



**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

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

| | |
|--|-----------|
| Índice de figuras | 6 |
| 1. Introducción | 1 |
| 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 |
| 2. Objetivos | 20 |
| 2.1. Problemas a abordar | 20 |
| 2.2. Requisitos | 22 |
| 2.3. Metodología | 22 |
| 2.4. Plan de trabajo | 24 |
| 3. Infraestructura | 26 |
| 3.1. Gazebo | 26 |
| 3.2. Balizas visuales AprilTags | 27 |

| | |
|---|-----------|
| ÍNDICE GENERAL | 4 |
| 3.3. Biblioteca OpenCV | 28 |
| 3.4. Biblioteca NumPy | 30 |
| 3.5. Entorno JdeRobot | 30 |
| 3.5.1. Herramienta ColorTuner | 31 |
| 3.5.2. Plugin ArDrone2 en Gazebo | 32 |
| 3.5.3. Interfaz Pose3D | 32 |
| 3.6. Visual States | 33 |
| 3.7. Slam-Visualmakers | 34 |
| 4. Navegación autónoma en seguimiento de rutas 3D | 36 |
| 4.1. Diseño | 36 |
| 4.2. Componente de Autolocalización | 38 |
| 4.3. Componente de Control basado en estados | 40 |
| 4.3.1. Estado de despegue | 42 |
| 4.3.2. Estado de seguimiento de ruta por puntos de paso | 43 |
| 4.3.3. Estado de seguimiento de ruta por trayectoria continua | 45 |
| 4.3.4. Estado de aterrizaje | 47 |
| 5. Experimentos | 49 |
| 5.1. Pruebas unitarias de autolocalización | 50 |
| 5.2. Pruebas unitarias de despegue | 53 |
| 5.3. Pruebas unitarias de pilotaje | 53 |
| 5.3.1. Pilotaje por puntos de paso | 53 |
| 5.3.2. Pilotaje por trayectorias continuas | 55 |
| 5.4. Pruebas integrales del sistema | 59 |
| 6. Conclusiones | 63 |
| 6.1. Conclusiones | 63 |
| 6.2. Trabajos futuros | 65 |
| Bibliografía | 67 |

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 VisualStates sobre Gazebo | 33 |

ÍNDICE DE FIGURAS

6

| | |
|---|----|
| 3.7. GUI de Slam-Visualmarkers. | 34 |
| 3.8. Entradas y Salidas del nodo Slam-VisualMarkers. | 35 |
| 4.1. Diseño del sistema de navegación autónoma en 3D. | 37 |
| 4.2. Entorno del componente Slam VisualMarkers | 39 |
| 4.3. Diseño en cuatro estados de la navegación visual autónoma para seguimiento de rutas en 3D, usando la herramienta VisualStates. | 41 |
| 4.4. Interfaces de comunicación de Visual States. | 42 |
| 4.5. Baliza utilizada en Gazebo. | 43 |
| 4.6. Ejemplo de los estados en Visual States | 48 |
| 5.1. Mundo Gazebo ArDones1. | 50 |
| 5.2. Diferencia angular entre lo real y VisualMarkers | 51 |
| 5.3. Cálculo de errores de distancia y de ángulos | 51 |
| 5.4. Cálculo de posición de Slam-VisualMarkers | 52 |
| 5.5. Seguimiento de ruta por puntos de paso en 3D. | 54 |
| 5.6. Ruta sencilla en el pilotaje por trayectorias. | 56 |
| 5.7. Comparación del error entre ambos pilotos en ruta sencilla. | 57 |
| 5.8. Ruta compleja en el pilotaje por trayectorias. | 58 |
| 5.9. Comparación del error entre ambos pilotos en ruta compleja. | 59 |
| 5.10. Prueba integral de ruta por puntos. | 60 |
| 5.11. Prueba integral de ruta por trayectoria. | 60 |
| 5.12. Error de distancia a la ruta deseada en la prueba integral del sistema . | 61 |

Capítulo 1

Introducción

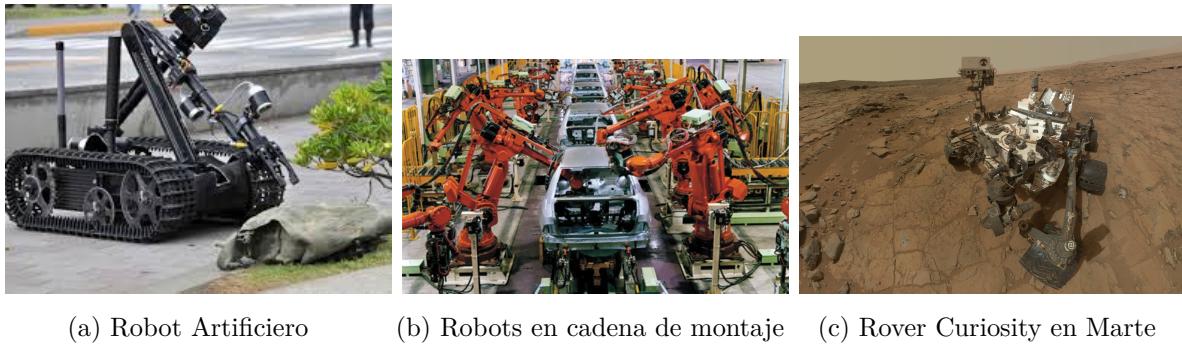
En este primer capítulo, antes de adentrarnos en la parte mas técnica, se va a introducir al lector de forma breve en qué es la robótica y más concretamente en los robots aéreos, para así conocer el estado actual de este campo y cómo ha evolucionado en los últimos años este sector, creando una gran rama 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^a 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^a 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^a 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^a 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.

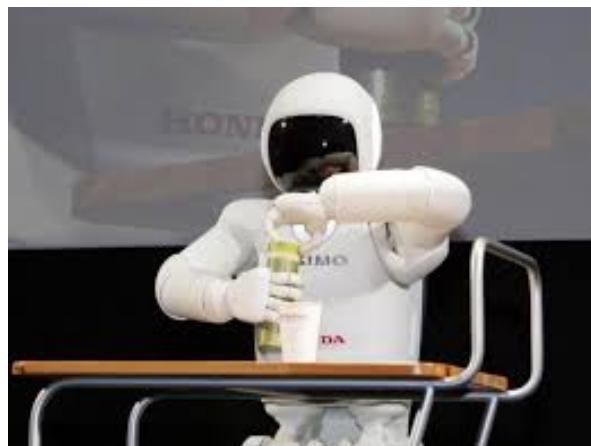


Figura 1.2: Robot ASIMO realizando acciones cotidianas.

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.



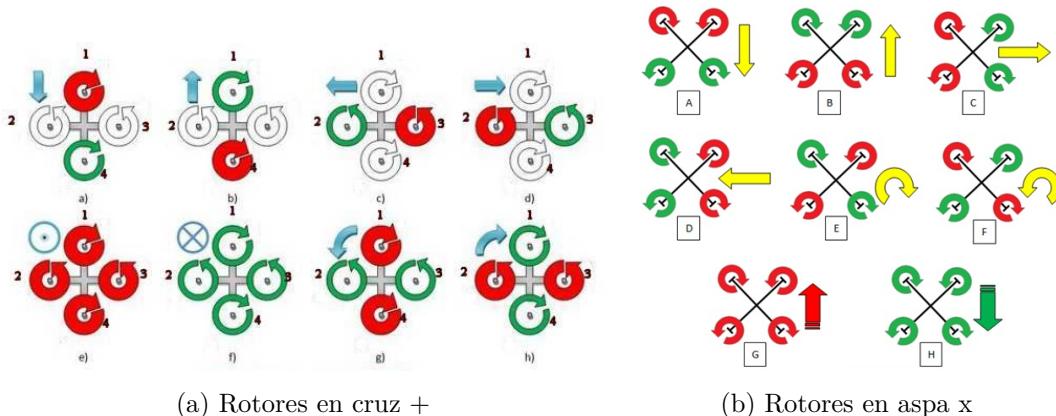
Figura 1.7: Drone ala fija Vs Drone ala móvil

- **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.



(a) Rotores en cruz +

(b) Rotores en aspa x

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 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 de incendios, ofrecer mejor detección de campo de minas y combatir la propagación de la enfermedad del sueño (Tse-Tse) mediante el control de mosquitos.

El *UAV Challenge - Outback Rescue* presenta un desafío abierto para adultos y un desafío para la escuela secundaria. El evento tiene como objetivo promover el uso civil de vehículos aéreos no tripulados y el desarrollo de sistemas de bajo costo que podrían usarse para misiones de búsqueda y rescate. Es uno de los desafíos de robótica más grandes del mundo y uno de los desafíos de UAV de mayor riesgo, el premio para el ganador es de 75.000 \$.



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 apagarlo¹.

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.

¹<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 ² [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.

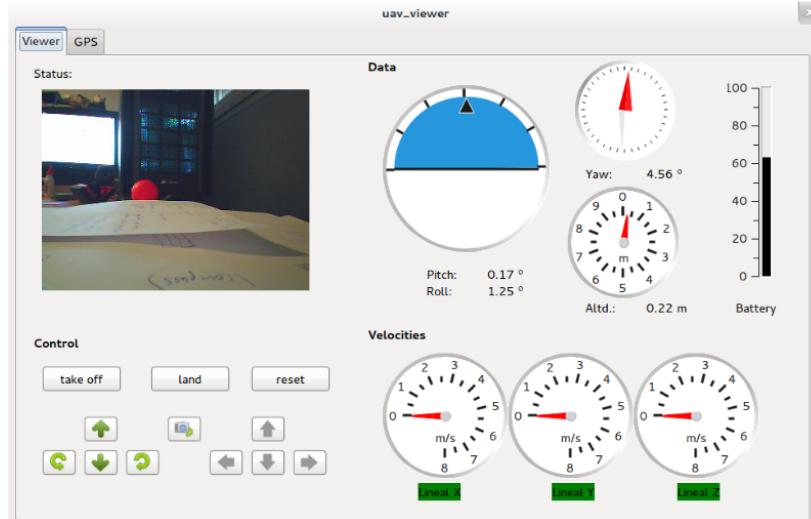


Figura 1.11: TFM Alberto Martín.

Daniel Yagüe ³ [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.

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

³<http://jderobot.org/Daniyague-pfc>

Alberto López-Cerón⁴ [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 Vélez⁵ [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 Zafra⁶ [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.

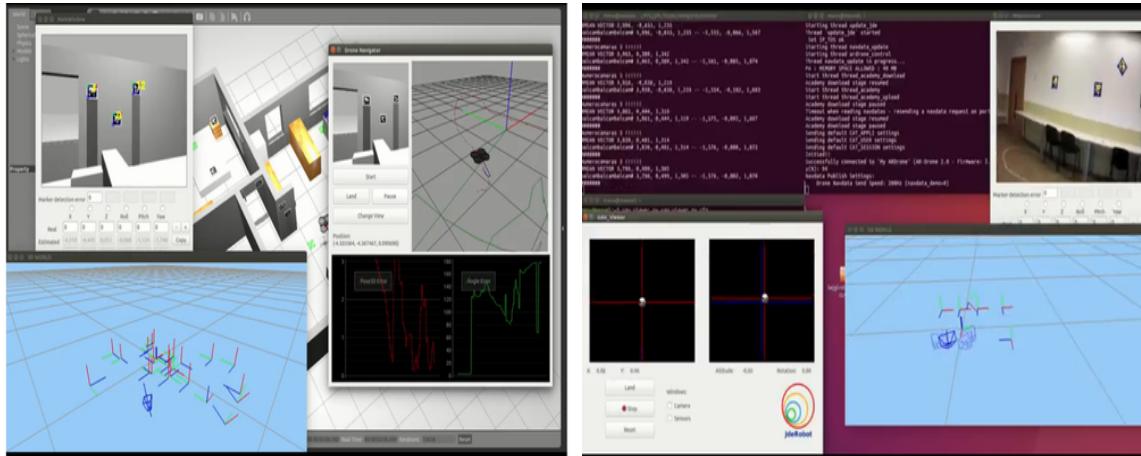


Figura 1.12: TFG Manuel Zafra.

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

⁵<http://jderobot.org/Avelez-tfg>

⁶<http://jderobot.org/Mazafra-pfc>

Jorge Vela ⁷ [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.

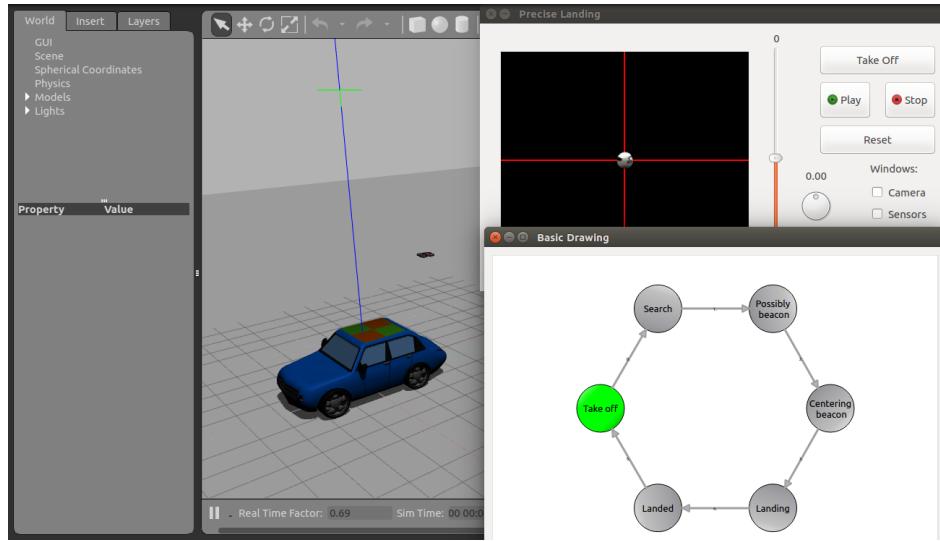


Figura 1.13: TFG Jorge Vela.

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.

⁷<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 debida al TFM de Alberto y al PFC de Manuel, pero nadie la había integrado dentro de un automata de estados finito, para ello 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.
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 ¹. 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 ².

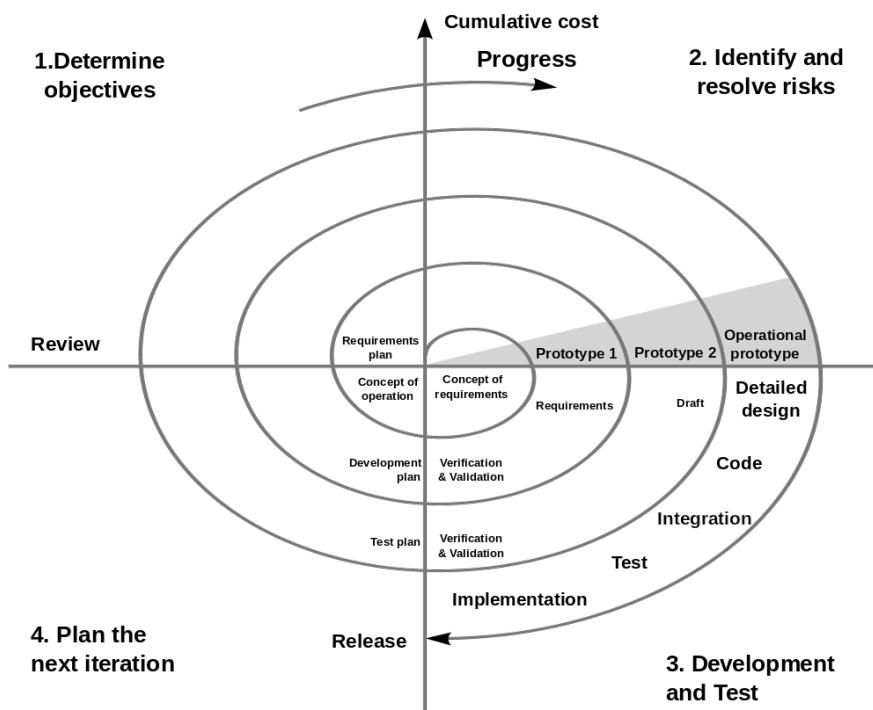


Figura 2.1: Representación del desarrollo en espiral.

¹<http://jderobot.org/Jsaizc-tfg>

²<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 explicará el funcionamiento de algunos componentes previos que hemos integrado.

3.1. Gazebo

El simulador Gazebo es un programa *OpenSource* distribuido bajo la licencia Apache 2.0 que se utiliza para 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.

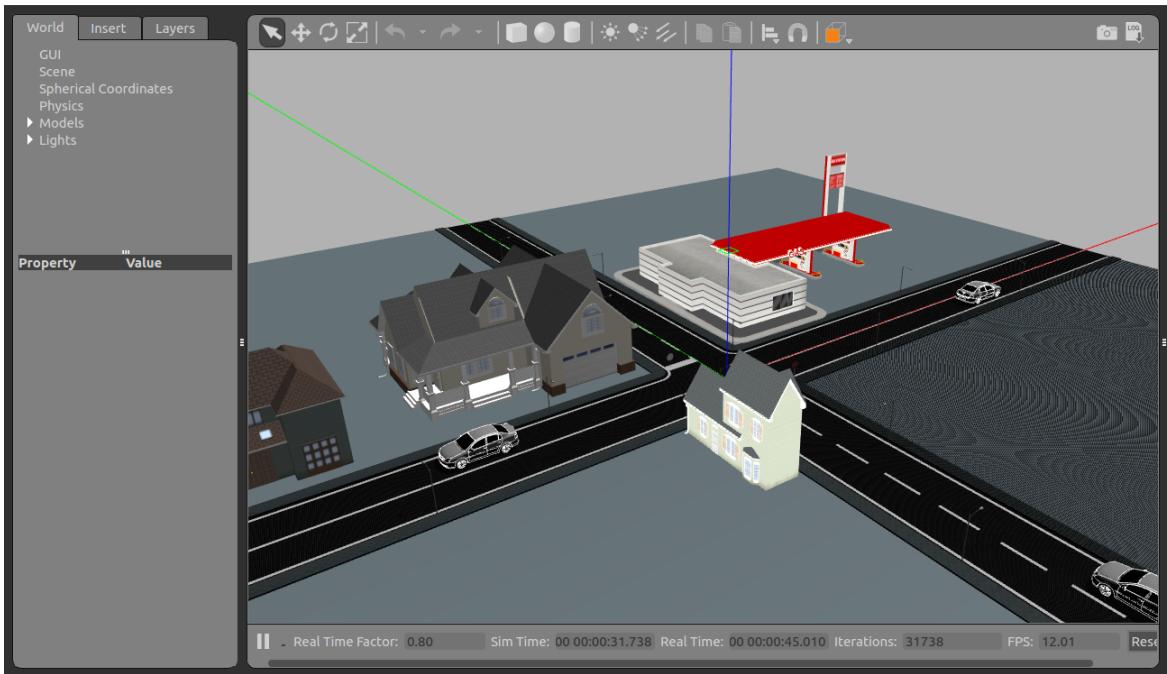


Figura 3.1: Entorno de Simulación Gazebo.

En este TFG se ha utilizado para simular al dron en un escenario con balizas y se ha programado la navegación autónoma de ese dron simulado, para ello se ha 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, pre-

senta un nuevo sistema de codificación que aborda problemas específicos de los códigos de barras 2D como es la robustez frente a la rotación y a los falsos positivos que pueden dar las imágenes naturales. El software de detección AprilTag calcula la posición, orientación e identidad 3D precisa de las etiquetas en relación con la cámara, además de su correspondiente ID. Eso lo realiza mediante un algoritmo de segmentación basado en gradientes locales que consigue que las líneas se estimen con precisión, el cual se implementa en C sin dependencias externas. Esto provoca una tasa de falsos negativos muy bajos, aunque aumenta la probabilidad de falsos positivos. Sin embargo, gracias a la codificación de las balizas esta probabilidad es reducida hasta niveles aceptables.

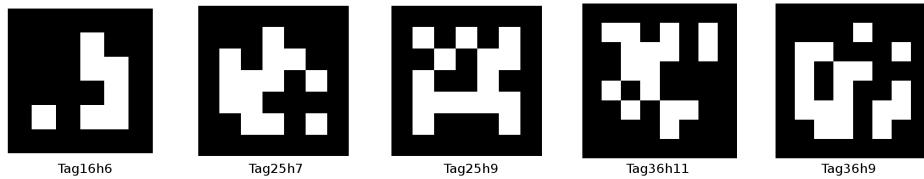


Figura 3.2: Ejemplos de AprilTags.

Esta biblioteca la hemos utilizado como ayuda al sistema de autolocalización de Slam-VisualMarkers y el código en C++ (que fue desarrollado primero por Edwin Olson y posteriormente por Michael Kaess) se puede obtener gratuitamente en su página ¹, nosotros hemos utilizado su versión 2.1 dentro del componente de localización de Slam Visual-Markers.

3.3. Biblioteca OpenCV

OpenCV ² es una biblioteca *OpenSource* 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

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

²<https://opencv.org/>

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

Gracias a OpenCV, al obtener la imagen que transmite el drone se puede tanto detectar objetos como aplicar cambios sobre ella, para marcar las zonas de interés o diferentes objetos. Permite la realización de filtros de color, por ejemplo 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 ruidos, que confunde objetos inexistentes o de no interés con 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 AprilTags según la distancia a la que se encuentran del drone y por tanto el peso que tienen a la hora de estimar la posición.

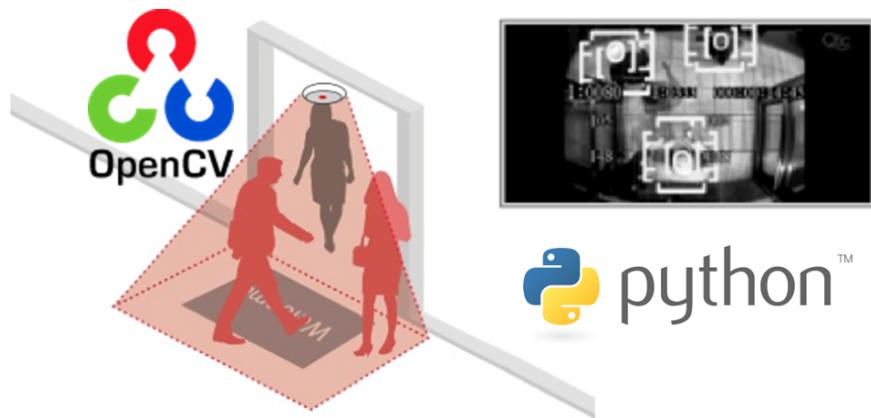


Figura 3.3: Ejemplo de algoritmo con OpenCV.

3.4. Biblioteca NumPy

Este TFG está escrito principalmente en el lenguaje de programación Python y NumPy³ es una extensión *open-source* de este lenguaje. Es el paquete fundamental para la informática científica con Python. Contiene, entre otras cosas, un poderoso objeto de matriz N-dimensional, herramientas para integrar el código C/C++ y Fortran, álgebra lineal útil, transformadas de Fourier y capacidad de trabajo sobre números aleatorios. Concretamente en este TFG se ha utilizado la versión Py3.5.

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

JdeRobot⁴ es un paquete de software libre para desarrollar aplicaciones de robótica y visión por computación. Estas aplicaciones 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 el que hemos trabajado y está mantenido por el grupo de robótica de la Universidad Rey Juan Carlos.

Para nuestro trabajo hemos utilizado la versión más reciente del repositorio, JdeRobot-5.6.3(Febrero 2018)⁵. Nos ha servido como herramienta para comprender

³<http://www.numpy.org/>

⁴http://jderobot.org/Main_Page

⁵<https://jderobot.org/Installation>

los componentes previos que hemos utilizado 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 obtener la gama de color que nos interesa o realizar un filtro sobre una imagen o vídeo. Utiliza un interfaz gráfico donde se representan dos imágenes, una la real y otra la filtrada, en ésta se pueden variar los parámetros para ver qué 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 los filtros de la aplicación final para que el drone las encuentre sin problemas.

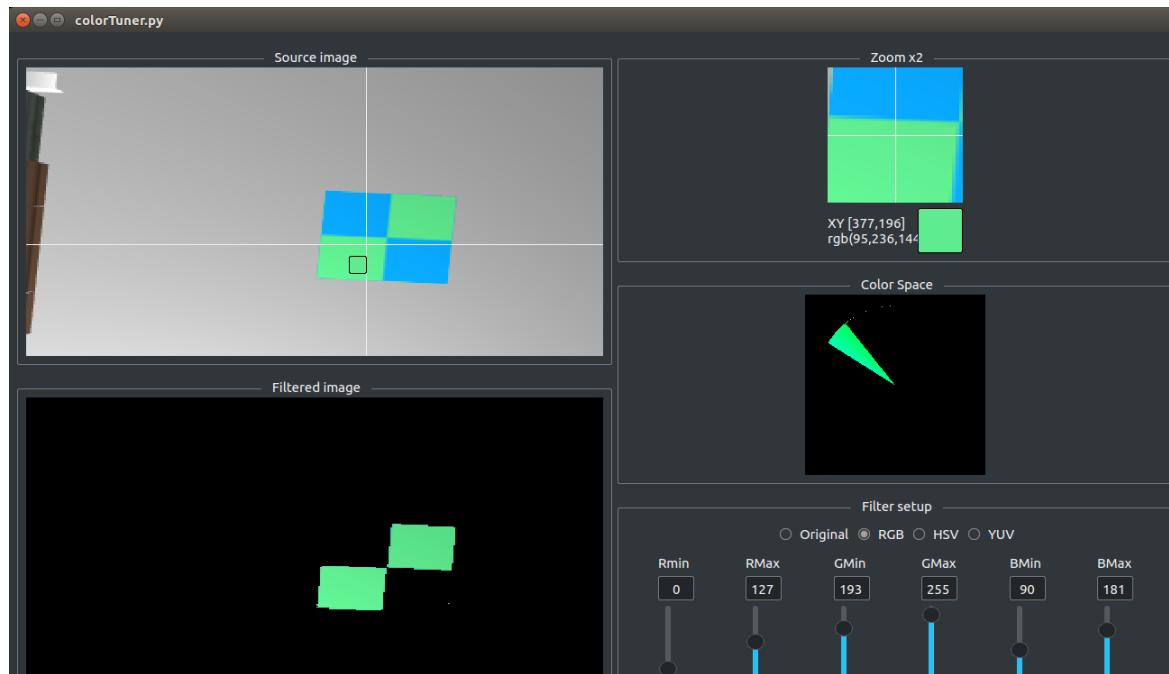


Figura 3.4: Herramienta ColorTuner.

3.5.2. Plugin ArDrone2 en Gazebo

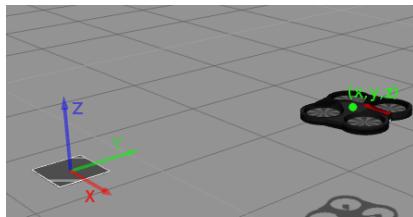
Hemos empleado para el desarrollo del proyecto el ArDrone 2.0 o más bien su modelo en el entorno de simulación Gazebo. Este plugin implementa una emulación realista y optimizada en simulación del drone, por lo que nos permite trabajar con los distintos sensores. Este drone cuenta con sensores IMU y con dos cámaras, una frontal y otra ventral, con ello el 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 interfaz *CMDVel* para dirigir sus movimientos y saber en todo momento en qué 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 está compuesta por un punto en 3D (el cual indica la posición en coordenadas cartesianas) y la orientación mediante los cuaterniones. A partir de éstos podremos obtener los ángulos de *roll*, *pitch* y *yaw*.

```
Pose3DData
{
    float x; /* x coord */
    float y; /* y coord */
    float z; /* z coord */
    float h; /* */
    float q0; /* qw */
    float q1; /* qx */
    float q2; /* qy */
    float q3; /* qz */
};
```

(a) Datos de Pose3DData



(b) Representación de los datos

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan \frac{2(q_0q_1 + q_2q_3)}{1 - 2(q_1^2 + q_2^2)} \\ \arcsin(2(q_0q_2 - q_3q_1)) \\ \arctan \frac{2(q_0q_3 + q_1q_2)}{1 - 2(q_2^2 + q_3^2)} \end{bmatrix}$$

(c) Matriz de Conversión Angular

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*). 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 sólo tiene que preocuparse por programar las acciones actuales del robot y seleccionar qué componentes puede necesitar de la interfaz del robot.

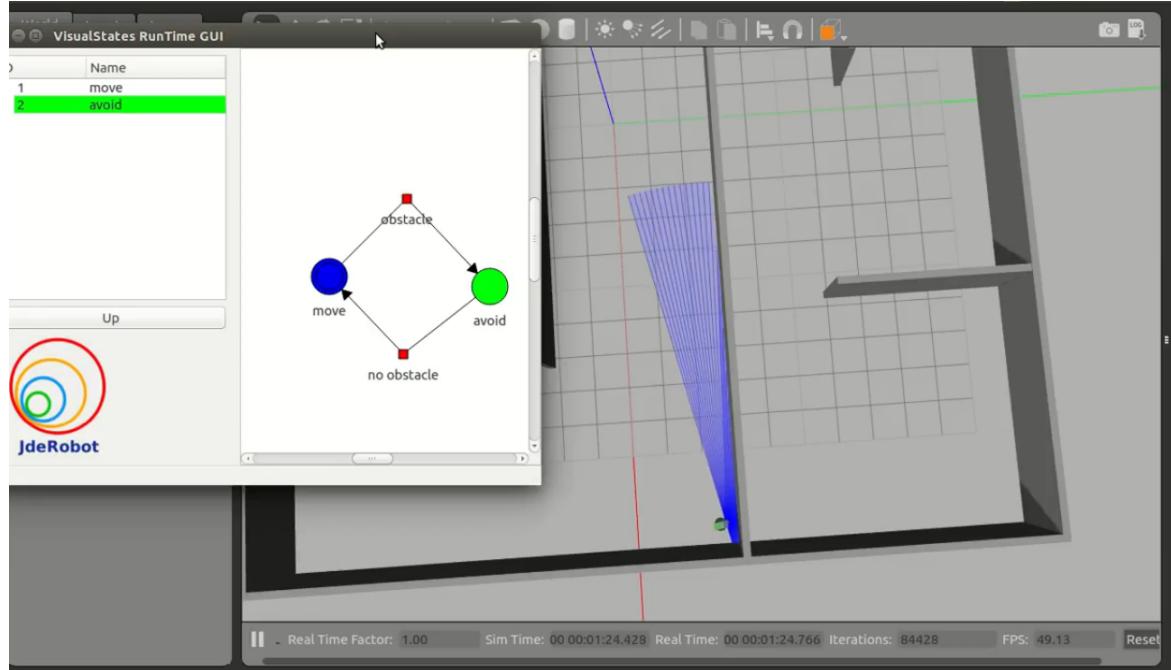


Figura 3.6: Herramienta VisualStates sobre Gazebo

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.

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 solucionarlos poco a poco hasta llegar al algoritmo final. Se puede encontrar una guía de la herramienta en la web ⁶.

3.7. Slam-Visualmakers

Es un nodo disponible en el ecosistema de JdeRobot que mediante una serie de algoritmos localiza en 3D la cámara a partir de balizas visuales en la escena. Es una aplicación desarrollada por Felipe Pérez ⁷ en su Trabajo Fin de Máster que ha ido creando paralelamente a este TFG. Está desarrollado en la plataforma JdeRobot y está programado en C++.

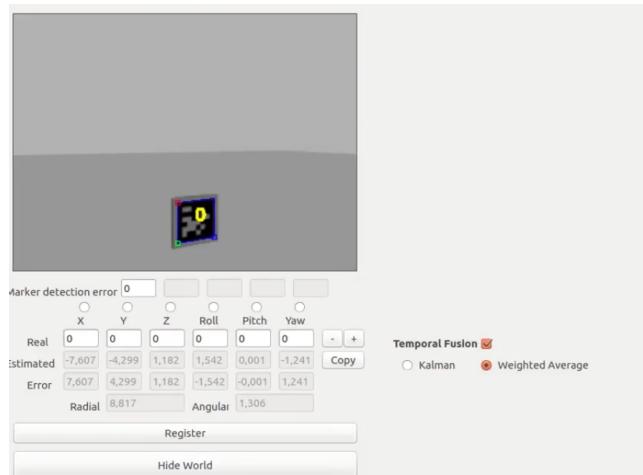


Figura 3.7: GUI de Slam-Visualmarkers.

⁶https://jderobot.org/Tutorials#VisualStates_tool

⁷<http://jderobot.org/Flperez-tfm>

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 absolutas de cada baliza visual ubicada en el entorno; y tercero, un fichero de configuración que contiene toda la información de los parámetros ópticos 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 y la función SolvePnP de OpenCV 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 la estimación obtenida a partir de cada baliza detectada en la imagen. La aplicación permite elegir qué clase de filtro temporal utilizar, pudiendo escoger entre un filtro por pesos o un filtro Kalman.

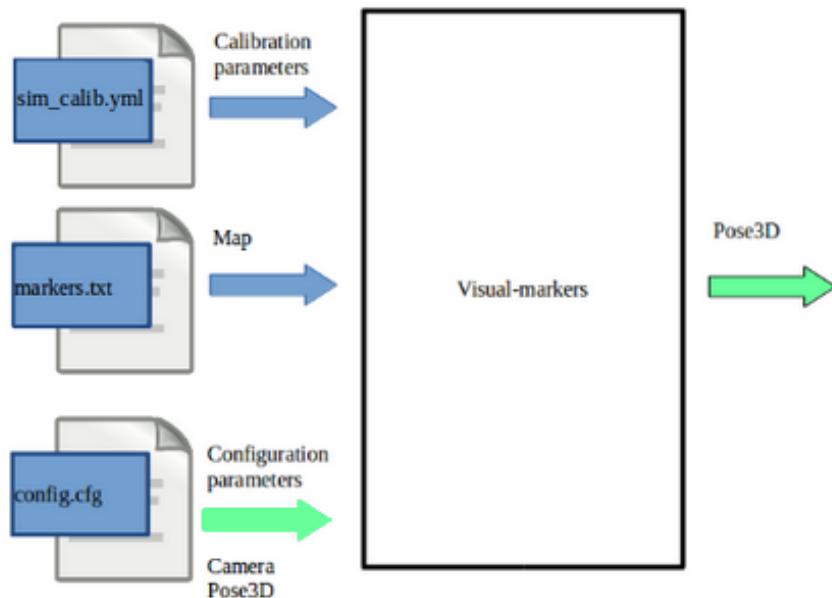


Figura 3.8: Entradas y Salidas del nodo Slam-VisualMarkers.

Capítulo 4

Navegación autónoma en seguimiento de rutas 3D

En este capítulo se describir el sistema diseñado y desarrollado para conseguir los objetivos planteados, utilizando la infraestructura mencionada anteriormente y como se ha implementado.

La solución ha sido un algoritmo de navegación autónoma sobre el cual se dará una visión global. A continuación, se explicará en detalle su diseño y el funcionamiento de cada uno de los componentes que lo integran. Por la complejidad inherente del problema, en este sistema ha sido necesario adaptar e integrar partes de software ya existentes previamente así como desarrollar otros bloques nuevos.

4.1. Diseño

El objetivo de este algoritmo es que un drone realice un comportamiento completamente autónomo desde el despegue, hasta el aterrizaje, pasando por recorrer de una ruta tridimensional previamente definida. Todo esto basándose únicamente en balizas de apoyo visual y en los motores de sus hélices. Es decir, el 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 la Figura 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 módulo para su correcto funcionamiento. Esta imagen nos da una idea del funcionamiento global de la aplicación, cada módulo es un proceso independiente que funciona iterativamente y como toda la comunicación entre procesos se lleva a cabo mediante la biblioteca ICE. A continuación vamos a explicar el comportamiento de cada módulo de una forma breve para tener una visión panorámica de la aplicación completa.

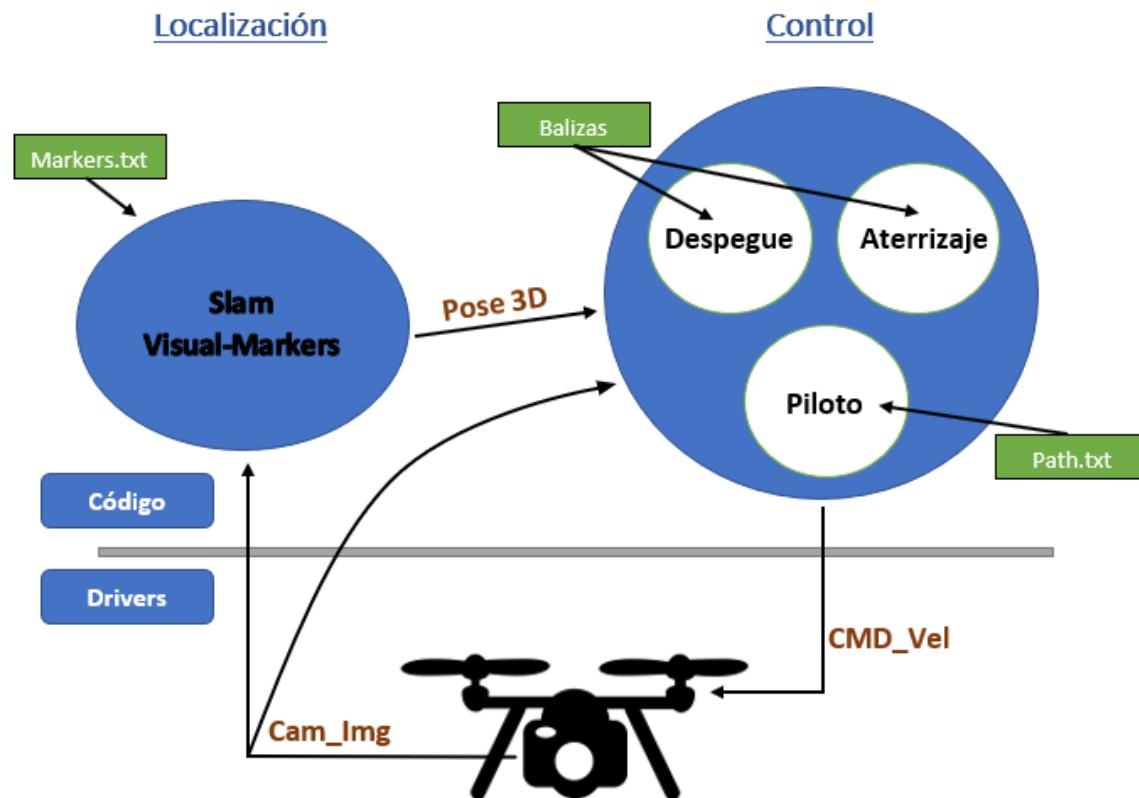


Figura 4.1: Diseño del sistema de navegación autónoma en 3D.

El nodo de localización **Slam VisualMarkers** recibe las imágenes que le proporciona el drone y envía los datos en forma de Pose3D(x,y,z,q0,q1,q2,q3) al componente de control. 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 por el interfaz ICE cuántas ha encontrado y aplica cálculos de geometría proyectiva para estimar la posición tridimensional 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 para elaborar su estimación de posición 3D. Finalmente, la estimación calculada se envía al componente de navegación o control mediante una interfaz ICE.

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

A continuación vamos a explicar cada módulo 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 será el apartado en el que más nos centraremos. Sin embargo, al ser un TFG de integración también explicaremos los módulos en los que nos hemos basado y hemos ajustado para su correcto funcionamiento en el algoritmo final.

4.2. Componente de Autolocalización

En cuanto al componente **Slam-VisualMarkers** está formado por el 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

CAPÍTULO 4. NAVEGACIÓN AUTÓNOMA EN SEGUIMIENTO DE RUTAS 3D

el fichero **Markers.txt** que contiene toda la información necesaria de cada marcador: id, tamaño y posición absoluta 3D.

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 en la imagen. 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 calculada estableciendo la correspondencia entre los puntos 2D de la imagen y los correspondientes puntos 3D de la baliza. 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 relativa del marcador con respecto a la cámara, y la matriz de posición del mundo con respecto al marcador.

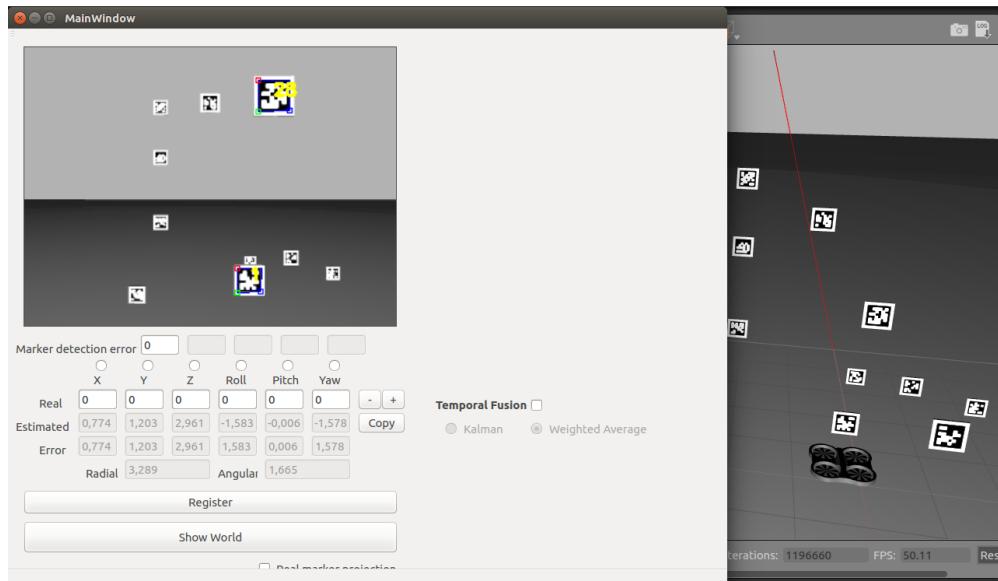


Figura 4.2: Entorno del componente **Slam VisualMarkers**

Las posiciones 3D estimadas para cada marcador observado en la imagen son almacenadas en un array y se fusionan mediante un filtro por pesos (fusión espacial).

Este filtro asigna diferentes pesos a cada estimación basándose en la distancia a la cámara, de forma que a los marcadores más cercanos se les asigna un peso mayor. En esta combinación ponderada, para 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 de dos maneras, que se elige por configuración: mediante un nuevo filtro por pesos, o mediante la aplicación de un Filtro de Kalman [9].

Con la nueva herramienta, además de calcular la posición tridimensional, también se incorpora el número de balizas detectadas en la imagen en las que se basa esa estimación y una marca temporal. Esto proporciona una indicación de la fiabilidad de la estimación y en qué momento se produce tal estimación.

4.3. Componente de Control basado en estados

Todo el algoritmo de navegación autónoma 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 de las diferentes partes del sistema en un solo programa.

Visual States tiene la capacidad de crear estados, donde dentro se escribe el código que se ejecuta cuando está activo, los cuales recorre mediante transiciones que pueden ser temporales o condicionales. Esto permite al programador centrarse en la escritura del algoritmo porque de la conexión entre estados se encarga la herramienta. Además, tiene otras secciones donde se especifican 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, **Visual States** ofrece la sección de las librerías, en la 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.

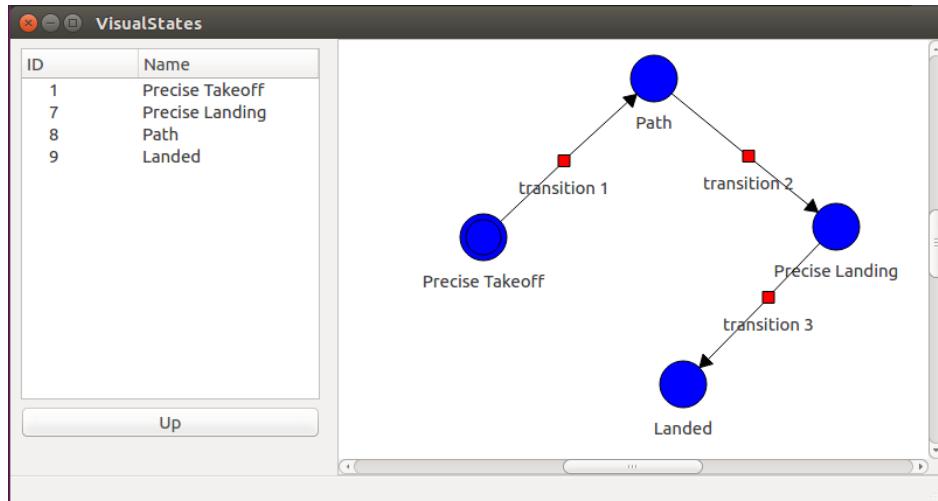


Figura 4.3: Diseño en cuatro estados de la navegación visual autónoma para seguimiento de rutas en 3D, usando la herramienta VisualStates.

En la Figura 4.3 se observan los diferentes estados y transiciones en los que se ha dividido la aplicación. El primer estado es el de *Precise Takeoff* en el cual el dron despegue y se posiciona sobre la baliza arlequinada de despegue, a continuación pasa por la primera transición en la cual se dejan de enviar ordenes de control para que el dron se estabilice y se cambia de la cámara inferior a la frontal. El siguiente estado es el de *Path* donde entra la herramienta de autolocalización para seguir la ruta establecida. Una vez concluida la ruta entra la transición dos en la cual se decide si el dron debe seguir realizando la ruta desde el inicio o pasar al modulo de aterrizaje. Éste seria el siguiente módulo *Precise Landing* en el cual se vuelve a cambiar la cámara a la inferior para buscar la baliza arlequinada de aterrizaje, una vez encontrada se procede al centrado y al descenso, hasta un punto en el cual la baliza ocupa la totalidad del objetivo de la cámara, que es cuando se para a la transición tres, donde se ponen todas las velocidades a cero menos la de descenso. Por último de llega al estado de *Landed* que significa que el dron desciende verticalmente hasta que la cámara esta sobre la baliza y se paran todos los motores.

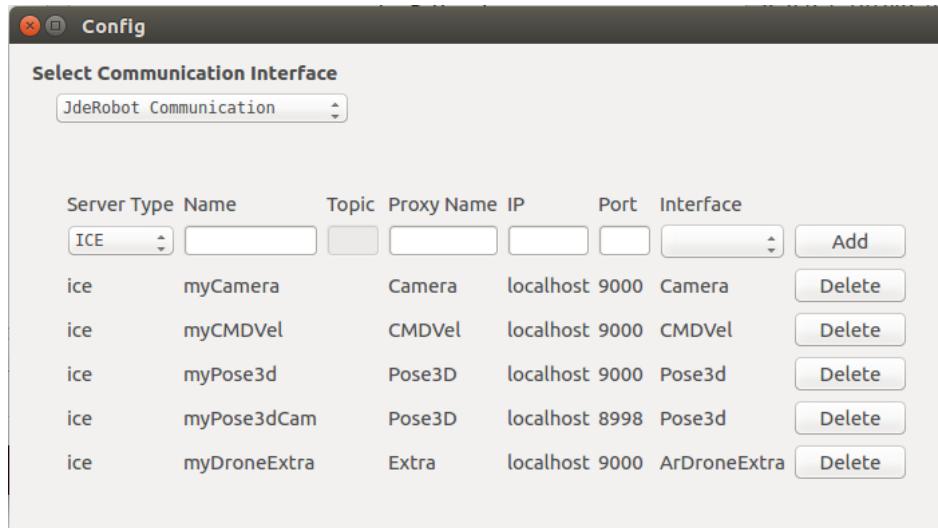


Figura 4.4: Interfaces de comunicación de Visual States.

La Figura 4.4 nos muestra los diferentes interfaces sobre los que se basa la aplicación, que cuadran con los explicados en la figura 4.1, estos son integrados por la herramienta para crear una correcta comunicación con el drone y así conseguir el correcto comportamiento de navegación desarrollado.

4.3.1. Estado de despegue

Para este componente se ha utilizado el trabajo que realizó Jorge Vela en su TFG [7] refactorizándolo y acoplándolo dentro de nuestro módulo de pilotaje en **Visual States**. Es el estado activo nada más arrancar la aplicación, el diseño de este algoritmo es un proceso basado en adquisición-procesado-envío de órdenes a los motores. La adquisición de los datos se realiza mediante los sensores de imágenes 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.

El lugar de despegue del drone es una baliza arlequinada previamente definida 4.5. 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.

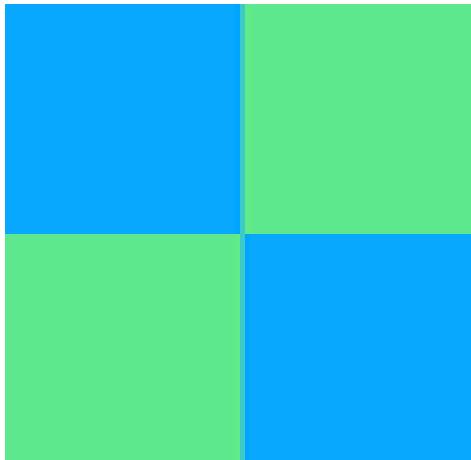


Figura 4.5: Baliza utilizada en Gazebo.

En el **despegue** se sitúa el drone sobre una baliza sobre la cual tiene que estabilizarse. De esta forma, al despegar detecta ésta y trata de centrarse, evitando así que se desvíe por factores externos como una ligera brisa o derivas en el propio movimiento, quedándose en la situación correcta. Para realizar el movimiento de centrarse en la baliza se ha utilizado un control PD (proporcional y derivativo) en el cual centra la cruceta de la baliza arlequinada con el punto central de la cámara, cuando esto se consigue y la baliza es una proporción de la imagen total se pasa al siguiente módulo que sería el del piloto.

4.3.2. Estado de seguimiento de ruta por puntos de paso

Como se puede observar en la imagen 4.3 el algoritmo de este componente se activa en el estado de Path y se encarga de que el drone siga una ruta. Hemos desarrollado dos algoritmos diferentes según el tipo de trayectoria que se le introduzca, si es una trayectoria de subobjetivos como puntos relativamente separados es el **Piloto**

por puntos de paso. Si en cambio la trayectoria es una ruta de puntos prácticamente continuos entonces se trataría del **Piloto por trayectoria**. Ambos algoritmos se basan en la posición tridimensional del drone. Para el control 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 son proporcionados por la estimación del componente de posición de **Slam VisualMarkers**. Y nuestro algoritmo de navegación devuelve una serie de velocidades que va 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 de paso al que debe llegar no variará prácticamente ninguno de sus ángulos de Euler, tan solo se moverá sobre los ejes x,y,z.

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

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

Una vez calculado este vector entre las posiciones, lo multiplicamos por un coeficiente regulador que reduce los valores hasta que éstos se encuentren dentro del rango de velocidades del drone y una vez alcanzado este rango se pasan los valores por una función que compruebe que las velocidades máximas no exceden las permisibles por **Slam VisuaMarkers** para la detección y análisis de balizas:

```
if math.fabs(Vx) > MaxVx :
    Vx = MaxVx * np.sign(Vx)
if math.fabs(Vy) > MaxVy :
    Vy = MaxVx * np.sign(Vy)
if math.fabs(Vz) > MaxVz :
    Vz = MaxVx * np.sign(Vz)
```

Con este método de control de navegación por posición nos aseguramos de que el drone alcanza el punto que queremos adecuadamente y a medida que se va acercando a él, para que la aproximación sea exacta, al reducirse el vector se reducirá la velocidad alcanzando la baliza con total exactitud. Una vez se alcanza la baliza se busca cual es la siguiente en la lista y se vuelve a realizar el proceso, así hasta alcanzar todas ellas.

4.3.3. Estado de seguimiento de ruta por trayectoria continua

En cuanto al **Piloto por trayectoria continua** se creó para permitir un control mucho más fino del movimiento del drone y así seguir trayectorias más angulosas y con variaciones más fuertes. Lo primero fue crear las rutas que posteriormente seguiría el drone, como éstas debían ser largas con giros y con puntos muy consecutivos se creó en Python la función que nos permitiese crear éstas según la separación entre puntos de ruta que considerásemos necesaria, la ruta va en un fichero como secuencia de puntos muy próximos entre sí:

```
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

```

A diferencia del *Piloto por puntos de paso* éste seguirá los puntos orientándose siempre en la dirección entre la posición y el siguiente punto, para ello tendremos que variar la velocidad angular en torno al eje Z, modificando por tanto el ángulo de yaw del drone y de esta forma conseguir orientarlo de la forma más rápida hasta el siguiente punto de la ruta. Conforme se fue investigando se descubrió que permitir al drone realizar pequeñas variaciones en la velocidad lineal del eje Y permitía un aproximación a los puntos efectiva y precisa, por lo que se optó por incluir también la variación de la velocidad en el eje Y.

Para el cálculo de las velocidades lineales Vx, Vy y Vz 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 Wz. Esta velocidad angular se calculará mediante las funciones de predicción de posición.

El primer paso de esa predicción 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 Ψ_z sacamos la posición que predecimos sólo 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 calculamos 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, Ψ_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 ω_z ajustamos la velocidad en el eje Y, v_y , a partir de ésta.

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 ω_z se envían al drone mediante una llamada al método *SendCMDVel(vx,vy,vz,wz)* de Interfaces, que se ocupa de mandar las órdenes decididas al cuadricóptero.

4.3.4. Estado de aterrizaje

Una vez el cuadricóptero termina la ruta establecida, éste procede a la siguiente etapa que es el **aterrizaje**, el cual se divide en dos partes, primero la búsqueda

CAPÍTULO 4. NAVEGACIÓN AUTÓNOMA EN SEGUIMIENTO DE RUTAS 3D

de la baliza de aterrizaje y luego el centrado de la misma y el descenso sobre ésta.

En cuanto a la búsqueda, sigue una navegación en espiral creciente de forma que irá rastreando la zona ampliando su giro de forma continua, hasta que detecte una baliza arlequinada. Si no se detecta una baliza, continuará el algoritmo de búsqueda, aumentando la amplitud de las espirales. Cuando se detecte, dejará el movimiento en espiral y buscará 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, ya que la percepción visual de la baliza es la misma que en el estado de despegue.

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

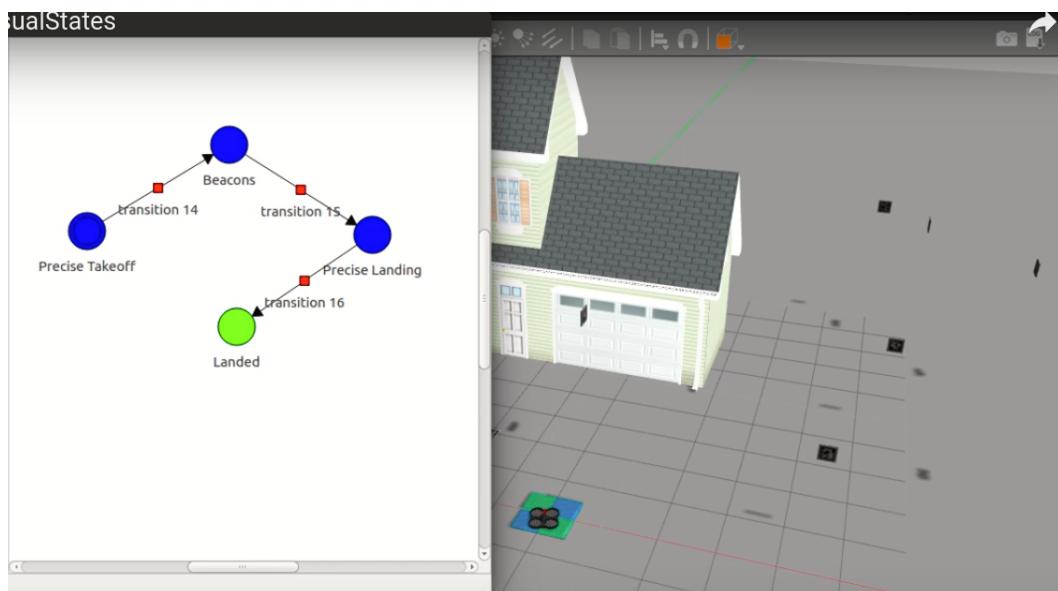


Figura 4.6: Ejemplo de los estados en Visual States

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 capítulo en tres partes a analizar: la primera parte, se ha comprobado la calidad de autolocalización, es decir, se ha realizado una ruta teleoperada para ver los errores de posición de la aplicación; la segunda parte, se ha evaluado la calidad del controlador, es decir, el error que comete el piloto al realizar una ruta predefinida y por último, se ha validado el funcionamiento de la aplicación final con todos los componentes a la vez.

Dada la complejidad de los experimentos, 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 realizar correctamente los diferentes experimentos se han creado una serie de mundos en 3D, los cuales cuentan con espacios abiertos, espacios cerrados, balizas arlequinadas y balizas AprilTags con sus correspondientes texturas. Los mundos creados se han integrado en el repositorio oficial de JdeRobot bajo los nombres de *ArDrone1*, *ArDrone2* y *ArDrone3*.

En la Figura 5.1 se puede observar el mundo ArDrones1, que fue el primero que creamos y el más representativo ya que se puede observar una posible aplicación

de reconocimiento de un casa mediante el dron completamente automatizado. Cuenta con las balizas arlequinadas de despegue (verde-naranja) y aterrizaje (verde-azul), con balizas 23 AprilTags para la autolocalización que rodean una casa a la cual el dron tiene que darle una vuelta completa.

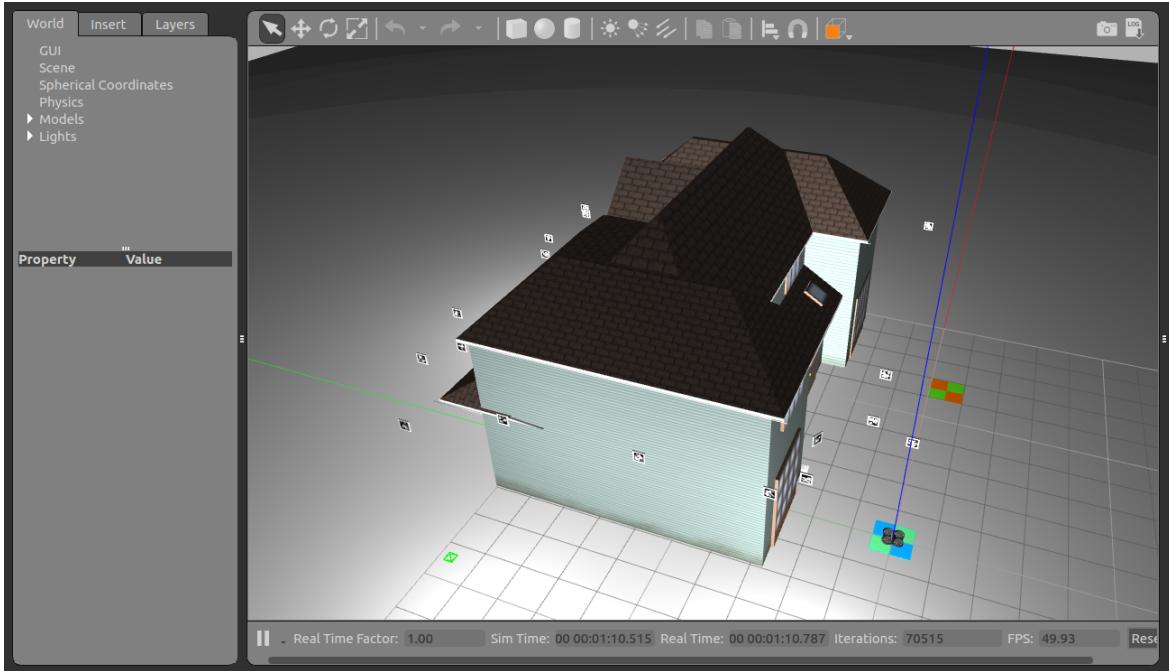


Figura 5.1: Mundo Gazebo ArDones1.

Tanto para el cálculo de trayectorias, como para el cálculo 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.

5.1. Pruebas unitarias de autolocalización

Para evaluar el comportamiento del componente `Slam VisualMarkers` que se encarga de la autolocalización del dron, se aisló completamente de la parte de pilotaje. Para ello lo que se hizo fue teledirigir a mano el dron dentro de un mundo de balizas el cual permitiese al componente ver al menos una baliza AprilTag en todo momento de la

ruta. Una vez terminada la ruta se comparó la secuencia de posiciones reales entregada por el simulador P_0 con la secuencia de posiciones estimadas por el componente P_A . Dado que ambas entregan 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*:

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

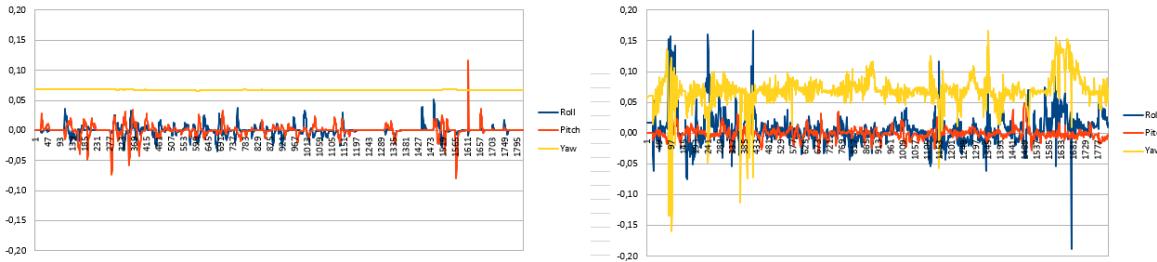


Figura 5.2: Diferencia angular entre lo real y VisualMarkers

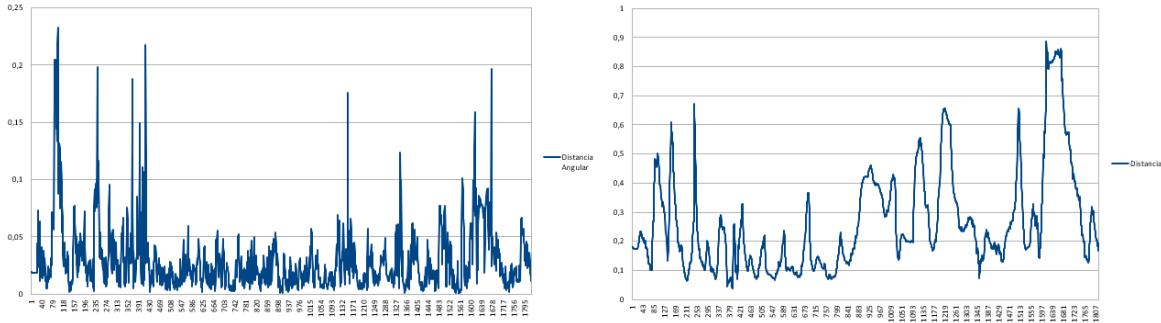


Figura 5.3: Cálculo de errores de distancia y de ángulos

Con la aplicación de las fórmulas anteriores se obtuvieron las gráficas de la figura 5.3. En las gráficas superiores se ve la diferencia entre los tres ángulos pitch, roll y

yaw del drone real (izquierda) con los calculados por el componente 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 izquierda de la Fig 5.3 se ve como el error angular medio es de 0.0293rad lo que equivale a 1.679 grados, que es un error muy pequeño, el error angular máximo es de 0.2329rad que son 13.344 grados. Por último, la gráfica derecha de la Fig 5.3 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 sólo ve una baliza y esta está alejada son cifras bastante razonables.

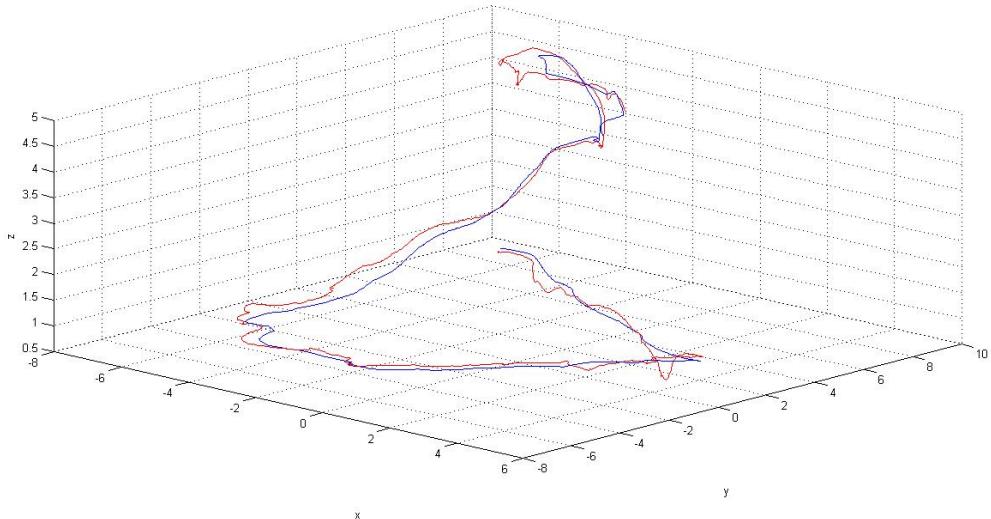


Figura 5.4: Cálculo de posición de Slam-VisualMarkers

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 Figura 5.4, en la cual tenemos la ruta realizada en azul y la ruta estimada por el componente en rojo. Para evitar esto se

habló con el desarrollador de Slam VisualMarkers y se añadió más información a su interfaz de salida en ROS de modo que ya indicaba el número de balizas que se estaban viendo. Esta caracterización experimental de la autolocalización permitió acotar los márgenes de error con los que tenía que lidiar el nodo de pilotaje y permitió mejorar la navegación autónoma contemplando la situación en la que el drone se perdía. Con esto se pudo crear un nuevo comportamiento que permite que cuando el drone no esté viendo ninguna baliza no siga la ruta, sino que gire sobre sí mismo realizando círculos, cada vez mayores, sobre el último punto donde veía alguna baliza, para así poder volver a encontrarla y posicionarse correctamente. Esto ayudó a que el drone no se perdiere 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 de navegación autónoma y nos aseguramos de que el cuadricóptero no se aleje excesivamente de la ruta definida, preparándolo para los experimentos reales.

5.2. Pruebas unitarias de despegue

5.3. Pruebas unitarias de pilotaje

A diferencia de las pruebas anteriores, en estas pruebas se suministraba al piloto la posición verdadera del drone en todo momento, extraída del simulador. Así, los errores entre la ruta deseada y la realmente conseguida son exclusivamente achacables al pilotaje, no a la autolocalización.

5.3.1. Pilotaje por puntos de paso

El primer piloto desarrollado se encargaba de seguir únicamente puntos de paso, es decir tan solo variaría la velocidad lineal del drone. Los primeros experimentos mostraban como el piloto se acercaba correctamente al punto, pero en 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 de paso. Una vez conseguido 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 de paso, pero aun así, la velocidad media del drone continuaba siendo el 0.82 de la máxima.

En la Figura 5.5 se puede ver una ruta por puntos compleja (en verde) y cómo ajustando la velocidad cercana a los puntos, la ruta final del drone es más exacta (rojo) que permitiéndole una velocidad máxima mayor (azul). Con todos estos ajustes y alternativas probadas, al final se ofrece un piloto que se puede configurar para que realice la ruta con mayor rapidez o con una mayor exactitud, recalculando que el error introducido por el pilotaje es aceptable, que consigue pasar por todos los puntos de paso especificados con un error por debajo de 0.02m.

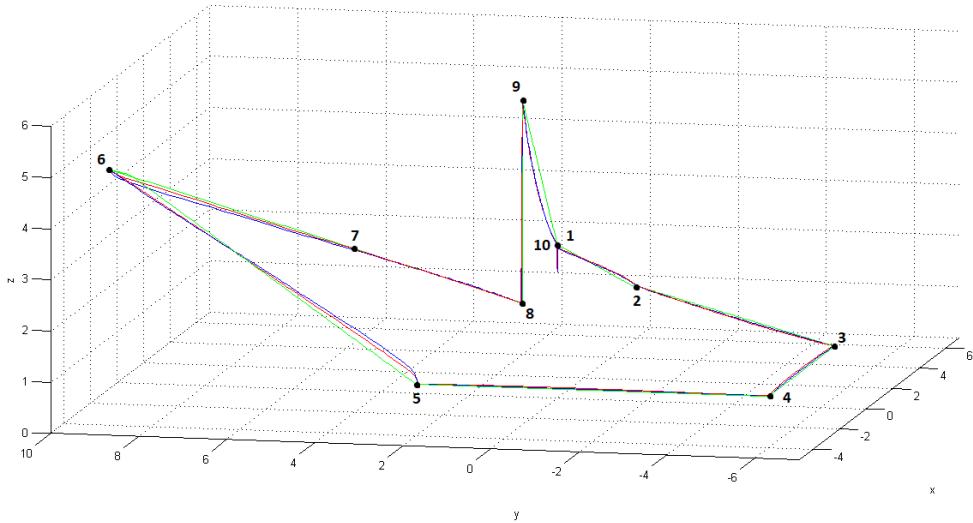


Figura 5.5: Seguimiento de ruta por puntos de paso en 3D.

5.3.2. Pilotaje por trayectorias continuas

Después de validar el correcto funcionamiento del piloto por puntos se observó que este piloto sólo sería válido para ciertas situaciones concretas de espacios abiertos, por lo que se desarrolló un segundo piloto, el piloto por trayectorias continuas que permite un control más fino de las posiciones del drone. Éste es más complejo ya que tiene variaciones de velocidad lineal y angular (en el ángulo de *yaw*), al introducir giros el sistema presenta comportamientos 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 un piloto anterior, el del PFC de Manuel Zafra [6], de trayectorias para observar la mejora conseguida tanto reduciendo los errores de distancias a la ruta como acortando el tiempo de realización de rutas.

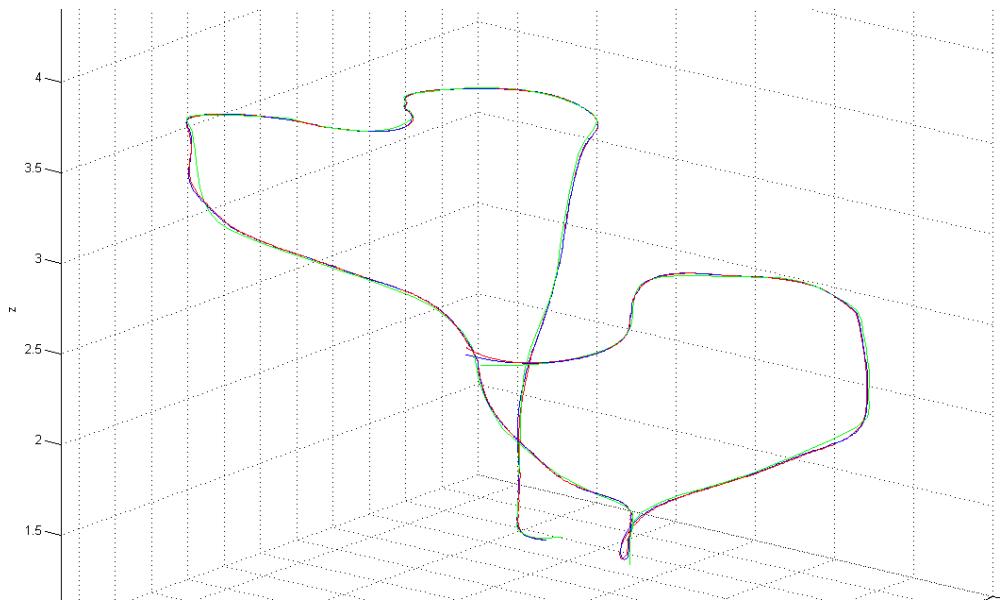


Figura 5.6: Ruta sencilla en el pilotaje por trayectorias.

Empezando con rutas sencillas, en la Figura 5.6 se distinguen: la trayectoria deseada (verde), que es la que se introduce al programa; la trayectoria que realiza el piloto antiguo (azul); y por último, la trayectoria que realiza nuestro piloto (rojo).

En la imagen 5.6 se puede observar cómo ambos pilotos son muy exactos a la hora de realizar las trayectorias deseadas suaves, que no cuentan con cambios de dirección muy bruscos ni con curvas muy cerradas, sobre todo en las partes rectas de las mismas donde se ve como el ajuste es prácticamente perfecto.

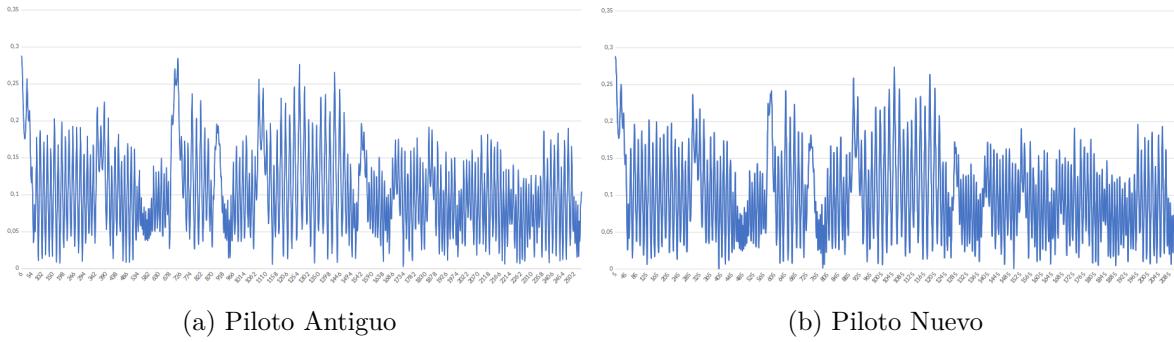


Figura 5.7: Comparación del error entre ambos pilotos en ruta sencilla.

En las gráficas 5.7 donde calculando el error obtenemos que el error medio de distancia 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 refleja una diferenciación significativa en cuanto al piloto nuevo y al piloto antiguo en rutas sencillas, en error espacial. Pero sí hay una notable diferencia 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.

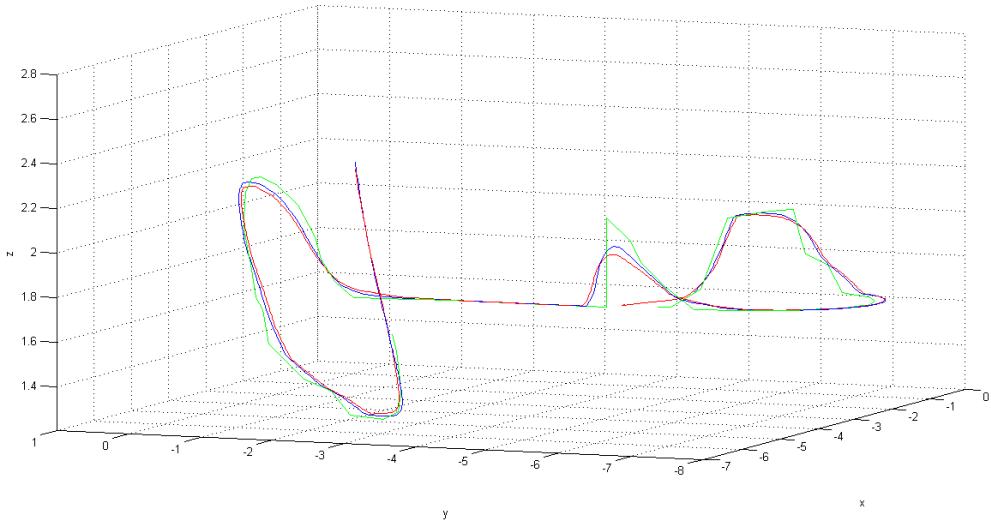


Figura 5.8: Ruta compleja en el pilotaje por trayectorias.

En cuanto a rutas complejas, para observar mejor el error en distancia, pedimos a los pilotos a una trayectoria con curvas mucho más pronunciadas y con cambios de altitud bruscos. Esta trayectoria se puede observar en la imagen 5.8 donde tenemos la trayectoria deseada en verde, el piloto nuevo en azul y el piloto antiguo en rojo. En esta imagen se ve con mayor claridad cómo el piloto nuevo se ajusta mucho más a la ruta deseada. Estas mejores prestaciones se corroboran con los datos de las gráficas de la Figura 5.9, 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.

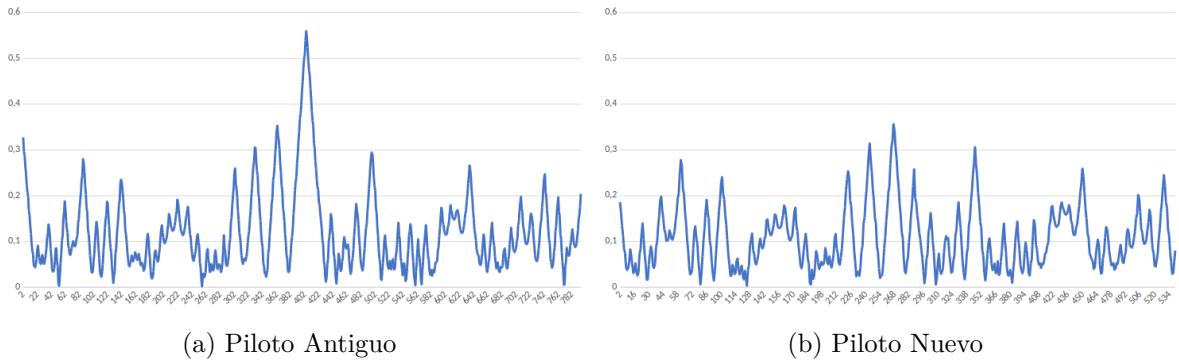


Figura 5.9: Comparación del error entre ambos pilotos en ruta compleja.

5.4. Pruebas integrales del sistema

Una vez conseguido que los componentes por separado funcionasen correctamente, se integraron ambos componentes conjuntamente y se realizaron las pruebas finales del sistema. Con ellas se valida experimentalmente la aplicación completa de navegación desarrollada. 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. El buen funcionamiento del sistema se corroboró con los experimentos tanto de la ruta por puntos de paso (en la Figura 5.10) donde tenemos en verde la ruta deseada, en rojo la posición estimada y en azul la ruta realizada, como de la ruta por trayectorias continuas (en la Figura 5.11). En ambos 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 aéreo las balizas AprilTags para su autolocalización.

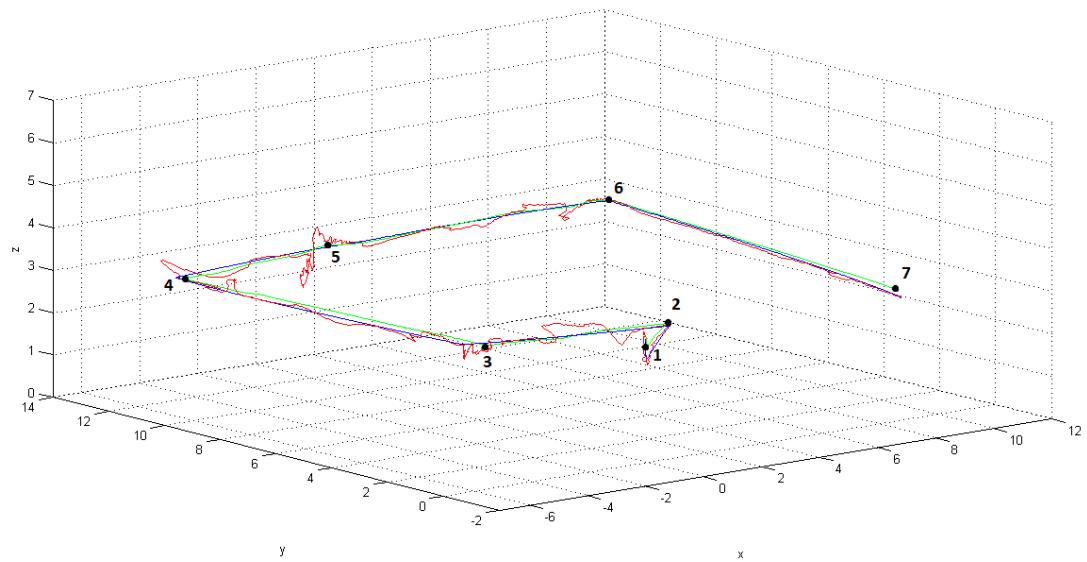


Figura 5.10: Prueba integral de ruta por puntos.

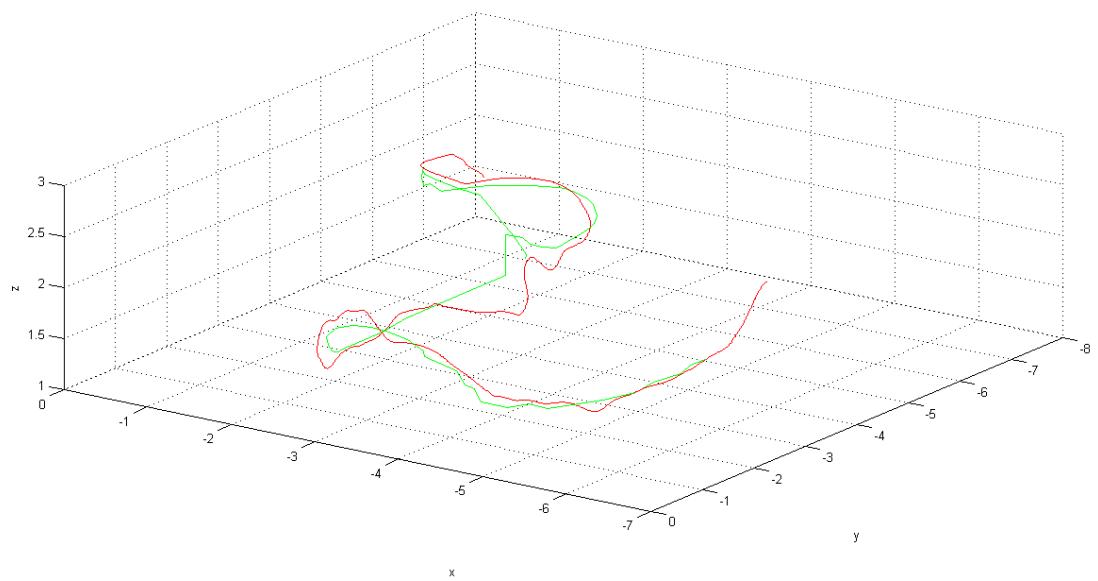


Figura 5.11: Prueba integral de ruta por trayectoria.

Para el experimento de la navegación integral por trayectoria continua decidimos utilizar una ruta compleja con curvas cerradas y saltos de altura, y así comprobar la integridad y el error de la aplicación para cualquier situación puesto que ése es el escenario de navegación más complejo. En la imagen 5.11 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 analizamos con más detalle los datos de error de distancia en la Figura 5.12. De este análisis extraemos que el error medio de distancia en esa ruta compleja fue de 0.128m con un error máximo de 0.563m. Este alto error máximo, que es puntual, se debe a que el reconocimiento de las balizas no se puede realizar a velocidades elevadas, el tiempo de ruta fue de 109.5 segundos.

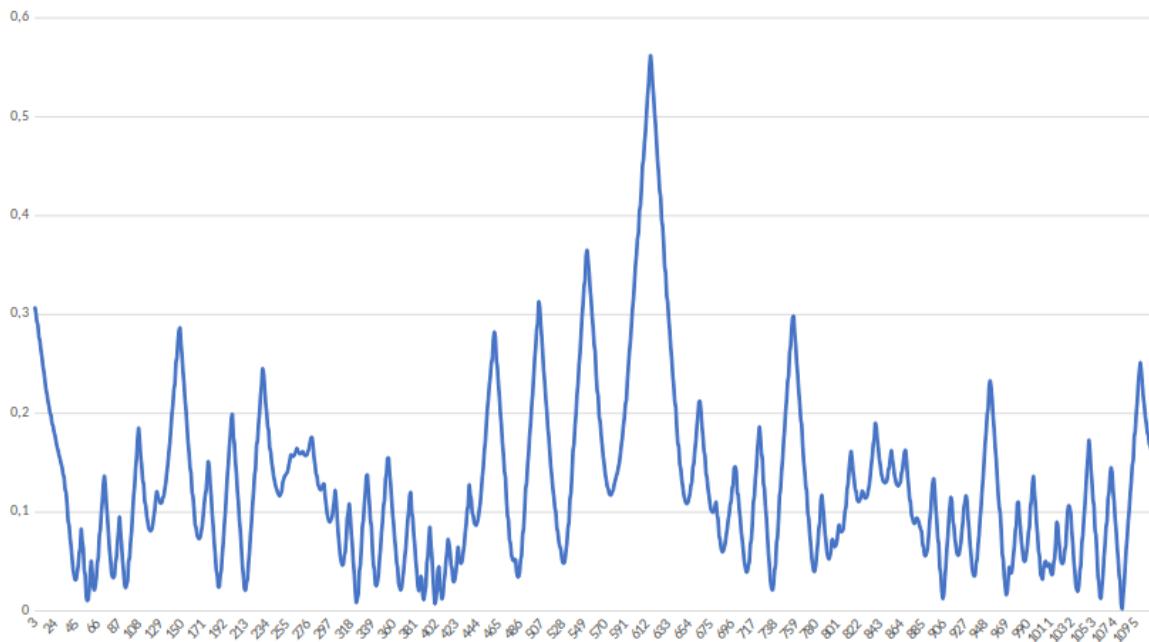


Figura 5.12: Error de distancia a la ruta deseada en la prueba integral del sistema

Este error entre la trayectoria ideal y la trayectoria conseguida incluye y acumula dos fuentes de errores independientes: los errores debidos a una mala autolocalización y los errores debidos a un mal pilotaje de los motores del drones. Es interesante destacar que este error medido experimentalmente es similar al que se

obtenía cuando se alimenta al pilotaje con las posición 3D perfecta del drone (figura 5.11).

Por último, anotar que el piloto desarrollado 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 se ha enriquecido el controlador con dos funcionalidades que lo preparan para el mundo real. Primero, 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. Segundo, una nueva función basada en el nuevo **Slam VisualMarkers** de ROS utilizando el número de balizas observadas en imagen. En el momento que está 2 segundos sin ver ninguna baliza AprilTag el drone interrumpe su intento de seguir la ruta, ya que no está recibiendo una localización real, y gira en círculos cada vez mayores en torno al último punto para así encontrar la baliza anterior y poder volver a localizarse. Si consigue observarla continuar siguiendo la ruta, si no, aterrizará para evitar que tenga 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 partan de este trabajo. Como visión global podemos decir que el trabajo ha sido satisfactorio, ya que se ha conseguido un algoritmo que permite al drone despegar, realizar una ruta y aterrizar, todo ello de manera completamente autónoma, como se deseaba en un primer momento.

6.1. Conclusiones

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

1. **Adaptación e integración del componente Slam-Visualmarkers:** hemos ajustado el componente y lo hemos introducido por pasos para finalmente conseguir su correcto funcionamiento, la nueva aplicación de JdeRobot está implementada en ROS y fruto de este TFG se ha añadido al componente oficial la interfaz del numero de balizas observadas, que permite medir indirectamente la incertidumbre de la estimación de posición.

2. **Recodificación e integración del despegue y aterrizaje visual:** El algoritmo permite que el drone despegue de forma controlada estabilizándose sobre la baliza arlequinada, evitando así que se desvíe por factores externos y poder iniciar la ruta deseada con la certeza de que el drone está completamente nivelado. También mencionar que el aterrizaje se ha simplificado para eliminar los temporizadores y la máquina virtual de estados, introduciéndolo correctamente en el nuevo algoritmo.
3. **Desarrollo de dos módulos 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 de paso, en el cual no se controla el giro, como el pilotaje por trayectorias continuas que se adapta mejor trayectorias complejas y permite un control más fino de la navegación autónoma, 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.
4. **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 pasar de un estado a otro mediante transiciones controladas. Por tanto hemos conseguido integrar navegación autónoma, despegue y aterrizaje.
5. **Validación experimental en entorno simulado Gazebo:** Todos los resultados obtenidos se han extraído a partir de las pruebas realizadas dentro de los mundos creados en este entorno de simulación tridimensional, tanto con experimentos con pruebas unitarias, como con pruebas integrales. Creando así por tanto ejemplos visuales de la robustez del algoritmo completo, con la utilización de todos los módulos para dotar al sistema de autonomía completa. Mejorando el pilotaje, ligeramente en precisión espacial pero notablemente en la realización de rutas y en el tiempo empleado para ello. Con todo 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.

Al cumplir el objetivo principal de la aplicación y todos los subobjetivos podemos decir que se ha conseguido mejorar lo existente anteriormente en JdeRobot sobre navegación autónoma de drones. 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.

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. Todas las pruebas y los avances que se han ido desarrollando y validando se pueden ver en la wiki oficial del proyecto ¹ y el código está accesible en github ²

6.2. Trabajos futuros

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. Este proyecto ha supuesto un aporte, un paso adelante dentro de JdeRobot pero abre a su vez nuevas posibilidades en esta misma línea. A continuación, se detallan algunas de estas líneas de mejora, que han surgido al investigar las herramientas, las nuevas aplicaciones y al desarrollar nuestro propio algoritmo.

¹<http://jderobot.org/Jsaizc-tfg>

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

- Lo primero sería probar el algoritmo en situaciones reales. Aunque hemos comprobado la robustez de algoritmo en entornos simulados y hemos incluido algunas funcionalidades para enfrentarse a entornos reales, 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 linea de investigación es 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, sería interesante extender nuestro 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.

BIBLIOGRAFÍA

- [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>