puntos 3D del mundo. De esta forma se obtienen los vectores de translación y rotación del marcador con respecto a la cámara. Finalmente, la matriz que contiene la posición de la cámara con respecto al mundo se obtiene multiplicando la matriz calculada, posición del marcador con respecto a la cámara, y la matriz de posición del mundo con respecto al marcador.

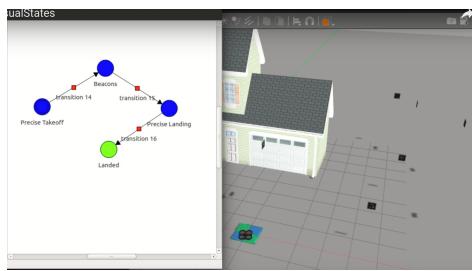


Figura 4.5: Entorno de la aplicación Slam VisualMarkers

Las posiciones estimadas para cada marcador son almacenadas en un array al que es aplicada una fusión espacial mediante un filtro por pesos. Este filtro asigna diferentes pesos a cada estimación basándose en la distancia a la cámara, de forma que

#### 4.4. INTEGRACIÓN EN EL ALGORITMO FINAL CAPÍTULO 4. DESARROLLO

los marcadores más cercanos son asignados con un peso mayor. En cuanto a los ángulos de rotación hay una operación especial, ya que no pueden ser sumados de la misma forma que las coordenadas lineales. Por último después de aplicar la fusión espacial, la aplicación original daba opción a aplicar una fusión temporal. Esta fusión puede llevarse a cabo mediante un filtro por pesos como en el caso de la fusión espacial, o mediante la aplicación de un Filtro de Kalman [9].

#### **4.4. Integración en el Algoritmo Final**

A

# **Capítulo 5**

## **Experimentos**

**5.1. A-Calidad del Controlador**

**5.2. B-Calidad de la Autolocalización**

**5.3. C-Calidad del Algoritmo completo**

**5.4. Visión Artificial y Autolocalización**

# **Capítulo 6**

## **Conclusiones**

### **6.1. Conclusiones**

### **6.2. Trabajos futuros**

# Capítulo 7

## Bibliografía

# Bibliografía

- [1] Real Decreto 1036/2017, de 15 de diciembre, por el que se regula la utilización civil de las aeronaves pilotadas por control remoto, y se modifican el Real Decreto 552/2014 y el Real Decreto 57/2002. (BOE núm. 316, 29 de diciembre de 2017).
- [2] ALBERTO MARTÍN FLORIDO. Navegación visual en un cuadricóptero para el seguimiento de objetos. *Proyecto Fin de Carrera*, URJC, 2014.
- [3] DANIEL YAGÜE SÁNCHEZ. Cuadricóptero AR.Drone en Gazebo y JdeRobot. *Proyecto Fin de Carrera*, URJC, 2014.
- [4] ALBERTO LÓPEZ-CERÓN PINILLA. Autolocalización visual robusta basada en marcadores. *Proyecto Fin de Carrera*, URJC, 2015.
- [5] ARTURO VÉLEZ. Seguimiento de un objeto con textura desde un drone con cámara. *Trabajo de Fin de Máster*, URJC, 2015.
- [6] MANUEL ZAFRA VILLAR. Seguimiento de rutas 3D por un drone con autolocalización visual con balizas. *Proyecto Fin de Carrera*, URJC, 2016.
- [7] JORGE VELA PEÑA. Despegue, navegación y aterrizaje visuales de un drone usando JdeRobot. *Proyecto Fin de Carrera*, URJC, 2017.
- [8] JOHN WANG y EDWIN OLSON, AprilTag 2: Efficient and robust fiducial detection. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Octubre 2016.

- [9] RUDOLF E. KALMAN, A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 1960.
- [10] JOSE MANUEL G., (Abril de 2010). UAVs, clasificación, tendencias y normativa de espacio aéreo. Disponible en:  
<http://blog.sandglasspatrol.com/index.php/articulos/41-militar/758-uavs-clasificacion-tendencias-y-normativa-de-espacio-aereo>