



**Universidad  
Rey Juan Carlos**

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE  
TELECOMUNICACIÓN

GRADO EN INGENIERÍA AEREOESPACIAL EN AERONAVEGACIÓN

**TRABAJO FIN DE GRADO**

**Programación de un drone para seguimiento  
autónomo de trayectorias en 3D**

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 en ventas de robots aéreos como en de software y capacidades que pueden llevar a cabo los mismos. Esto se corrobora con la nuevas aplicaciones para las que se desarrollan como el drone ambulancia que puede socorrer a una persona en caso de infarto o el taxi drone capaz de transportar personas.

En este Trabajo de Fin de Grado se aborda el seguimiento autónomo de rutas en 3D. Esta funcionalidad incluye despegue, autolocalización visual, pilotaje siguiendo una ruta y aterrizaje visual. Con ello se crea una navegación y guiado de un dron completamente autónomo, únicamente utilizando la ayuda balizas de aterrizaje y despegue, y localizadores (AprilTags). El pilotaje se ha abordado de dos maneras diferentes: primero usando como objetivo una secuencia de puntos de paso y segundo usando como objetivo trayectorias continuas en el espacio 3D. Se han integrado varios componentes en una aplicación final y se ha utilizado un autómata finito de estados para vertebrar la lógica de la aplicación. El autómata se ha diseñado empleado la herramienta VisualStates.

El componente final se ha escrito con el lenguaje de programación Python, en la versión JdeRobot 5.6 y se ha validado experimentalmente la aplicación desarrollada en el simulador Gazebo, con un drone simulado de manera realista. Se han realizado tanto experimentos unitarios como varios experimentos globales.

# 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 . . . . .	11
1.2.3. Marco legislativo español para drones . . . . .	11
1.2.4. Competiciones con robots aéreos . . . . .	14
1.3. Visión Artificial y Autolocalización . . . . .	15
1.4. Robótica aérea con JdeRobot . . . . .	17
<b>2. Objetivos</b>	<b>20</b>
2.1. Problemas a abordar . . . . .	20
2.2. Requisitos . . . . .	22
2.3. Metodología . . . . .	22
2.4. Plan de trabajo . . . . .	24
<b>3. Infraestructura</b>	<b>26</b>
3.1. Gazebo . . . . .	26
3.2. Balizas visuales AprilTags . . . . .	27

<b>ÍNDICE GENERAL</b>	<b>IV</b>
3.3. Biblioteca OpenCV . . . . .	28
3.4. Biblioteca NumPy . . . . .	29
3.5. Entorno JdeRobot . . . . .	30
3.5.1. Herramienta ColorTuner . . . . .	30
3.5.2. Plugin ArDrone2 en Gazebo . . . . .	31
3.5.3. Interfaz Pose3D . . . . .	32
3.6. Visual States . . . . .	32
3.7. Slam-Visualmakers . . . . .	34
<b>4. Navegación autónoma para seguimiento de rutas 3D</b>	<b>36</b>
4.1. Diseño . . . . .	36
4.2. Componente de Autolocalización . . . . .	38
4.3. Componente de Control basado en estados . . . . .	40
4.4. Estados de despegue y aterrizaje . . . . .	42
4.5. Estado de seguimiento de ruta . . . . .	44
<b>5. Experimentos</b>	<b>48</b>
5.1. Pruebas unitarias Autolocalización . . . . .	49
5.2. Pruebas unitarias Pilotaje . . . . .	52
5.3. Pruebas integrales del sistema . . . . .	56
<b>6. Conclusiones</b>	<b>60</b>
6.1. Conclusiones . . . . .	60
6.2. Trabajos futuros . . . . .	62
<b>Bibliografía</b>	<b>64</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.	Robot Atlas de aspecto humanoide. . . . .	6
1.6.	AAI RQ-2 Pioneer - Uno de los primeros Robots Aéreos. . . . .	8
1.7.	Drone ala fija Vs Drone ala móvil . . . . .	9
1.8.	Movimiento del Drone según sus rotores. . . . .	11
1.9.	Dron contra incendios. . . . .	14
1.10.	Visión artificial de un automóvil. . . . .	15
1.11.	TFM Alberto Martín. . . . .	17
1.12.	TFG Manuel Zafra. . . . .	18
1.13.	TFG Jorge Vela. . . . .	19
2.1.	Representación del desarrollo en espiral. . . . .	23
3.1.	Entorno de Simulación Gazebo. . . . .	27
3.2.	Ejemplos de AprilTags. . . . .	28
3.3.	Ejemplo de algoritmo con OpenCV. . . . .	29
3.4.	Herramienta ColorTuner. . . . .	31
3.5.	Ejemplo de Pose3DData . . . . .	32
3.6.	Herramienta Visual States sobre Gazebo . . . . .	33

3.7. GUI de Slam-Visualmarkers. . . . .	34
3.8. Entradas y Salidas del nodo. . . . .	35
4.1. Esquema representativo de la aplicación. . . . .	37
4.2. Entorno de la aplicación Slam VisualMarkers . . . . .	40
4.3. Esquema de los componentes de Visual States. . . . .	41
4.4. Baliza utilizada en Gazebo. . . . .	42
4.5. Ejemplo de los estados en Visual States . . . . .	43
5.1. Mundo Gazebo ArDones1. . . . .	49
5.2. Cálculo de errores de distancia y de ángulos . . . . .	50
5.3. Cálculo de posición de Slam-VisualMarkers . . . . .	51
5.4. Error asociado al pilotaje. . . . .	53
5.5. Ruta sencilla en el pilotaje por trayectorias. . . . .	54
5.6. Comparación del error entre ambos pilotos en ruta sencilla. . . . .	54
5.7. Ruta compleja en el pilotaje por trayectorias. . . . .	55
5.8. Comparación del error entre ambos pilotos en ruta compleja. . . . .	55
5.9. Prueba integral de ruta por puntos. . . . .	57
5.10. Prueba integral de ruta por trayectoria. . . . .	57
5.11. Error de distancia a la ruta deseada en la prueba integral del sistema . .	58

# Capítulo 1

## Introducción

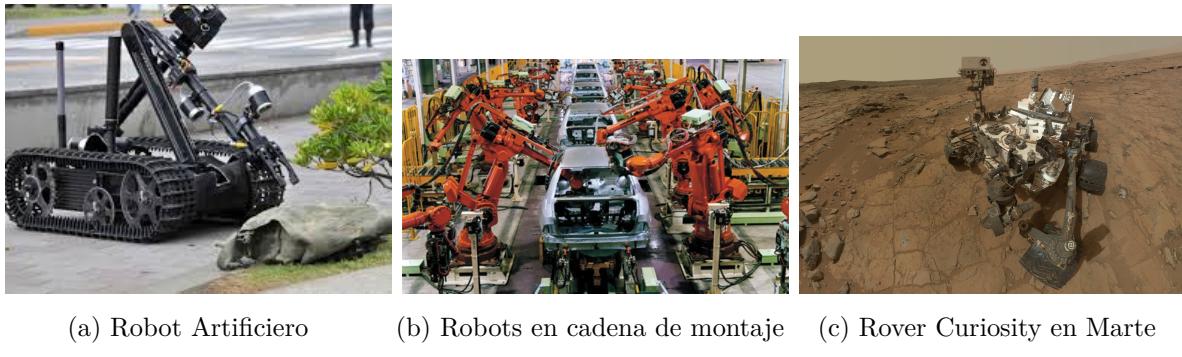
En este primer capítulo, antes de adentrarnos en la parte mas técnica, se va a introducir al lector de forma breve en qué es la robótica y más concretamente en los robots aéreos, para así conocer el estado actual de este campo y cómo ha evolucionado en los últimos años este sector, creando una gran rama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

### 1.1. Robótica actual

La Robótica es una rama multidisciplinar 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. 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 por Isaac Asimov, para el correcto funcionamiento de los robots 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 estas 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 dista mucho de la realidad y, aunque sí que se exigen unos mínimos a la hora de crear robots, cualquier parecido con la robótica de ciencia ficción en las películas 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 muchas clasificaciones posibles 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 es capaz de realizar los movimientos ordenados.
- **4<sup>a</sup> Generación:** Robots inteligentes. Similares a la generación anterior pero incluyen una gran mejora, están equipados con sensores que mediante la comunicación con la computadora de control permite una toma inteligente de decisiones y un control de procesos en tiempo real.

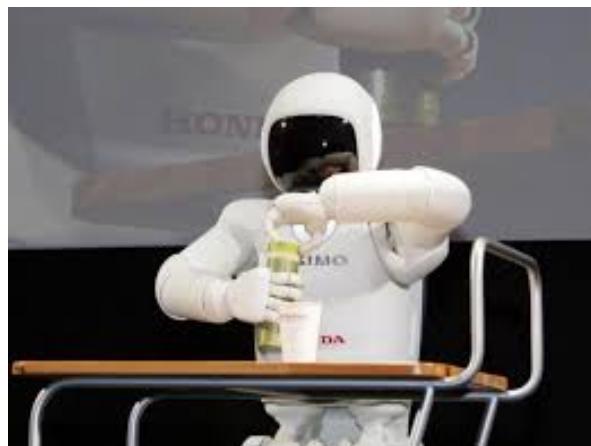


Figura 1.2: Robot ASIMO realizando acciones cotidianas.

### 1.1.2. Aplicaciones

La robótica está en constante desarrollo, esto ligado al incremento de los procesadores, así como en dispositivos hardware y desarrollo de software ha permitido que 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 monitorice por sí 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 PEPPEP en un aula como complemento educador.

- **Industria:** Se utilizan 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; también 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, 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 que son capaces de guiarse por la fábrica autónomamente. Fuera de éstas 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.
- **Medicina:** Se han desarrollado dispositivos que permiten realizar desde trabajos quirúrgicos guiados por imágenes hasta cirugía mínimamente invasiva realizada mecánicamente por un robot. 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 para sustitución parcial de alguna parte dañada del cuerpo, o incluso exoesqueletos. Otro ejemplo estaría en la robótica terapéutica, utilizada como medio de rehabilitación fisiológica y de estimulación cognitiva en tratamientos para enfermedades como el Alzheimer.



Figura 1.4: Robot DaVinci utilizado en hospitales.

- **Militar:** 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 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. 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. Por último mencionaremos el robot que está creando gran expectación por sus habilidades locomotoras y aspecto humanoide, es el llamado Atlas creado por Boston Dynamics, capaz de correr y saltar en cualquier tipo de entorno e incluso de dar volteretas.



Figura 1.5: Robot Atlas de aspecto humanoide.

- **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, también 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 últimos 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 desarrollados por este sector. Empezaron a crearse entre los años 1914 y 1918, durante la I Guerra Mundial, 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 empezó a utilizar también con fines más 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 proteger la misma de la guerra electrónica y la criptografía ya que las comunicaciones son mucho más seguras y difíciles de detectar.

Si bien el uso principal en el siglo XX era en la industria militar, en el siglo XXI visto el gran potencial que tenían, los nuevos usos que se les estaban dando y los avances en la industria aeronáutica e informática hicieron que se convirtiese en una herramienta útil para la sociedad civil. En la actualidad los UAV son útiles en diferentes industrias con diferentes objetivos. Una posible clasificación de los usos de robots aéreos sería:

- **Blanco:** Servían como simulación de aviones y ataque enemigos para 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 1.6: 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 exploran, prueban y mejoran los sistemas que avanzan cómo serán los drones del futuro.
- **Comercial y Civil:** esta utilización se encuentra en enorme crecimiento tanto de innovaciones como de ventas ya que son diseñados para propósitos civiles como:

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 se encuentran:

- **Motor:** En cuanto a la disposición de los motores 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, donde los motores giran en horizontal, 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 el vuelo del drone.



Figura 1.7: 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 muy ligero y con ello incrementar la carga útil (*pay-load*) para mejorar la funcionalidad de los mismos.

- **Batería:** La parte más crítica de los UAV actualmente 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 grande es el drone más potencia es necesaria y por tanto mayor batería se necesita, por lo que hay que encontrar el equilibrio 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 simultánea 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 de 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 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 dirigir 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 1.8 se pueden observar 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 éstos 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.

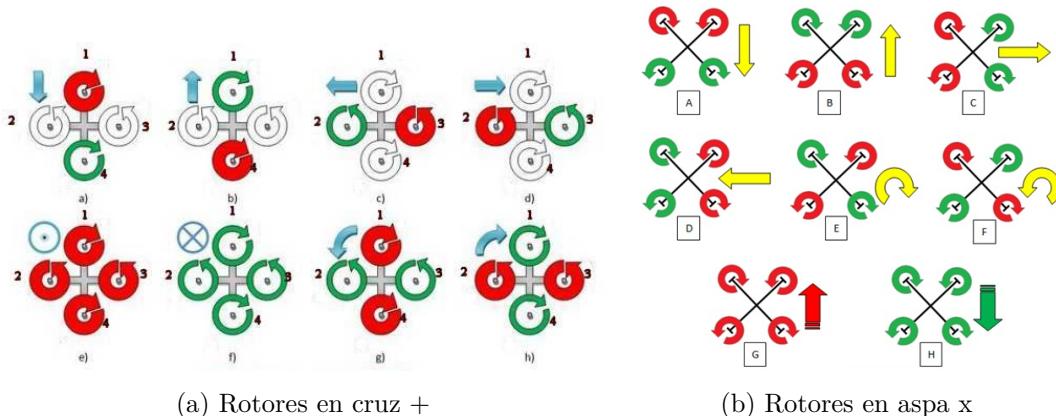


Figura 1.8: Movimiento del Drone según sus rotores.

### 1.2.3. Marco legislativo español para drones

En el marco legislativo actual en España sobre los UAV se acordó el 15 de Diciembre de 2017 en el Consejo de Ministros y se publicó en el BOE [1]. 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 operar cualquiera de ellos es imprescindible disponer de un carnet de piloto de drones válido en toda 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 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 antigua 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 sólamente 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.

#### 1.2.4. Competiciones con robots aéreos

En cuanto a los inventos de drones o de usos de éstos ilustraremos tres certámenes 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 se 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 1.9: 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, drone que extingue incendios forestales, el cual tiene la capacidad de adaptarse a las condiciones de un fuego para apagarlo<sup>1</sup>.

### 1.3. Visión Artificial y Autolocalización

El drone manejado en este TFG incorpora un sistema de autolocalización visual, 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 estima que el 70 % de las tareas del celebro se emplean en analizar estos datos visuales. La Visión Artificial o Visión por Computador busca extraer la información visual desde los datos visuales mediante procedimientos automáticos. Para conseguir que esto sea viable y no se tarde días se han desarrollado técnicas de procesamiento de imágenes que han avanzado a niveles de que el desfase entre la visión artificial y la realidad sea prácticamente nulo.



Figura 1.10: Visión artificial de un automóvil.

---

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

La visión artificial se basa en la naturaleza de la luz, que es la parte de la radiación electromagnética que puede ser percibida por el ojo humano, está formada por fotones cuyas propiedades en relación con la dualidad de onda explican las características de su comportamiento. La Visión Artificial se basa en la formación de imágenes y, como hemos dicho antes, en el sistema de procesamiento de éstas. Típicamente el primer apartado de un sistema de visión artificial 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 privatización del GNSS y la utilización de este sistema para la sociedad, ha permitido que hoy podamos utilizar la localización en cualquier dispositivo con una simple antena, a 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.), además existen 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 todos los nuevos dispositivos utilizan un localizador GNSS, el problema ocurre 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 un sistema de autolocalización basado en visión y en balizas, el 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 ecosistema del 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 seguimiento de objetos*. En él abordaba la navegación visual autónoma de un cuadricóptero implementando algoritmos de navegación visual para el seguimiento de objetos de manera automática tanto utilizando la cámara frontal como utilizando la cámara inferior.

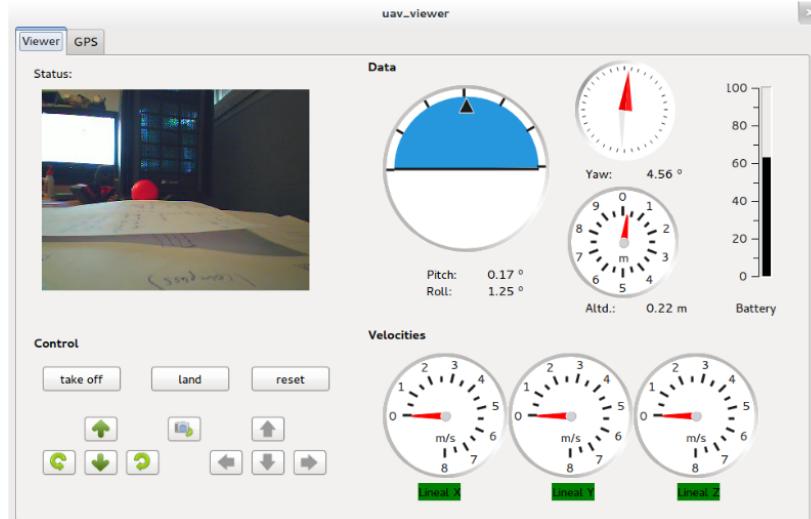


Figura 1.11: TFM Alberto Martín.

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 dentro del simulador Gazebo en el entorno JdeRobot y crear varias aplicaciones de navegación para validarla.

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

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

Alberto López-Cerón<sup>4</sup> [4] *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 estimar la posición de la cámara. 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 drone con cámara*. Aquí trabajó en un seguimiento visual, esta vez sin filtro de color, 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*. Diseñó un sistema de vuelo autónomo para drones en espacios interiores, basándose en la autolocalización mediante la visión artificial como se puede observar en la figura 1.12.

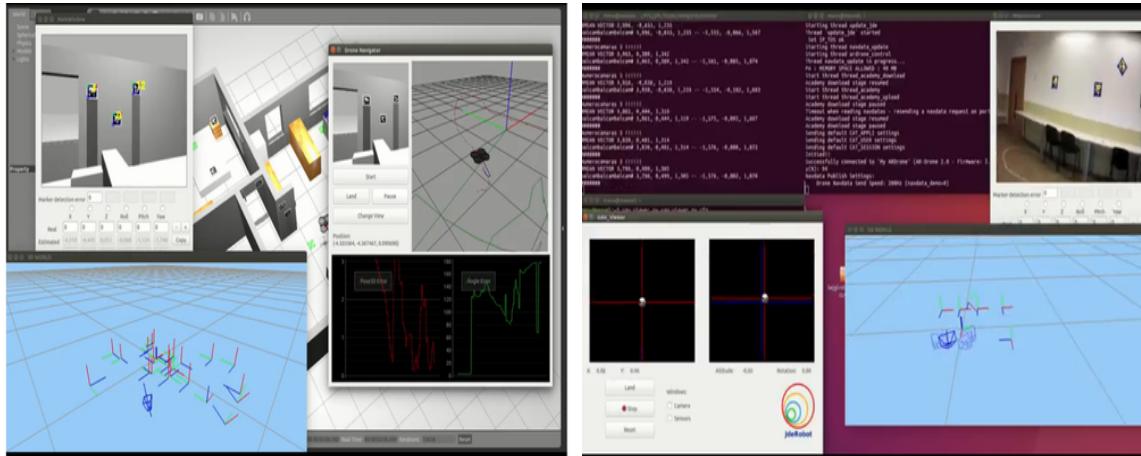


Figura 1.12: TFG Manuel Zafra.

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

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

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

Jorge Vela <sup>7</sup> [7] Aborda el *despegue, navegación y aterrizaje visuales de un drone usando JdeRobot*. Se centró en la localización y aterrizaje controlado por visión de un drone mediante la detección visual de balizas visible en la figura 1.13.

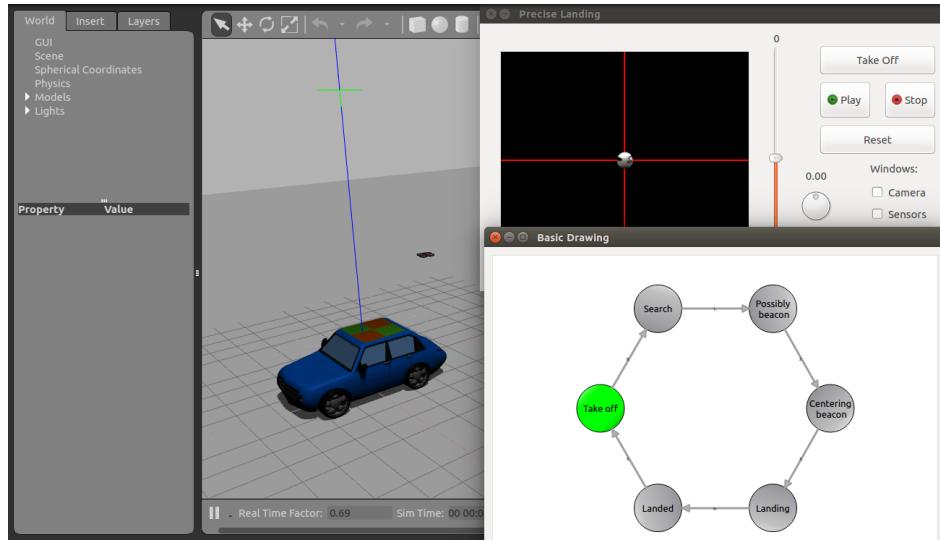


Figura 1.13: TFG Jorge Vela.

Una vez expuesta la breve introducción, el resto de la memoria se ha organizado en otros cinco capítulos. 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 el diseño y 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>7</sup><http://jderobot.org/Jvela-tfg>

# Capítulo 2

## Objetivos

Una vez introducidas las motivaciones que han llevado a hacer este Trabajo Fin de Grado y expuesto el contexto, en este capítulo se van a exponer los diferentes problemas concretos que se abordarán, los requisitos para las soluciones desarrolladas y por último 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 que despegue de forma controlada, siga una ruta previamente establecida y aterrice 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.

Este objetivo se ha desglosado en varios subobjetivos para abordarlo por partes:

1. **Adaptación e integración del módulo de autolocalización visual:** Este módulo proporciona la posición absoluta del drone en el mundo basándose en la detección de balizas visuales y cálculos geométricos. Existía una versión previa

debido al TFM de Alberto y al PFC de Manuel pero nadie la había integrado dentro de un automata de estados finito hemos ajustado el componente y lo hemos introducido por pasos para finalmente conseguir su correcto funcionamiento.

2. **Adaptación e integración del módulo de aterrizaje visual:** Este módulo se encarga de controlar al drone para que se pose encima de una baliza visual arlequinada. Existía una versión previa fruto del TFG de Jorge Vela, que hay que adaptarla e integrarla, además a partir de esta se ha creado el propio sistema de despegue controlado.
3. **Diseño y desarrollo de un módulo pilotaje del dron basado en la posición absoluta:** Se explorará tanto basado en una ruta continua de puntos consecutivos como basado en una colección de subobjetivos más distantes.
4. **Desarrollo de la inteligencia del drone materializándola en un autómata de estados finito:** 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.
5. **Validación experimental en entorno simulado:** Se creará un mundo tridimensional en el que se realizarán las pruebas pertinentes para validar la solución final desarrollada. También se realizarán pruebas unitarias, se compararán diferentes soluciones de pilotaje, se analizará la sensibilidad del sistema al ruido en el subsistema de autolocalización y se calculará el error producido en la estimación de posición por el componente.

## 2.2. Requisitos

Una vez descritos todos los objetivos, a la solución a desarrollar se le va a exigir también que cumpla con varios requisitos adicionales:

- El algoritmo funcionará en la plataforma de desarrollo JdeRobot-5.6.3.
- El sistema se ejecutará en el entorno GNU/Linux Ubuntu 16.04.
- La aplicación se ejecutará 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.
- El control de navegación mediante el pilotaje debe ser suave para no perder los marcadores y balizas, pero también 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 ha permitido trabajar de forma progresiva, es decir, empezando por las partes más sencillas y a medida que las resolvíamos, avanzar hasta las más complejas. La forma de conseguirlo fue mediante la marcación de hitos a alcanzar que se revisaban periódicamente mediante reuniones con el tutor. Se analizaban si se había conseguido llegar al resultado esperado en la etapa y así pasar al siguiente objetivo. Para este nuevo objetivo se analizaban

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 <sup>1</sup>. Esta bitácora es pública y 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 cómo solucionarlas. En ella se pueden encontrar desde fotos y videos de la práctica final como resultados y los pasos iniciales que se hicieron para adentrarse en el mundo de la programación de robots y drones.

El código fuente desarrollado a lo largo de todas estas etapas que se observan en el mediawiki, de acceso público y está disponible en <sup>2</sup>.

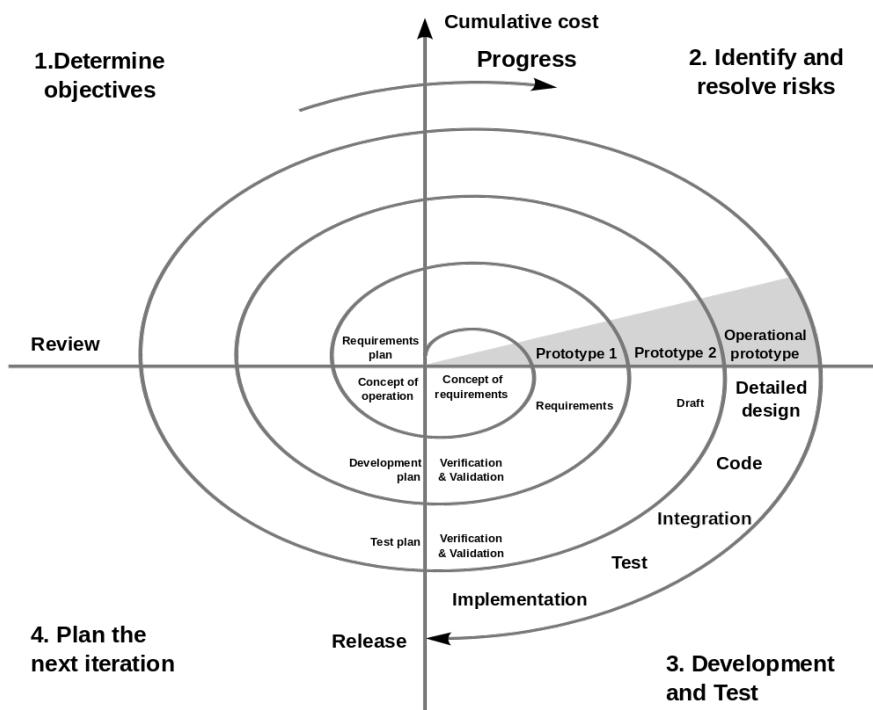


Figura 2.1: Representación del desarrollo en espiral.

<sup>1</sup><http://jderobot.org/Jsaizc-tfg>

<sup>2</sup><https://github.com/RoboticsURJC-students/2017-tfg-jesus-saiz>

## 2.4. Plan de trabajo

Para conseguir los objetivos expuestos anteriormente se ha seguido una planificación dividida en estas fases:

- **Formación en Python y C++:** Necesaria para entender el código existente de los antecedentes directos de este Trabajo Fin de Grado en los que me he basado y para crear el mío propio. Con el conocimiento y seguimiento de tutoriales y ejercicios básicos de c/c++ y una serie de guías de ayuda fue el primer paso de aprendizaje.
- **Formación con herramientas:** Introducción a JdeRobot para comprender el funcionamiento de la infraestructura y saber todas las herramientas con las que se contaba. 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 el simulador 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. Formación con la herramienta VisualStates de programación con autómatas.
- **Aprendizaje e integración del componente de autolocalización:** Necesario para conocer el funcionamiento y el uso del componente Slam-VisualMarkers, y manejar los parámetros y las balizas con las que trabaja, conseguir con él una correcta autolocalización del drone y tambien obtener los errores de la herramienta y extraerlos de los errores de pilotaje.
- **Aprendizaje e integración del control de aterrizaje:** Para adaptarlo a la aplicación dentro de un autómata finito de estados y a su vez crear un propio sistema de despegue del drone.
- **Diseño y desarrollo del algoritmo para el pilotaje:** Se han explorado dos algoritmos, por un lado seguimiento de puntos separados y por otro lado el de

seguimiento de trayectorias continuas. Ambos capaces de gobernar el movimiento del drone para seguir las rutas en 3D.

- **Integración de todos los módulos de navegación en un autómata de estados finito desarrollado con VisualStates:** 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.
- **Validación experimental:** Para comprobar y validar el correcto funcionamiento de las fases anteriores tanto unitariamente como del sistema global 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 acercándonos a su viabilidad en entornos reales.

# Capítulo 3

## Infraestructura

Una vez vistos los objetivos de este trabajo, en este capítulo se mostrarán las diferentes infraestructuras en las que nos hemos apoyado para la programación de la aplicación robótica de este TFG. También se explicara el funcionamiento de algunos componentes previos que hemos integrado.

### 3.1. 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.

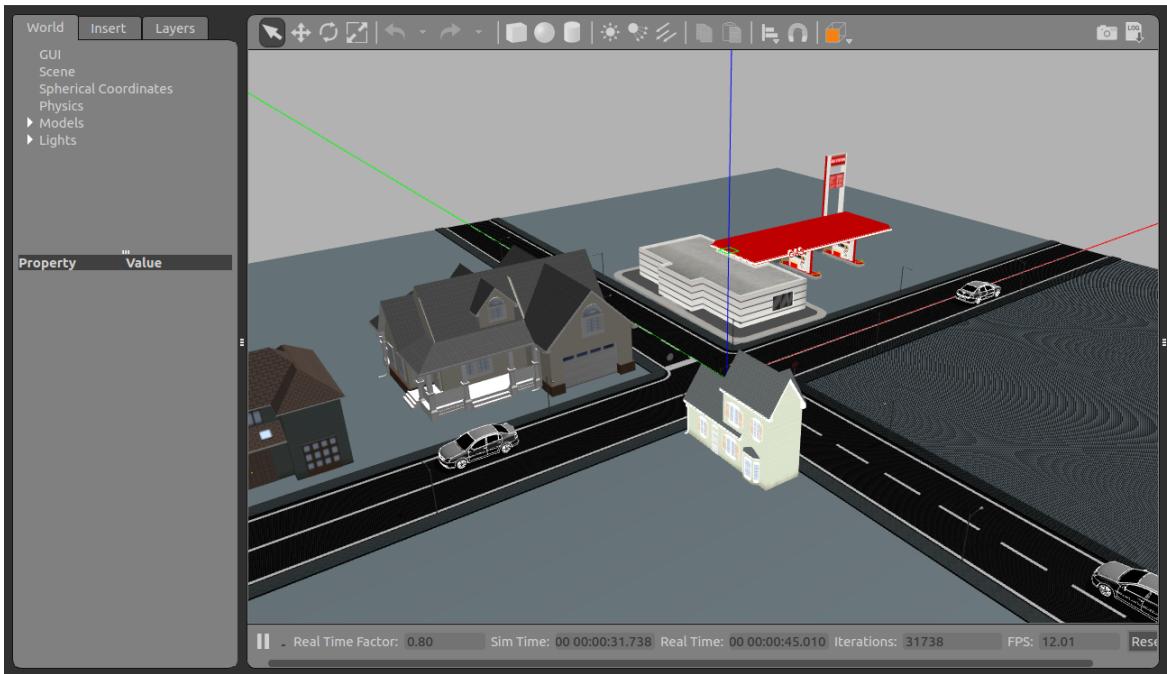


Figura 3.1: Entorno de Simulación Gazebo.

En nuestro TFG hemos trabajado con la versión Gazebo 7.9. Los mundos creados en esta aplicación se pueden lanzar sin GUI o con GUI y se definen con la extensión '.world' y escritos mediante SDF (*Simulation Description Format*).

### 3.2. Balizas visuales AprilTags

AprilTags [8] es una biblioteca para el 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 pue-

den 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.

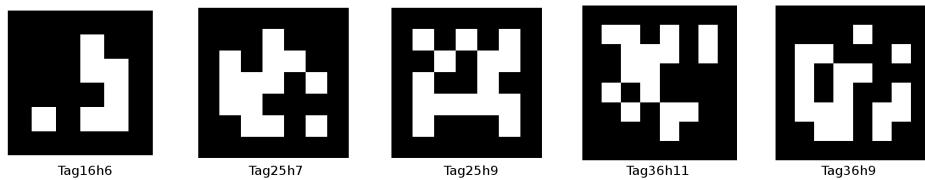


Figura 3.2: Ejemplos de AprilTags.

Esta biblioteca la hemos utilizado como ayuda al sistema de autolocalización de Slam-VisualMarkers y su el código en C++ desarrollado primero por Edwin Olson y posteriormente por Michael Kaess se puede obtener gratuitamente en su página <sup>1</sup>, nosotros hemos utilizado su versión 2.1.

### 3.3. 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 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.

---

<sup>1</sup><http://people.csail.mit.edu/kaess/apriltags/>

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

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. En nuestro caso, nos ha servido tanto para la localización de las balizas de despegue y aterrizaje, como para la clasificación de las balizas AprilTag según la distancia a la que se encuentran del drone y por tanto el peso que tienen a la hora de calcular la posición.

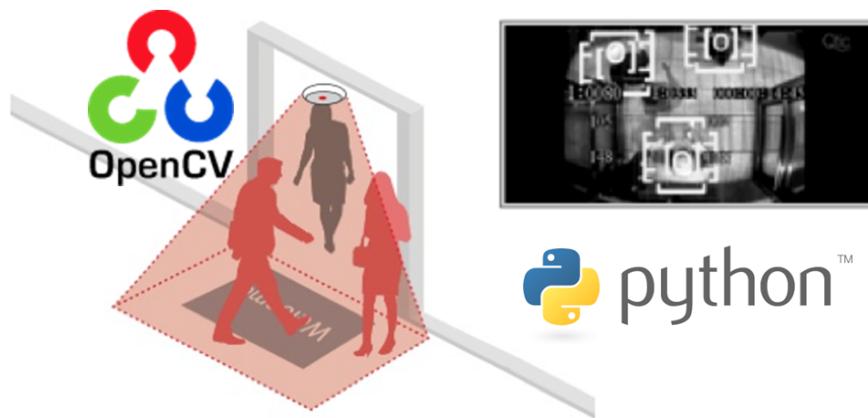


Figura 3.3: Ejemplo de algoritmo con OpenCV.

### 3.4. Biblioteca NumPy

Como hemos dicho este TFG está escrito en el lenguaje de programación Python y NumPy<sup>3</sup> 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++

---

<sup>3</sup><http://www.numpy.org/>

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.

## 3.5. Entorno JdeRobot

JdeRobot<sup>4</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.

Para nuestro trabajo hemos utilizado la versión mas reciente del repositorio, JdeRobot-5.6.3(Febrero 2018)<sup>5</sup>. Nos ha servido como herramienta para comprender los componentes previos que hemos utilizado del repositorio y la aplicación final es un conjunto de nodos desarrollados en el ecosistema JdeRobot.

### 3.5.1. 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

---

<sup>4</sup>[http://jderobot.org/Main\\_Page](http://jderobot.org/Main_Page)

<sup>5</sup><https://jderobot.org/Installation>

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. Se ha utilizado para extraer los colores de las balizas de aterrizaje y despegue y así poner su gama de colores correctamente en la aplicación final para que el drone las encuentre sin problemas.

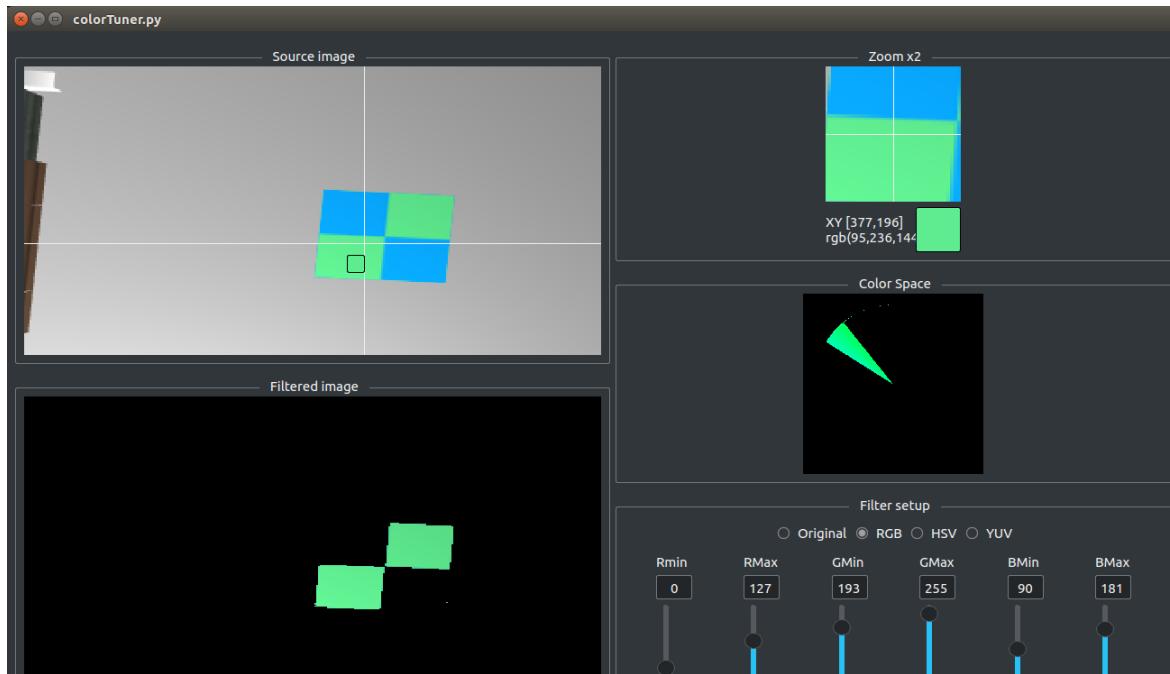


Figura 3.4: Herramienta ColorTuner.

### 3.5.2. Plugin ArDrone2 en Gazebo

Hemos empleado para el desarrollo del proyecto el ArDrone 2.0 o más 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. Este drone cuenta con sensores IMU y con dos camaras una frontal y otra ventral, con ello y gracias al simulador Gazebo es capaz de ofrecer datos como la posición del drone o las imágenes que ve el mismo. También cuenta con una

brújula, un altímetro y cuatro motores cada uno con su correspondiente hélice lo que nos permite enviar al drone una serie de comandos mediante el plugin *CMDVel* para dirigir sus movimientos y saber en todo momento en que posición se encuentra.

### 3.5.3. 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.

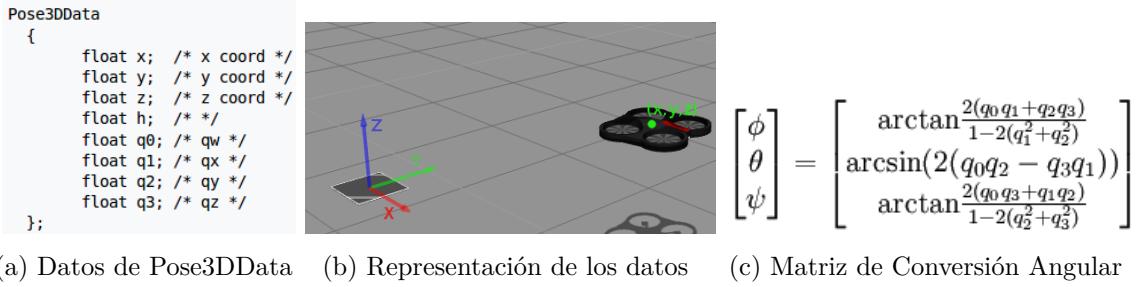


Figura 3.5: Ejemplo de Pose3DData

## 3.6. 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*) que permite el diseño gráfico de la inteligencia de un robot expresándola como estados y transiciones, además genera automáticamente un nodo JdeRobot en python que lo materializa. Representa el comportamiento del robot gráficamente en una hoja en blanco compuesto 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 una interfaz de usuario, la cual varía según estemos modificando el código o ejecutándolo. 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.

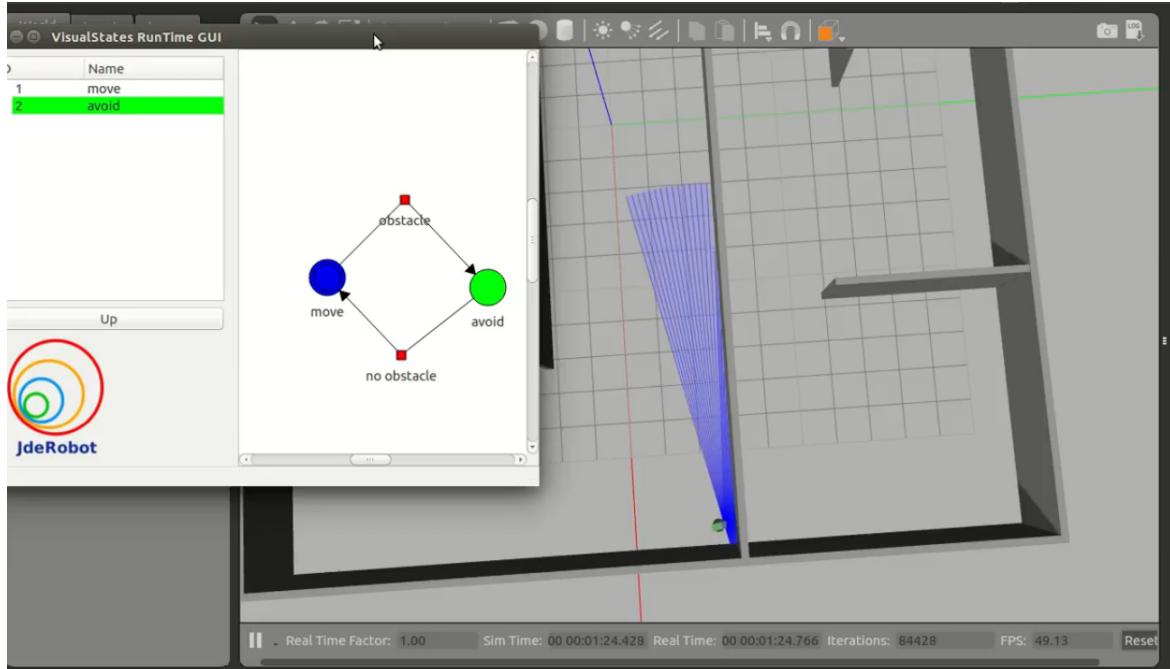


Figura 3.6: Herramienta Visual States sobre Gazebo

En nuestro TFG hemos utilizado esta herramienta para la creación de nuestro propio autómata, gracias a ello hemos podido dividir el problema final en subapartados y así solucionarlos poco a poco hasta llegar al algoritmo final. Se puede encontrar una guía de la herramienta en <sup>6</sup>.

<sup>6</sup>[https://jderobot.org/Tutorials#VisualStates\\_tool](https://jderobot.org/Tutorials#VisualStates_tool)

### 3.7. Slam-Visualmakers

Es un nodo disponible en el ecosistema de JdeRobot que mediante una serie de algoritmos capaces de localizar al sensor de forma simultánea a partir de balizas visuales proporciona la capacidad de autolocalización. Es una aplicación desarrollada por Felipe Pérez Molina <sup>7</sup> en su Trabajo Fin de Máster que ha ido creando paralelamente a este TFG. Su trabajo esta desarrollado en la plataforma JdeRobot y está programada en C++.

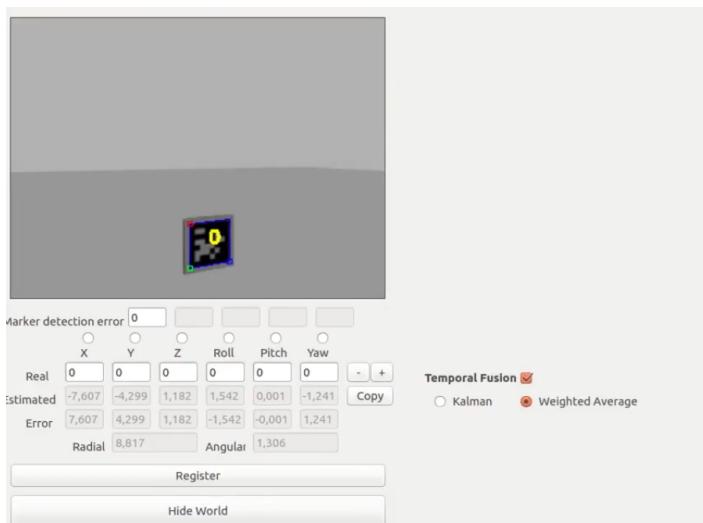


Figura 3.7: GUI de Slam-Visualmarkers.

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

<sup>7</sup><http://jderobot.org/F1perez-tfm>

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.

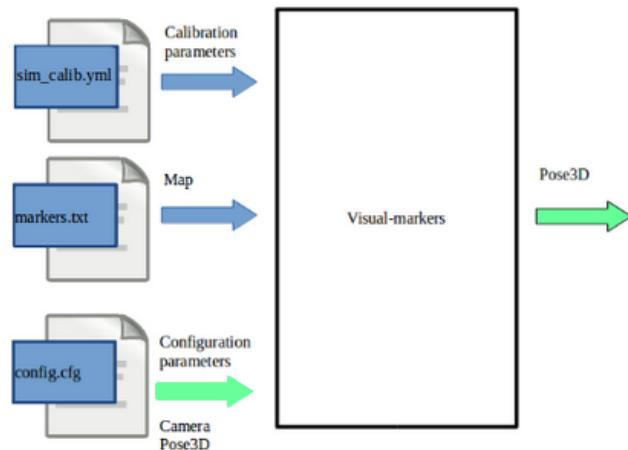


Figura 3.8: Entradas y Salidas del nodo.

# **Capítulo 4**

## **Navegación autónoma en seguimiento de rutas 3D**

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 explicará en detalle el diseño y el funcionamiento de cada uno de los componentes utilizados. Por último, se detallará cómo ha sido el proceso de integración y cual es la estructura del producto 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

## CAPÍTULO 4. NAVEGACIÓN AUTÓNOMA EN SEGUIMIENTO DE RUTAS 3D

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 según 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.

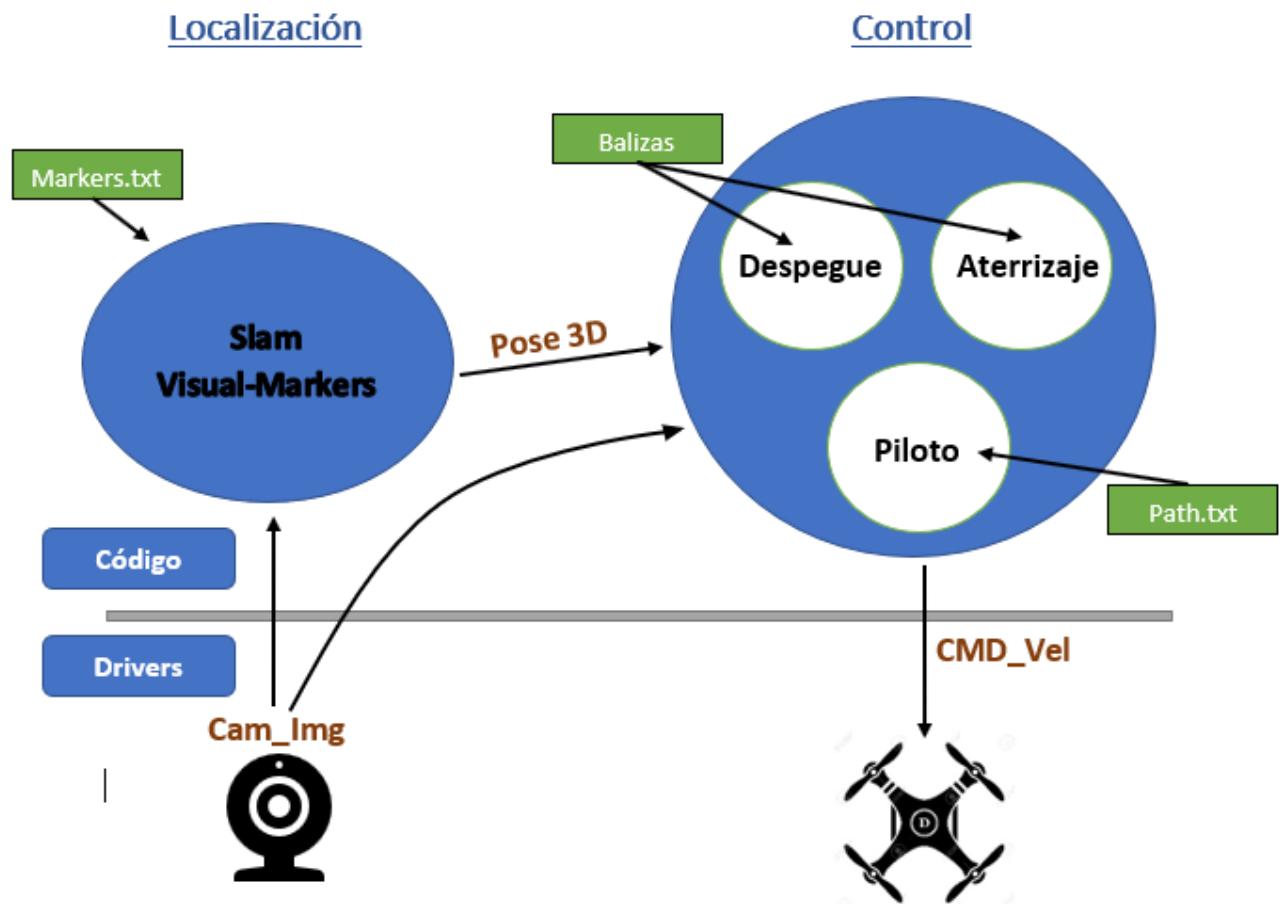


Figura 4.1: Esquema representativo de la aplicación.

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 número 0 como indicador de ello, en cambio si encuentra alguna envía cuántas 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 fusión 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. Componente de Autolocalización

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 de posición.

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, se 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.

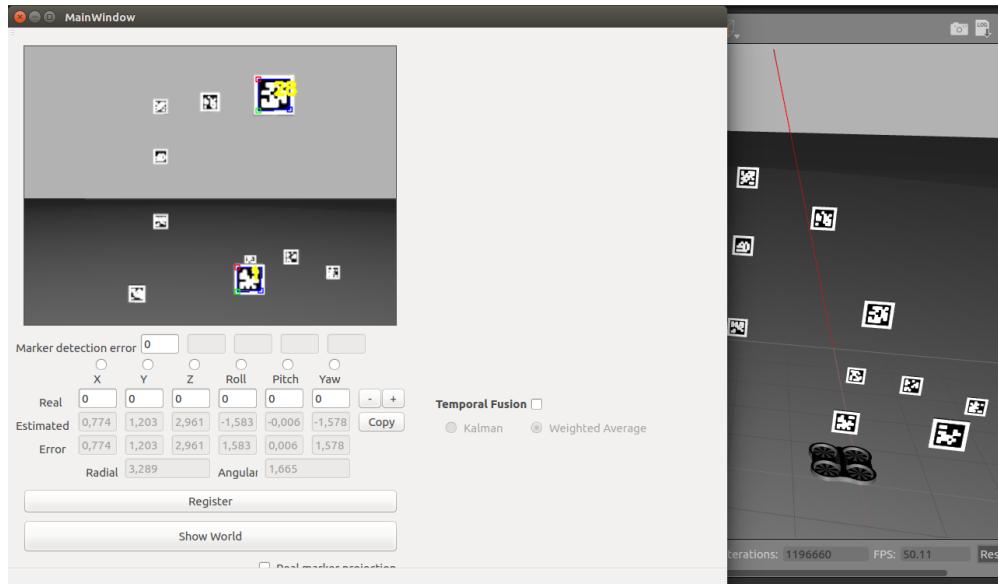


Figura 4.2: Entorno de la aplicación Slam VisualMarkers

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 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].

### 4.3. Componente de Control basado en estados

Todo el algoritmo de la aplicación se ha basado en un componente de control basado en estados, este componente se ha realizado con la herramienta de Visual States, la cual nos ha servido tanto para la creación del código como para la integración del sistema en un solo programa.

Visual States tiene la capacidad de crear estados, donde dentro se encuentra el código que se ejecuta, los cuales recorre mediante transiciones que pueden ser temporales o condicionales. Esto te permite centrarte en la escritura del algoritmo y de la conexión entre estados se encarga la aplicación. Además tiene otros módulos como las constantes y las funciones que una vez creadas se pueden utilizar en todos los estados y transiciones pudiendo así conectar y llevar un seguimiento de lo que ocurre en cada estado. Por último tenemos el apartado de las librerías, en el cual se introducen aquellas de las que depende tu programa, y el apartado de configuración, donde irán los diferentes enlaces a los interfaces de comunicación que hayamos utilizado.

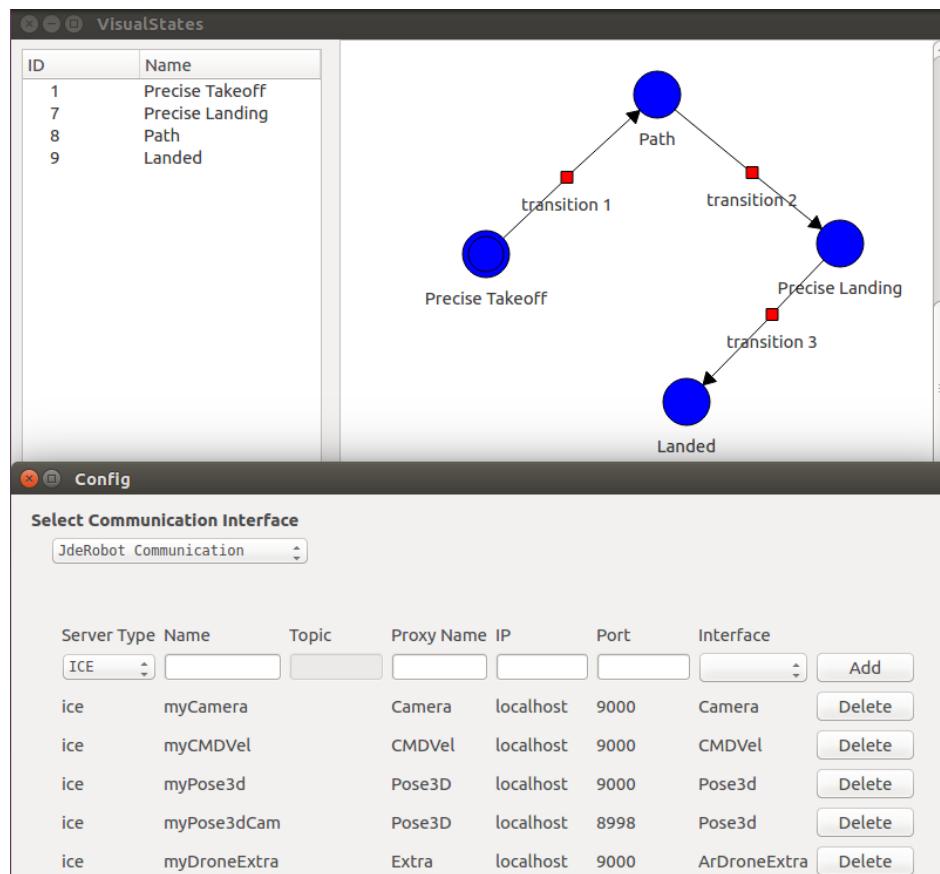


Figura 4.3: Esquema de los componentes de Visual States.

#### 4.4. Estados de despegue y aterrizaje

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 4.4. 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.

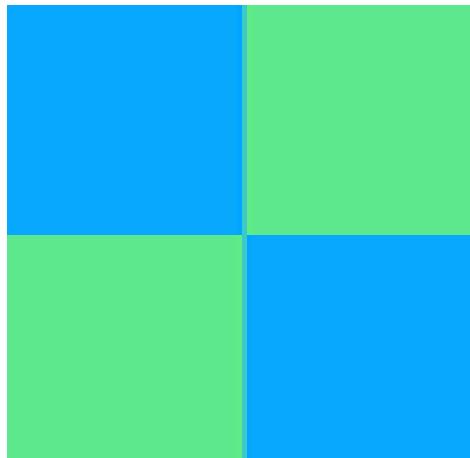


Figura 4.4: Baliza utilizada en Gazebo.

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 último 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 dron se desvía por algún factor externo. Cuando el área que detecta la baliza es prácticamente el área de la baliza, en ese momento el dron desciende hasta posarse en el suelo, entonces para los motores y el estado cambia a al estado final de "*Landed*".

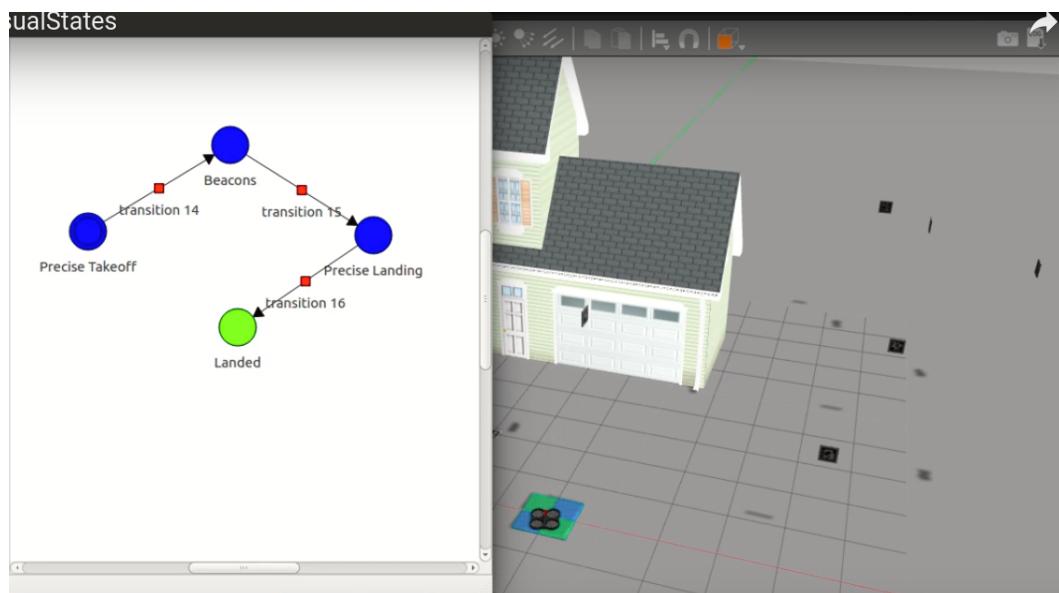


Figura 4.5: Ejemplo de los estados en Visual States

## 4.5. Estado de seguimiento de ruta

Como se puede observar en la imagen 4.3 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 **Piloto por puntos de paso** 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 **Piloto por trayectoria**. 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 **Piloto por puntos de paso**. 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 variara prácticamente ninguno de sus ángulos de Euler, tan solo se moverá sobre los ejes x,y,z.

Como el movimiento sera 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 compruebe que las velocidades máximas no exceden las permisibles por Slam VisualMarkers 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 el, 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 **Piloto por trayectoria** lo primero fue crear las rutas que posteriormente seguiría el drone, como estas debían ser largas con giros y con puntos muy consecutivos se creo en python la función que nos permitiese crear estas según la separación entre puntos de ruta que considerásemos 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 **Piloto por puntos de paso** 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 mas 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 versión 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 cálculo de las velocidades lineales se seguirán los mismos pasos que en el **Piloto por puntos de paso** pero variaremos los coeficientes de corrección para que las velocidades lineales sean parejas a la nueva velocidad angular. Esta velocidad angular se calculará 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 una conversión para conocer el ángulo 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.

# Capítulo 5

## Experimentos

En este capítulo se describen los experimentos realizados para validar el prototipo final, para comprobar el correcto funcionamiento del algoritmo completo y llegar a una solución final tras depurar mientras se desarrollaba. Para ello se ha dividido el código en tres partes a analizar: la primera parte, se ha evaluado la calidad del controlador, es decir, el error que comete el piloto al realizar una ruta predefinida; la segunda parte, se ha comprobado la calidad de autolocalización, es decir, se ha realizado un ruta teleoperada para ver los errores de posición de la aplicación y por último, se ha validado el error final de la aplicación con todos los componentes funcionando a la vez.

Dada la complejidad de los experimentos, estos se han llevado a cabo dentro del entorno de simulación Gazebo. El plugin para la simulación del comportamiento del drone, así como el modelo necesario para la renderización del vehículo y sus sensores están incluidos en JdeRobot. Pero para evaluar correctamente los diferentes experimentos se han creado una serie de mundos en 3D, los cuales cuentan con espacios abiertos, espacios cerrados, balizas cuarteadas y AprilTags con sus correspondientes texturas. Los mundos creados se han integrado en los repositorios oficiales de JdeRobot bajo los nombres de *ArDrone1*, *ArDrone2* y *ArDrone3*.

Tanto para el cálculo de trayectorias como para el calculo de errores de distancia y de tiempos de ruta se ha hecho uso del programa Matlab el cual nos

ha facilitado todos los procesos de cálculo. En la Figura 5.1 se puede observar el mundo ArDrones1, que fue el primero que creamos y el mas representativo ya que se puede observar una posible aplicación de reconocimiento de un casa mediante el dron completamente automatizado. Cunetas con las balizas cuarteadas de despegue y aterrizaje, con 23 AprilTags para la autolocalización y con una casa a la cual tiene que darle una vuelta.

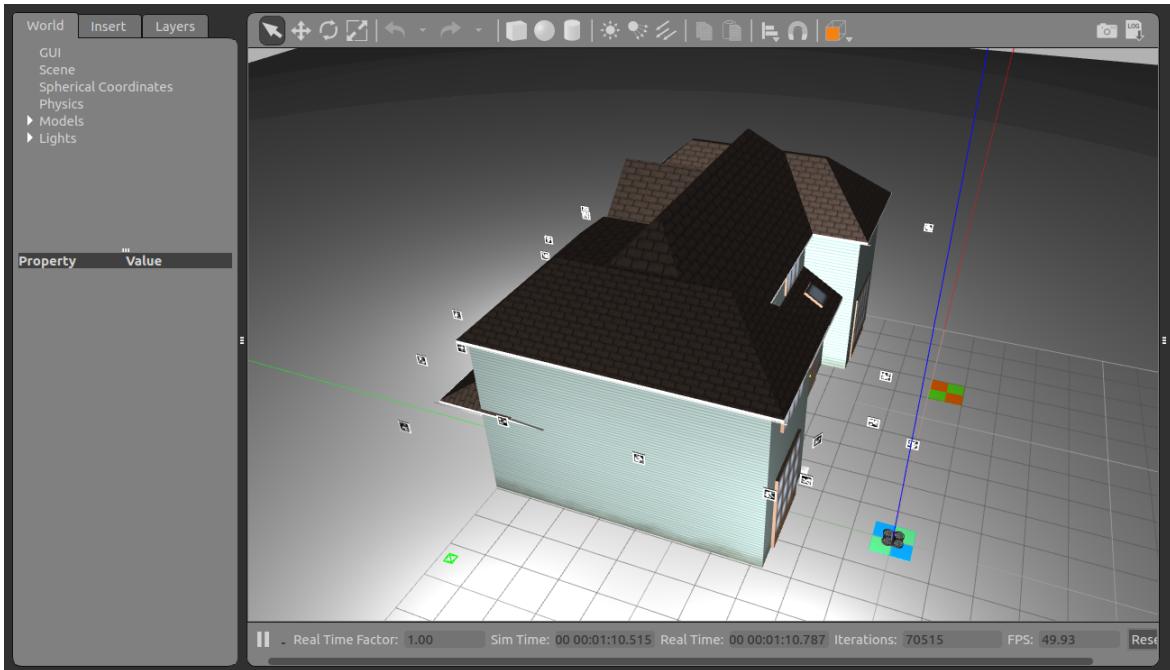


Figura 5.1: Mundo Gazebo ArDones1.

## 5.1. Pruebas unitarias Autolocalización

Para evaluar el comportamiento de la nueva aplicación de JdeRobot, Slam-Visualmarkers, que se encarga de la autolocalización del dron, se aislo completamente de la parte de pilotaje. Para ello lo que se hizo fue teledirigir el dron dentro de un mundo de balizas el cual permitiese a la aplicación funcionar correctamente viendo al menos una AprilTag en toda la ruta. Una vez terminada la ruta se comparo la posición

real entregada por el simulador  $P_0$  con la posición estimada por el componente  $P_A$ . Dado que ambas estrechan tanto posición (x y z) como dirección (Pitch Roll Yaw) se realizó un cálculo de error de posición mediante la distancia euclídea:

$$E_p = \sqrt{(P_{Ax} - P_{0x})^2 + (P_{Ay} - P_{0y})^2 + (P_{Az} - P_{0z})^2}$$

y lo que hemos definido como el error angular:

$$E_a = \sqrt{(P_{AP} - P_{0P})^2 + (P_{AR} - P_{0R})^2 + (P_{AY} - P_{0Y})^2}$$

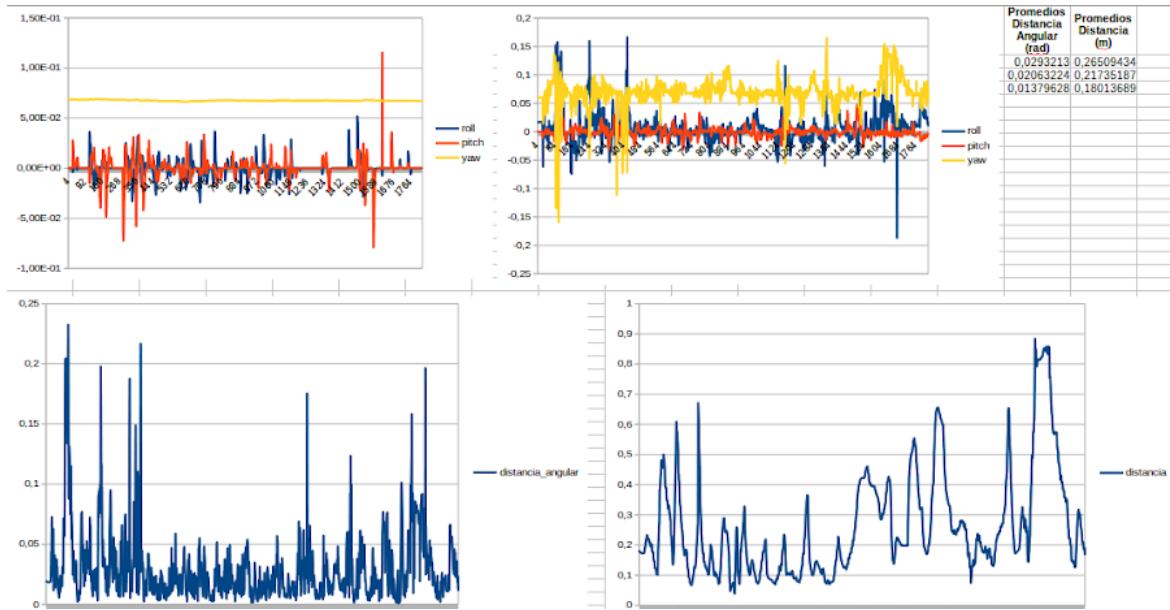


Figura 5.2: Cálculo de errores de distancia y de ángulos

Con la aplicación de las fórmulas anteriores se pueden obtener las gráficas de la figura 5.2. En las gráficas superiores se ve la diferencia entre los tres ángulos pitch, roll y yaw del drone real (izquierda) con los calculados por la aplicación de Slam-VisualMarkers (derecha). Comparando visualmente las gráficas parece que el error sea elevado pero una vez analizados numéricamente los datos y calculando los errores con la fórmula anterior y representándolos en la gráfica inferior izquierda se ve como el error angular medio es de 0.0293rad lo que equivale a 1.679 grados lo que es un error

muy pequeño, el error angular máximo es de 0.2329rad que son 13.344 grados. Por último, la gráfica inferior derecha nos muestra los errores de distancia euclídea entre la ruta realizada realmente con la posición que se calculaba con Slam-VisualMarkers, esta nos da que el error medio de distancia euclídea es de 0.2651m y el error máximo de distancia euclídea es de 0.8856m, con el cálculo de estos números y teniendo en cuenta que los errores máximos se producen cuando el drone solo ve una baliza y esta está alejada son cifras bastante razonables.

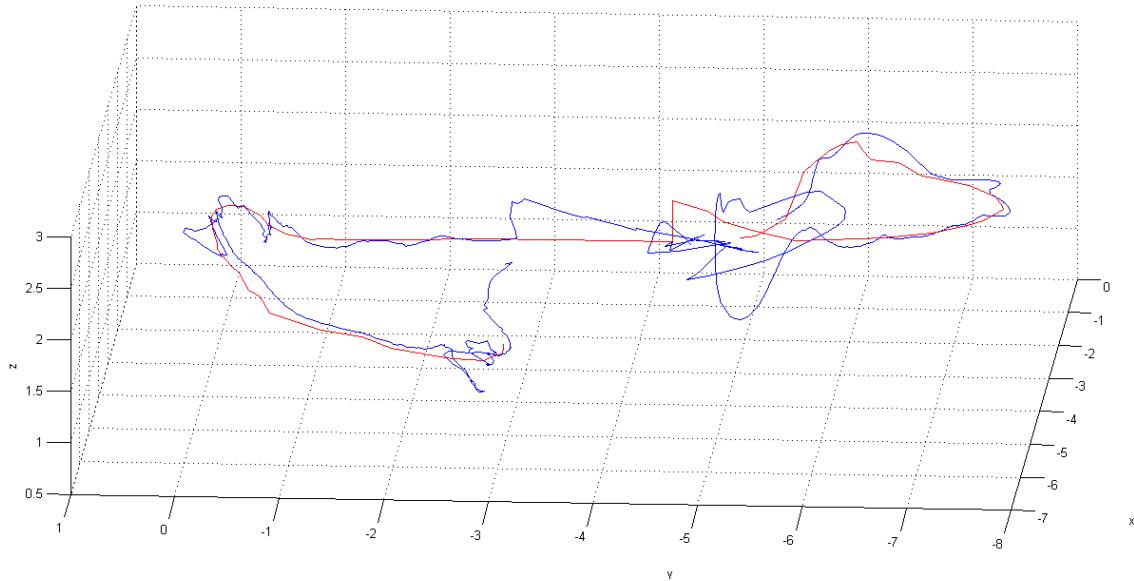


Figura 5.3: Cálculo de posición de Slam-VisualMarkers

Una vez realizados los cálculos del error se comprobó como cuando el componente pierde de vista las AprilTags los errores de posicionamiento aumentan considerablemente como se puede observar en la imagen 5.3. Para evitar esto se hablo con el desarrollador de la aplicación y se crearon una serie de marcadores dentro del nuevo componente en ROS los cuales nos indicaban el numero de balizas que se estaban viendo, con esto se pudo crear una nueva función que permite que cuando el drone no este viendo ninguna baliza no siga la ruta si no que gire sobre si mismo realizando círculos, cada vez mayores, sobre el último punto donde encontró la baliza para así

poder volver a encontrarla y posicionarse correctamente. Esto nos ayudo a que el drone no se perdiese de la ruta y siguiese moviéndose cuando no encontraba ninguna baliza, ya que ahora si da tres vueltas y seguía sin encontrar ninguna baliza, el drone aterriza. Con esta nueva función mejoramos la integridad de la aplicación y nos aseguramos de que el cuadricóptero no se aleje excesivamente de la ruta definida, preparándolo para los experimentos reales.

## 5.2. Pruebas unitarias Pilotaje

Una vez comprendido el funcionamiento del drone con las prácticas iniciales de *JdeRobot Academy* empezamos a crear el piloto que permitiera al drone seguir rutas. El primer piloto se encargaba de seguir únicamente puntos de paso, es decir tan solo variaría su velocidad lineal. Los primeros experimentos mostraban como el piloto se acercaba correctamente al punto pero a la hora de la aproximación final no lograba alcanzarlo por lo que se tuvieron que realizar varias correcciones tanto de velocidades máximas en los ejes x e y, como de variaciones de velocidad según la cercanía al siguiente punto. Una vez conseguido todo esto se volvieron a ajustar los parámetros de velocidades máximas, esta vez en los tres ejes para conseguir que el error en una ruta por puntos nunca superase los 10cm. Para ello se tuvo que sacrificar la velocidad máxima cuando el drone se acercaba al siguiente punto pero aun así en la velocidad media del drone continuaba siendo el 0.82 de la máxima. En la siguiente imagen se puede ver una ruta por puntos compleja (en verde) y como ajustando la velocidad cercana a los puntos la ruta final del drone es mas exacta (rojo) que permitiéndole una velocidad máxima mayor (azul). Por tanto podremos elegir si queremos que realice la ruta con mayor rapidez o con una mayor exactitud.

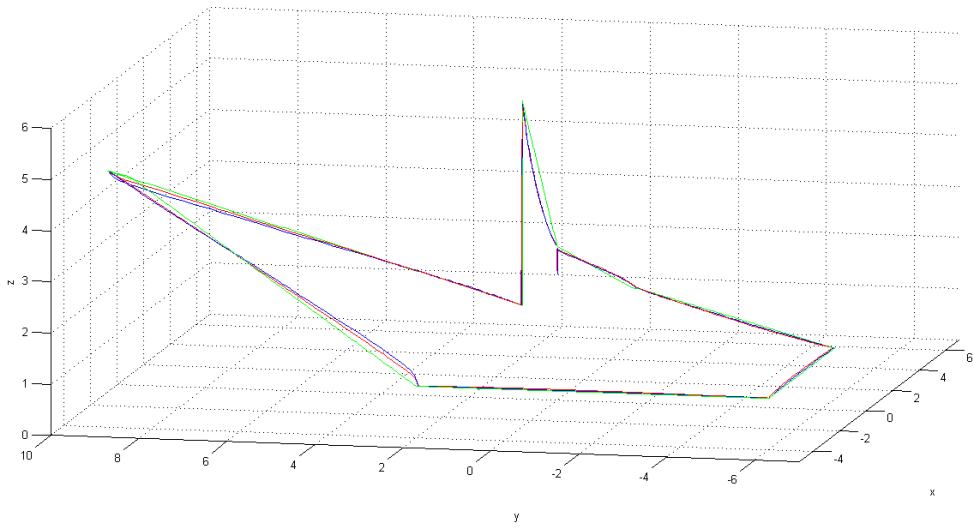


Figura 5.4: Error asociado al pilotaje.

Después de validar el correcto funcionamiento del piloto por puntos se observó que este piloto solo sería válido para ciertas situaciones concretas por lo que se desarrolló un segundo piloto, el piloto por trayectorias, este piloto es más complejo ya que tiene variaciones de velocidad lineal y angular (en el ángulo de yaw), al introducir giros el sistema presentaba comportamientos muy inestables. Para solucionar este comportamiento se realizó un estudio de relaciones de velocidades y se observó que el rango de velocidades para el funcionamiento estable del sistema de navegación era de [0.3-3.2]m/s. Del mismo modo, los valores de ganancia de giro debían estar comprendidos entre el rango [0.14-0.36]. Esta ganancia de giro era muy importante ya que valores muy altos significaban comportamientos inestables y valores muy bajos no permitían ajustar el giro de forma correcta.

A continuación se muestran diferentes ejemplos de rutas sencillas y complejas, en los cuales se ha comparado con pilotos anteriores de trayectorias para observar su mejora tanto de errores de distancias a la ruta como de tiempo de realización de rutas. En verde se muestra la ruta a seguir que es la que se le introduce al programa, en azul se ve la ruta que realiza el piloto antiguo y en rojo se muestra la ruta que realiza

nuestro piloto.

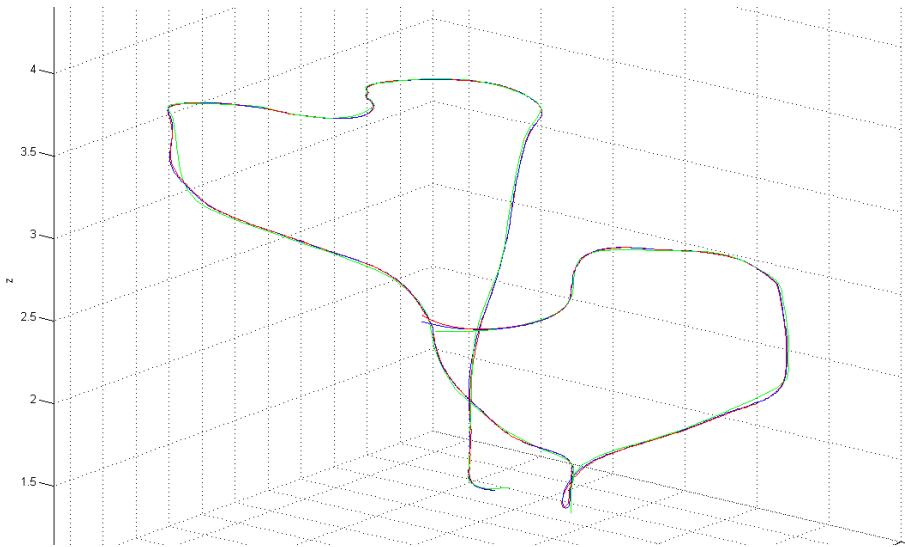


Figura 5.5: Ruta sencilla en el pilotaje por trayectorias.

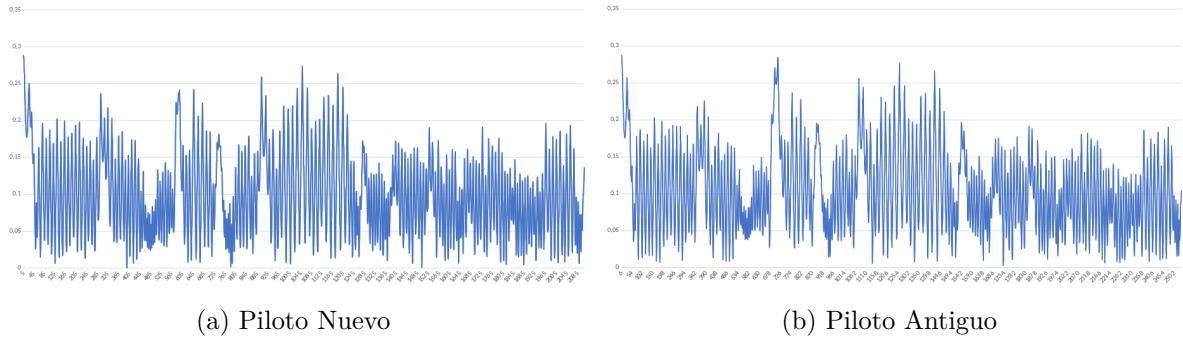


Figura 5.6: Comparación del error entre ambos pilotos en ruta sencilla.

En la imagen 5.5 se puede observar como ambos pilotos son muy exactos a la hora de realizar las trayectorias deseadas que no cuentan con cambios de dirección muy bruscos ni con curvas muy cerradas, sobretodo en las partes rectas de las mismas donde se ve como el ajuste es prácticamente perfecto. En cuanto a las partes mas difíciles como las curvas algo cerradas o los cambios bruscos de posición se observa en las gráficas 5.6 donde calculando el error obtenemos que el error de distancia media

en nuestro piloto es de 0.1023m con un error máximo en ruta de 0.276m en cambio el piloto antiguo tiene un error medio de 0.1026m y un error máximo de 0.285m. Este error no crea una diferenciación demasiado grande en cuanto al piloto nuevo y al piloto antiguo en rutas sencillas, pero si nos centramos en el tiempo en que realizan la ruta el piloto nuevo tarda 188.72 segundos mientras que el antiguo utilizaba 253.35 segundos lo que supone una reducción del 25,51 % del tiempo en vuelo algo muy importante debido a la corta autonomía de los drones.

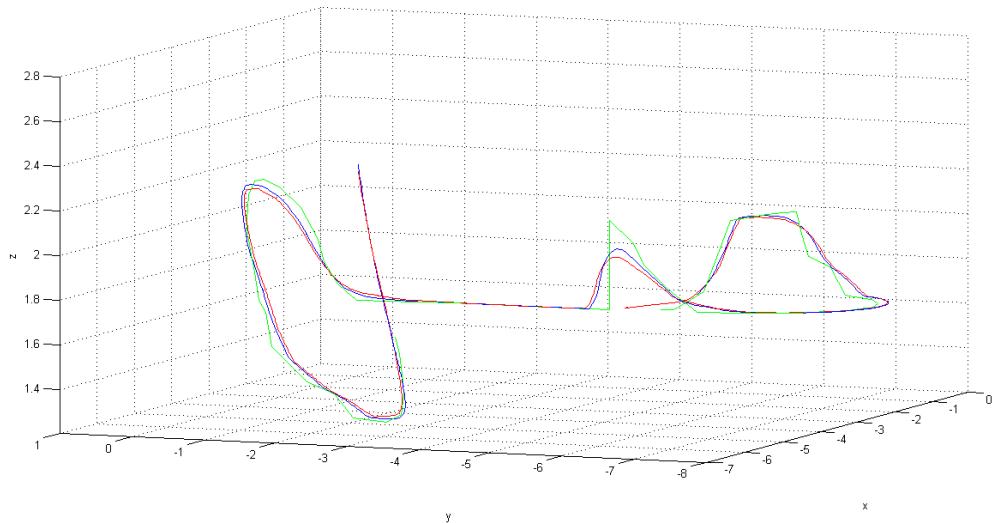


Figura 5.7: Ruta compleja en el pilotaje por trayectorias.

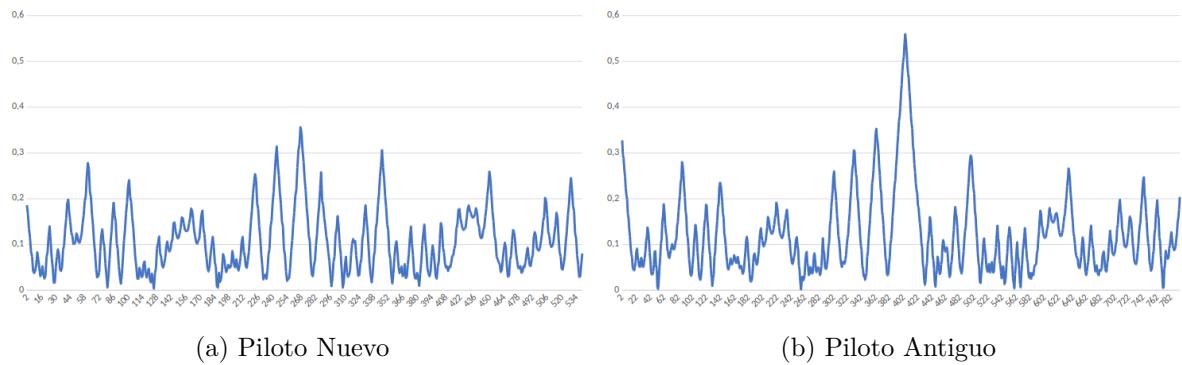


Figura 5.8: Comparación del error entre ambos pilotos en ruta compleja.

Para poder observar mejor el error en cuanto a distancia sobre la ruta deseada del pilotaje, sometimos los pilotos a una trayectoria con curvas mucho mas pronunciadas y con cambios de altitud bruscos. Esta trayectoria se puede observar en la imagen 5.7 donde tenemos la trayectoria deseada en verde, el piloto nuevo en azul y el piloto antiguo en rojo. En esta imagen si que se ve con mayor claridad como el piloto nuevo se ajusta mucho mas a la ruta deseada, se corrobora con los datos de las gráficas 5.8 en los cuales el error medio del piloto nuevo es de 0.106m y el error máximo es de 0.356m mientras que el error medio del piloto antiguo es de 0.119m con un error máximo de 0.559m. A su vez el nuevo piloto tarda 59.85 segundos en realizar la ruta mientras que el antiguo lo hace en 84.82 segundos reduciendo en este caso el tiempo de ruta en un 29.44 % por lo que en rutas complejas el nuevo piloto se desenvuelve mejor que el antiguo en todos los aspectos.

### 5.3. Pruebas integrales del sistema

Una vez conseguido que los componentes por separado funcionasen correctamente y demostrando que hemos conseguido errores mínimos en su comportamiento, se integraron ambos componentes conjuntamente y se realizaron las pruebas finales del sistema. Pese a la complejidad de unir los dos componentes ya que cada uno utilizaba unas velocidades máximas del drone y unos datos diferentes de posicionamiento, cuando se consiguió combinar y al haber realizado las pruebas por separado con buenos resultados, se esperaba que al unirlos los resultados siguiesen siendo óptimos. Esto se corroboró con los experimentos tanto de la ruta por puntos 5.9 donde tenemos en verde la ruta deseada, en rojo la posición estimada y en azul la ruta realizada, como de la ruta por trayectorias 5.10 que analizaremos a continuación con mas detalle ya que es el experimento final mas complejo, aun así en ambas se ve como el drone seguía la ruta de forma estable y con un margen de error prácticamente nulo, siempre y cuando estén a la vista del vehículo balizas AprilTags.

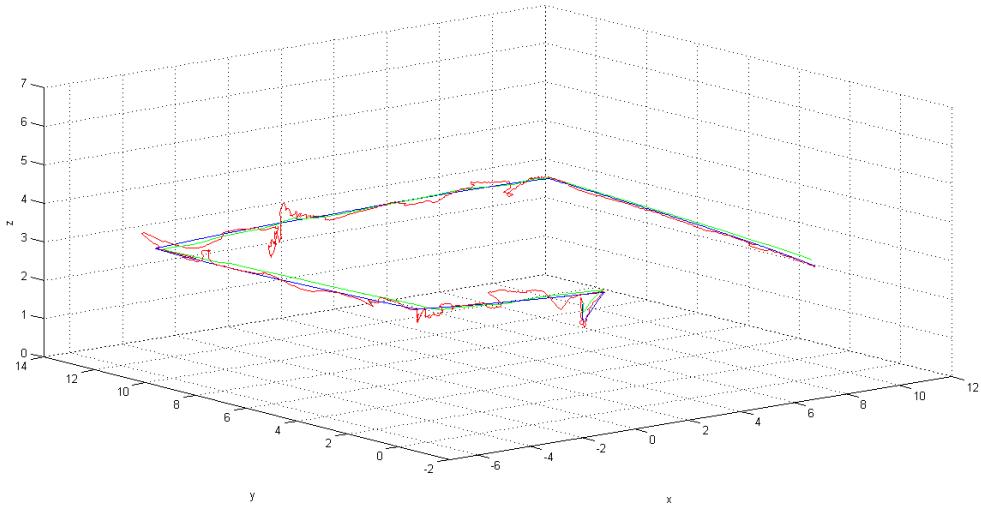


Figura 5.9: Prueba integral de ruta por puntos.

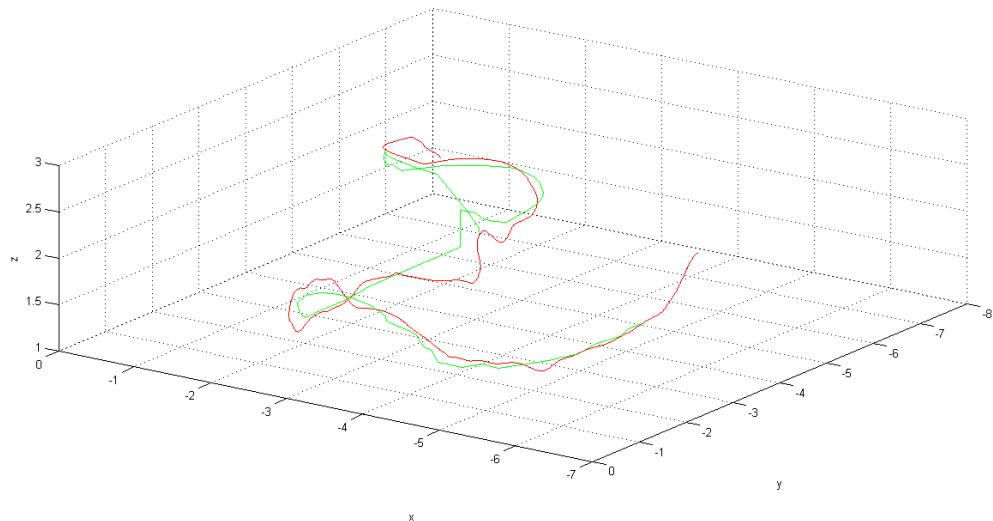


Figura 5.10: Prueba integral de ruta por trayectoria.

Para la realización del experimento final decidimos utilizar una trayectoria compleja con curvas cerradas y saltos de altura, así comprobar la integridad y el error

de la aplicación para cualquier situación. En la imagen 5.10 se observa la ruta deseada en verde y la ruta realizada en rojo. A primera vista no se aprecian errores demasiado elevados, por lo que calculamos la gráfica 5.11 para obtener los datos numéricos reales. De esta extraemos que el error medio de distancia en una ruta compleja es de 0.128m con un error máximo de 0.563m, debido a que el reconocimiento de las balizas no se puede realizar a velocidades elevadas el tiempo de ruta ha aumentado a los 109.5 segundos.

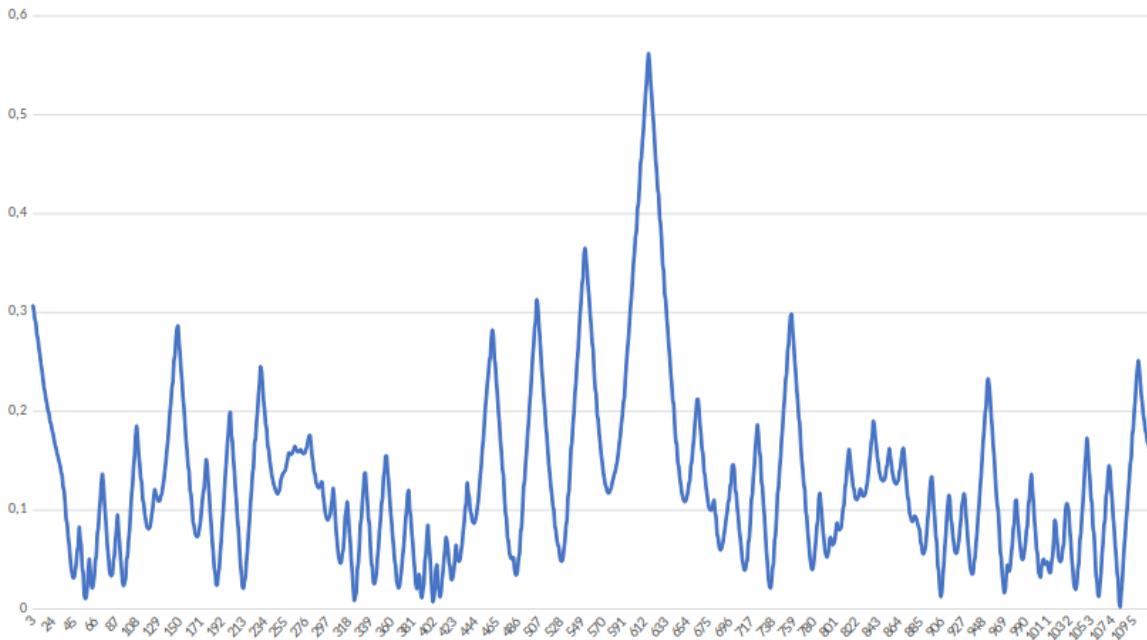


Figura 5.11: Error de distancia a la ruta deseada en la prueba integral del sistema

Por último anotar que el piloto cuenta con un margen de error de 15cm, lo que permite al vehículo realizar maniobras más amplias para evitar giros abruptos y así conseguir estabilidad durante toda la ruta. También destacar las nuevas funcionalidades que lo preparan para el mundo real que son la función en el pilotaje de que cuando un punto no lo puede alcanzar correctamente por una limitación en sus movimientos aterriza y no se quede infinitamente dando vueltas sobre este punto sin probabilidad de éxito, y la nueva función basada en el nuevo Slam-VisualMarkers de Ros utilizando la

constante del número de balizas en vista, creada explícitamente para ello, esta permite que en el momento que esta 2 segundos sin ver ninguna AprilTag el drone cese en su intento de seguir la ruta ya que no esta recibiendo una localización real y que gire en círculos cada vez mayores en torno al ultimo punto para así encontrar la baliza anterior y poder volver a localizarse y seguir la ruta, si no lo consigue aterrizará para evitar que intente conseguir una autolocalización errónea y se pierda en su intento de seguir la ruta deseada.

# **Capítulo 6**

## **Conclusiones**

Una vez expuesto el desarrollo fundamental de este trabajo, en este capítulo vamos a analizar si los objetivos planteados anteriormente se han cumplido. A su vez comentaremos las posibles líneas de desarrollo que parten de este trabajo. Como visión global podemos decir que el trabajo ha sido satisfactorio, ya que se ha conseguido un algoritmo que permite al dron despegar, realizar una ruta y aterrizar, todo ello de manera completamente autónoma, como se deseaba en un primer momento.

### **6.1. Conclusiones**

Antes de extraer las conclusiones sobre los objetivos en el capítulo 2, mencionar que para conseguirlos se ha hecho uso de un gran número de herramientas y tecnologías que han tenido que ser comprendidas y adaptadas para el propósito buscado. Con ello se ha conseguido validar el funcionamiento integral del sistema desarrollado en simulación como un sistema estable y aunque el ruido de las medidas y el comportamiento de los soportes físicos pueden afectar el comportamiento del sistema, este ha probado ser lo suficientemente robusto para satisfacer las metas propuestas dentro de entornos reales.

En cuanto al análisis de los objetivos vamos a extraer conclusiones sobre cada uno de ellos superado:

1. **Utilización de la herramienta Visual States:** Se ha logrado desarrollar completamente el algoritmo dentro de esta herramienta, con ello queremos decir que se ha creado la jerarquía necesaria para poder pasar de un estado a otro mediante transiciones controladas. También se ha conseguido crear la interfaz gráfica de forma que sea sencillo para el usuario saber cual es el comportamiento del robot y distinguir en que estado se encuentra.
2. **Adaptación e integración del componente Slam-Visualmarkers:** Se ha conseguido introducir con éxito dentro de Visual States la nueva aplicación de JdeRobot implementada en ROS y utilizando sus nuevas variables creadas junto con este trabajo para mejorar la herramienta y ayudarnos a hacer el algoritmo mas robusto.
3. **Recodificación e integración del despegue y aterrizaje visual:** En el nuevo algoritmo el drone podrá despegar desde cualquier altura siempre que se encuentre sobre su baliza, para así centrarse sobre ella con el nuevo sistema de despegue controlado y poder iniciar la ruta deseada con la certeza de que el drone esta completamente nivelado. También mencionar que el aterrizaje se ha simplificado para eliminar los temporizadores y la maquina virtual de estados, introduciéndolo correctamente en el nuevo algoritmo.
4. **Desarrollo de dos componentes de navegación basados en la posición absoluta del drone:** Éxito para ambos controles de pilotaje ya que hemos logrado desarrollar y comprobar que, tanto el método de pilotaje por puntos, como el pilotaje por trayectorias, funcionan correctamente, con errores mínimos y con una robustez elevada en cualquier tipo de situaciones, demostrando así que se ha creado una mejora con respecto a los anteriores sistemas de pilotaje creados.
5. **Validación experimental en entorno simulado Gazebo:** Todos los resultados obtenidos en el capítulo 5 se han extraído a partir de las pruebas realizadas dentro de los mundos creados en este entorno de simulación tridimensional. Creando así por tanto ejemplos visuales de la robustez del algoritmo comple-

to, con la utilización de todos los módulos para dotar al sistema de autonomía completa.

Al cumplir el objetivo principal de la aplicación y todos los subobjetivos que se extrajeron del mismo para dividirlo podemos decir que se ha conseguido algo diferente a lo que se encuentra en los repositorios de JdeRobot ya que ahora cuenta con un algoritmo que permite a un drone navegar de forma completamente autónoma desde el despegue hasta el aterrizaje, siguiendo una ruta previamente establecida y utilizando una autolocalización propia, únicamente colocando una serie de balizas y pulsando el botón de inicio.

Todas las pruebas y los avances que se han ido desarrollando y validando se pueden ver en la wiki oficial del proyecto: <http://jderobot.org/Jsaizc-tfg>

## 6.2. Trabajos futuros

Como comentamos en la introducción, el mundo de la robótica, de la visión artificial y dentro de ambos de los drones, está en constante desarrollo por lo que cada día surgen nuevas necesidades y aplicaciones posibles para este tipo de robots, por lo que pienso que es interesante continuar la línea de este proyecto. A continuación se detallan algunas de estas líneas de mejora, que han surgido al investigar las herramientas, las nuevas aplicaciones y desarrollar nuestro propio algoritmo.

- Lo primero sería probar el algoritmo en situaciones reales, ya que, aunque hemos comprobado la robustez del algoritmo en entornos simulados y suponemos la misma dentro de entornos reales con las funciones creadas para tales propósitos, no hemos podido comprobar situaciones como la deriva propia del vehículo, que añade un mayor grado de complejidad al sistema de control; los sistemas de estabilización establecidos por el fabricante, que en ocasiones entran en conflicto con el control proporcionado por el componente; y el ruido de los sensores.
- Otra posible línea de investigación sería el cambio del sistema de autolocalización visual basado en marcadores por una autolocalización basada en un sistema de

odometría visual, evitando así el uso de balizas y localizándonos gracias a la percepción de objetos a nuestro alrededor pudiendo crear incluso mapas en 3D de lo que se encuentra a nuestro alrededor.

- Por último nombrar algunas mejoras sobre nuestro propio algoritmo para adaptarlo a otras necesidades reales como, por ejemplo, capturas de fotografías de puntos estratégicos. Esto puede ser de gran utilidad para su representación posterior en 3D y creación de mapas u objetos e incluso para tareas de vigilancia o monitorización de espacios.

# 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](http://blog.sandglasspatrol.com/index.php/articulos/41-militar/758-uavs-clasificacion-tendencias-y-normativa-de-espacio-aereo)