Trabajo Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Aerial co-workers: a task planning approach for multi-drone teams supporting inspection operations

Autor: Álvaro Calvo Matos

Tutor: Jesús Capitán Fernandez

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Aerial co-workers: a task planning approach for multi-drone teams supporting inspection operations

Autor:
Álvaro Calvo Matos

Tutor:
Jesús Capitán Fernandez
Associate Professor

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Máster: Aerial co-workers: a task planning approach for multi-drone teams supporting inspection operations

Autor: Álvaro Calvo Matos
Tutor: Jesús Capitán Fernandez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi asesor, Jesús, por guiarme en este proyecto, por confiar en mí para formar parte del grupo de investigación al que pertenece, por apoyarme en mi decisión de incorporarme a un programa de doctorado, por buscar siempre lo mejor para nosotros a pesar de sus preferencias y por su amabilidad.

A todos los compañeros de departamento que me han ayudado cada vez que lo he necesitado y a todos los que se han ofrecido como voluntarios para apoyarme. En particular, quiero agradecer a Fran y Arturo todo el tiempo que han dedicado a ayudarme.

A Damián, por acompañarme en los momentos fáciles y difíciles, pero sobre todo, por ser mi amigo, y estar ahí incondicionalmente para lo que necesitara.

A mis compañeros de clase, que a pesar de ser un año difícil con distanciamiento social, han estado tan unidos como siempre.

A todos mis amigos, por ser tan buenos amigos.

A toda mi familia, por su amor y apoyo incondicional, y por su paciencia y comprensión.

Gracias por todo

Álvaro Calvo Matos

Sevilla, 2021

Resumen

Este Trabajo de Fin de Máster ha afrontado problemas que surgen del reciente aumento de las aplicaciones de equipos cooperativos de Unmanned Aerial Vehicle (UAV), los cuales son la autonomía para operar de forma prolongada en el tiempo con robustez ante posibles fallos, y la dificultad de aportar al equipo capacidades cognitivas para poder operar en entornos dinámicos con humanos.

Muchas de estas aplicaciones están siendo ejecutadas actualmente por humanos, haciendo las actividaded mucho más costosas, lentas, e incluso en algunos casos, peligrosas. Es por eso que actualmente existe un gran interés y se están destinando muchos esfuerzos para desarrollar soluciones para los problemas planteados.

El objetivo del trabajo en este TFM es desarrollar técnicas cognitvas de planificación para coordinar flotas de UAVs que asistan a operarios humanos en tareas de inspección y mantenimiento en líneas eléctricas de alta tensión. Estas técnicas deben además extender la autonomía del sistema, garantizar que se cumplan los requisitos de seguridad entre UAVs y trabajadores humanos, y asegurar el éxito de la misión.

Se ha propuesto una arquitectura de software basada en un planificador central y un gestor de comportamientos distribuido. Para llevar a cabo la planificación se han definido costes para las distintas tareas existentes. De esta forma, se asignan a los distintos UAVs de manera eficiente, teniendo en cuenta sus restricciones de batería. Por el otro lado, para controlar el comportamiento de los UAVs y asegurar la seguridad de los equipos aéreos, se han implementado diferentes árboles de comportamiento.

Como resultado, se ha conseguido desarrollar una arquitectura de software capaz de realizar la planificación de las misiones de forma dinámica asegurando mientras tanto la seguridad de los equipos involucrados. Esto constituye una buena base que se puede adaptar fácilmente y a partir de la cual se pueden desarrollar futuros planificadores más complejos. Comparado con la forma típica de implementar gestores de comportamiento, ivolucrando complejas máquinas de estados finitas difíciles de leer, reutilizar y ampliar, el uso de árboles de comportamiento supone una gran mejora y permitirá la creación de comportamientos cada vez más complejos.

Índice

1 Introducción

El uso de los vehículos aéreos no tripulados (UAV) ha crecido considerablemente en los últimos años para numerosas aplicaciones civiles, como la supervisión en tiempo real, la búsqueda y el rescate, la provisión de cobertura inalámbrica, la seguridad y la vigilancia, la agricultura de precisión, la entrega de paquetes y la inspección de infraestructuras [1]. Con el rápido desarrollo de la tecnología en este ámbito, y las demostraciones de lo que pueden hacer los UAV, cada vez se hacen más esfuerzos para llevar esta tecnología a otras aplicaciones. Con el aumento previsto de las aplicaciones de esta tecnología, surgen nuevos problemas y desafíos, como la autonomía, la seguridad, la evitación de obstáculos y la coordinación de equipos multi-UAV. El desarrollo de la tecnología para resolver estos problemas supone un gran esfuerzo, pero como los UAV han demostrado ser fundamentales en situaciones en las que los humanos corren un gran riesgo o son muy ineficaces, y han demostrado su capacidad para evolucionar y desarrollar aún más su potencial a corto plazo, las empresas están invirtiendo en el desarrollo de todo tipo de soluciones basadas en los UAV.

1.1 Motivación

Con el aumento de la demanda mundial de electricidad, ha surgido el reto para las empresas de suministro eléctrico que consiste en mantener y reparar las redes eléctricas de forma que se minimice la frecuencia de los cortes. Según [2], una de las principales causas de los cortes de electricidad son los daños en las líneas de transmisión debidos al mal tiempo o a campañas de inspección ineficaces.

La estrategia que suelen utilizar las compañías eléctricas para reducir los cortes de energía es programar operaciones periódicas de mantenimiento en las líneas activas. Este es el método más adecuado si se quiere asegurar el correcto funcionamiento del sistema y cuando la sustitución de un circuito es inaceptable [**?**]. Estas misiones de mantenimiento son realizadas por tripulaciones experimentadas a bordo de helicópteros y equipadas con trajes de seguridad y arneses, entre otras cosas, que evitan que los operarios reciban una descarga eléctrica (véase la figura **??**). El problema de esta solución es que estas actividades son peligrosas para los operarios, ya que trabajan a gran altura y en líneas electrificadas, consumen mucho tiempo, son muy caras (1.500 dólares por hora) y están sujetas a errores humanos.

Estas son las razones por las que las empresas de distribución tienen la necesidad de desarrollar métodos de mantenimiento más eficientes y seguros. Se han propuesto múltiples soluciones para automatizar esta tarea [**?**], pero la mejor parece ser el uso de UAVs, debido a su flexibilidad y capacidad para inspeccionar a diferentes niveles [**?**]. Para ello, todavía hay que superar algunas barreras importantes, como la limitada autonomía de estos aparatos, las fuertes interferencias electromagnéticas a las que estarían sometidos por estar cerca de las líneas eléctricas, y la capacidad de detectar y evitar los obstáculos de distinta naturaleza que se podrían encontrar en este tipo de

Figura 1.1 Operadores bajando del helicóptero durante una misión de mantenimiento.

entornos [**?**]. Dotar a los UAVs de la capacidad cognitiva necesaria para operar de forma autónoma en entornos tan dinámicos y con presencia humana, y dotarles de un método de planificación rápida en línea [3], es clave para hacer frente a estas complejidades y cumplir con seguridad y éxito la misión asignada con las flotas de UAV.

Una arquitectura de software versátil y fiable es esencial para integrar e interconectar todos los componentes heterogéneos que componen estos sistemas cognitivos multi-UAV. En [4], como parte del proyecto europeo AERIAL-CORE[1], se presenta una arquitectura de software multicapa para llevar a cabo este tipo de misiones de forma cooperativa entre operadores humanos y una flota de quadrotors. Uno de los componentes de software implicados es un planificador de tareas de alto nivel. Su función es coordinar toda la flota de UAVs para generar comportamientos de alto nivel con el fin de completar de forma eficiente, segura y exitosa la misión de mantenimiento o inspección. Este tipo de trabajos tienen la característica de ser dinámicos, ya que no es posible conocer de antemano cuál será el resultado de la inspección como tal para planificarla fuera de línea, sino que, a medida que se desarrolla la misión, surgirán nuevas tareas que la flota deberá atender. Por lo tanto, el planificador de tareas debe ser capaz de reaccionar ante eventos inesperados (nuevas tareas, fallo de un UAV, pérdida de conexión, menos autonomía de la calculada, etc.) y volver a planificar en línea. Así, este planificador de alto nivel será el principal bloque cognitivo del sistema [4].

1.2 Objetivos

El objetivo general de este proyecto es desarrollar un planificador cognitivo de tareas encargado de gobernar el comportamiento de equipos multi-UAV para la inspección y mantenimiento de líneas eléctricas de forma colaborativa con operadores humanos, siendo una de las capas de software que componen la mencionada arquitectura de software [4] desarrollada para el proyecto europeo AERIAL-CORE. La flota de UAVs gobernada actúa como co-trabajadores aéreos y puede realizar diversas tareas, como entregar una herramienta a un operario, inspeccionar regiones de la línea eléctrica o vigilar a un trabajador mientras opera para garantizar su seguridad. El planificador recibe tanto información de alto nivel como de las distintas plataformas que componen la flota, y procesa

[1] Página principal del proyecto europeo AERIAL-CORE: *https://aerial-core.eu/*

toda la información para elaborar un plan que permita gestionar el equipo de UAV o modificarlo como reacción a un imprevisto. Para ello, se definieron los siguientes objetivos:

- Garantizar la utilización de los recursos y la ejecución eficaz de las tareas.
- Cumplir con todos los requisitos de seguridad y garantizar la integridad de las plataformas aéreas y el éxito de la misión.
- Ser capaz de volver a planificar en línea para reaccionar ante acontecimientos imprevistos.
- Implementar la capa de software en Robot Operating System (ROS) y gestionar la comunicación necesaria con el resto de capas y módulos de software que conforman la arquitectura completa.
- Realizar simulaciones de sofware en el bucle (Software In The Loop (SITL)) para demostrar que el algoritmo es capaz de gobernar el comportamiento de la flota de forma eficiente y segura, y que es capaz de reaccionar ante imprevistos de forma dinámica, demostrando capacidades cognitivas.
- Diseñar el planificador de tareas de forma que sea fácil de mantener, modificar o ampliar, buscando que sea modular y reutilizable para que pueda servir de base para la construcción de planificadores para otras aplicaciones.

2 Formulación del problema

Como se mencionó en el capítulo 1, el contexto en torno al cual se desarrolla este planificador cognitivo de tareas es la inspección y el mantenimiento de redes eléctricas. Aunque uno de los objetivos es construir un planificador de tareas cuyas características permitan su fácil reutilización y adaptación para otras aplicaciones, es relevante exponer el problema para el que se está elaborando originalmente.

Como ya se ha mencionado, el proyecto AERIAL-CORE pretende desarrollar diferentes tecnologías para el uso de sistemas multi-UAV en tareas de inspección y mantenimiento en instalaciones eléctricas de alta tensión. En concreto, una de las tecnologías propuestas es el uso de co-trabajadores aéreos (Aerial Co-Worker), es decir, pequeños equipos de UAVs cooperativos para apoyar de forma segura a los trabajadores de mantenimiento mientras trabajan en altura en las líneas eléctricas. Estos sistemas tendrían que interactuar con los humanos (véase la figura 2.1) para inspeccionar ciertas partes que se les indiquen, controlar la seguridad de los trabajadores durante la operación y entregar herramientas u otros equipos ligeros, con el fin de hacer el trabajo más eficiente y seguro. Además, para tener un mayor impacto, el sistema tendría que funcionar durante largos periodos de tiempo, siendo capaz de hacer frente de forma autónoma a determinadas averías o recargas.

Figura 2.1 Equipo multi-UAV apoyando a un trabajador. Fuente: Página web de Aerial-Core.

Se hace referencia a tres tipos de ACWs, cada uno destinado a proporcionar una funcionalidad diferente: *Inspection-ACW*, *Safety-ACW*, y *Physical-ACW*. Los escenarios de los casos de uso pueden resumirse como sigue:

- *Inspección*, donde una flota de ACWs (es decir, *Inspection-ACWs*) lleva a cabo una investigación detallada de los equipos de energía de forma autónoma, ayudando a los trabajadores humanos a adquirir vistas de la torre de energía que no son fácilmente accesibles (véase la figura 2.2);

- *Seguridad*, donde una formación de ACWs (es decir, *Safety-ACWs*) proporciona al equipo de supervisión una visión de los seres humanos que trabajan en la torre de energía con el fin de controlar su estado y garantizar su seguridad (véase la figura 2.3);

- *Interacción física*, donde un ACW (es decir, *Physical-ACW*) interactúa físicamente con el trabajador humano y le proporciona asistencia física, es decir, mientras está en contacto con el humano vuela de forma estable y fiable, y realiza la tarea física requerida (por ejemplo, la entrega de una herramienta) sin resultar perjudicial para el trabajador humano (véase la figura 2.4).

Aunque exista un tipo de ACW específico para cada una de las tareas (inspección, monitorización y entrega de herramientas), esto no significa que un UAV pueda realizar en un momento dado una tarea para la que no es el mejor. Por tanto, será tarea del planificador tener en cuenta qué ACWs son los más adecuados para cada tarea, cuáles no lo son pero podrían realizarla sin problema, y cuáles no tienen capacidad para realizarla. En consecuencia, se multiplica el número de formas en que se puede realizar la planificación de la misión, lo que aumenta considerablemente la dificultad del problema que tiene que resolver el planificador de tareas.

Este problema de planificación de misiones con múltiples UAVs con limitaciones de batería puede plantearse como un problema de optimización, cuya solución indica la forma más eficiente de asignar las diferentes tareas y planificar las recargas. Para reaccionar ante posibles fallos, una de las opciones más extendidas es la de plantear métodos dinámicos que puedan replanificar en tiempo real cuando se produzcan determinados eventos. Aunque existen muchas variantes, la mayoría de las formulaciones para las misiones en las que varios vehículos visitan varias ubicaciones para inspeccionar o realizar entregas dan lugar a problemas de optimización difíciles de resolver (NP-hard) y, por lo tanto, el enfoque más extendido es resolverlos mediante algoritmos heurísticos.

Los métodos de planificación basados en la incertidumbre son apropiados para añadir capacidades cognitivas a un sistema que tenga que interactuar con humanos en entornos dinámicos, ya que permiten optimizar los planes mediante la predicción de las intenciones más probables de los humanos y los resultados de las acciones futuras. El principal problema es su complejidad computacional, ya que el espacio de búsqueda de planes crecería exponencialmente con el número de UAVs y con el horizonte temporal futuro sobre el que se va a realizar la planificación.

En este contexto y con estas ideas en mente, se desarrolló el planificador de tareas cognitivas de esta tesis. Como este planificador cognitivo es un módulo que forma parte de una arquitectura de software más grande para abordar todo el problema, se presenta brevemente la imagen completa de esa arquitectura. Principalmente, qué intercambios de información existen entre las capas superiores e inferiores de la arquitectura de software, las interfaces por las que viaja esta información y las interacciones entre capas para activar los controladores de bajo nivel. En la siguiente sección se presenta esta información explicando individualmente las diferentes tareas contempladas en el proyecto.

Además, se hará un repaso de otras consideraciones importantes que el planificador debe tener en cuenta, como las recargas de batería, las pérdidas de conexión y la reprogramación de tareas; analizando las diferentes situaciones en las que se puede producir cada una de ellas y sus diferentes causas.

2.1 Descripción de las tareas

Como ya se ha dicho, en el proyecto se contemplan tres tipos de tareas diferentes. Estas tareas son solicitadas en todo momento por los trabajadores humanos a través de gestos. Habrá una capa de software de nivel superior que procese la información contenida en los gestos de forma que el planificador reciba una comunicación asíncrona de la capa superior con las especificaciones de una nueva tarea. En este momento, el planificador se encarga de procesar la nueva información junto con la que ya tenía para elaborar y ejecutar un nuevo plan. El mismo planificador también se encarga de llamar a los controladores de bajo nivel cuando es necesario y de garantizar la seguridad del UAVs y el cumplimiento de la misión. Cada tarea se explica en detalle en los siguientes apartados.

2.1.1 Tareas de inspección

Esta tarea puede ser realizada por los tres tipos de ACWs. Es la segunda tarea más prioritaria, siendo la tarea de entrega de herramientas la única que la supera. Consiste en realizar una inspección detallada de las zonas especificadas de las líneas de alta tensión (ver Fig. **??**). La capa inmediatamente superior al planificador de tareas se encarga de pasarle una lista de waypoints (WPs) que definen la tarea de inspección, y el planificador se encarga de decidir cuántos ACWs recluta para ejecutar la tarea y a cuáles de los ACWs disponibles se la asigna. Dividir la lista total de WP a inspeccionar en subconjuntos y asignar cada uno a uno de los ACWs seleccionados para la tarea es el trabajo de un controlador de bajo nivel. Por tanto, una vez ejecutada la planificación, las tareas de este tipo se transmiten a las capas inferiores con la lista total de WPs a inspeccionar y una lista con las identificaciones (identifications (IDs)) de los UAVs seleccionados.

Todas las comunicaciones mencionadas se realizarán de forma asíncrona, ya que la creación de la tarea por parte de los trabajadores, que desencadena toda la secuencia de acciones, se realiza de esta forma.

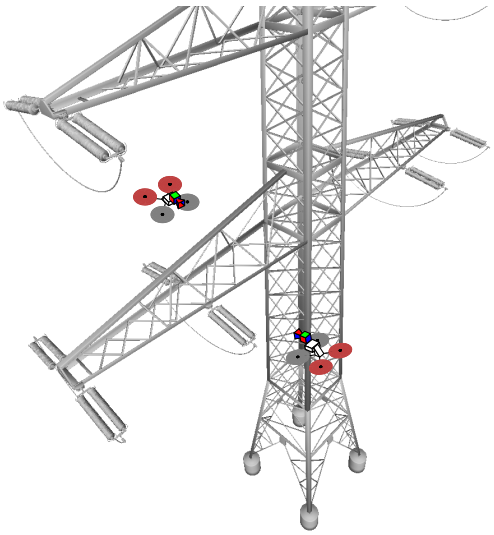

Figura 2.2 *Inspection-ACW* llevando a cabo una tarea de inspección.

2.1.2 Tareas de monitorización

Esta tarea también puede ser ejecutada por los tres tipos de ACWs. Es la tarea de menor prioridad. Monitorizar la seguridad de los trabajadores consiste en proporcionar al equipo de supervisión una visión de las personas que trabajan en la torre de energía para controlar su estado y garantizar su seguridad (ver Fig. 2.3). La capa inmediatamente superior al planificador de tareas comunica en esta ocasión el ID del trabajador a vigilar, el número de UAVs deseado y la distancia que deben

mantener con el trabajador. Es responsabilidad del planificador de la tarea decidir de nuevo a cuál de los ACWs disponibles asignar a esta tarea y la formación que deben mantener durante el vuelo. Una vez realizada la planificación, las tareas de este tipo se transmiten a los estratos inferiores tanto con la información original como con la resultante de la planificación.

Las comunicaciones mencionadas también se realizarán de forma asíncrona por el mismo motivo.

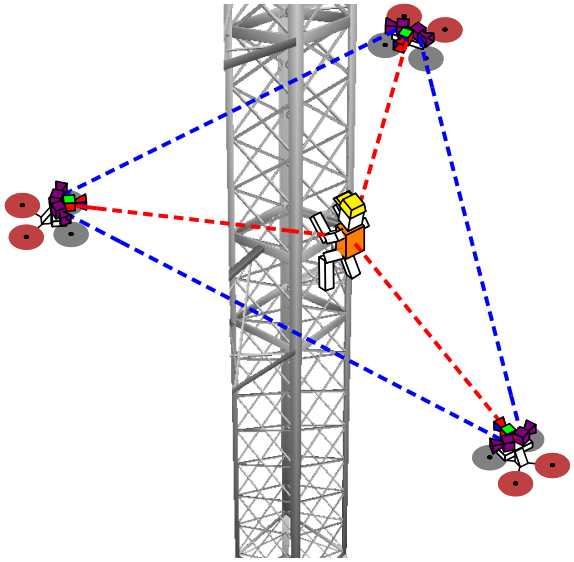

Figura 2.3 *Safety-ACW* llevando a cabo una tarea de monitorización.

2.1.3 Tareas de entrega de herramienta

Esta tarea sólo puede ser realizada por UAVs de tipo *Physical-ACW*, ya que se requiere un hardware especial para realizar la interacción física con los objetos de bajo peso y el humano. Esta es la tarea más prioritaria. La entrega de una herramienta consiste en recoger una herramienta y transportarla hasta el trabajador, con el que se producirá una interacción física a través de la cual se realizará la entrega de la herramienta (véase la Fig. 2.4). Los controladores de bajo nivel tendrán que ser especialmente precisos y cuidadosos para no dañar al trabajador. Esta vez, la capa inmediatamente superior al planificador de tareas comunica el ID del trabajador al que hay que entregar la herramienta y el ID de la herramienta solicitada. De nuevo, la misión del planificador de tareas es decidir a cuál de los ACWs disponibles asignar esta tarea. Una vez realizada la planificación, las tareas de este tipo pasan a las capas de nivel inferior con la misma información que en un principio.

Las mencionadas comunicaciones, una vez más, se realizarán de forma asíncrona.

2.2 Recargas de batería

Dado el problema actual de autonomía con la tecnología de los UAVs, eventualmente cada uno de los ACWs que participan en la misión se quedará sin batería. El momento en que se agotará la batería de cada uno puede estimarse desde la propia planificación de la misión, por lo que el planificador puede tenerlo en cuenta a la hora de distribuir las tareas para que el propio ACW se anticipe a este evento. La recarga no tiene por qué producirse cuando el UAV esté a punto de quedarse sin batería, ni tampoco hasta que ésta llegue al máximo, por lo que ambos serán parámetros a tener en cuenta durante el proceso de planificación y optimización de la misión.

Además, es posible que los cálculos fallen por algún motivo y la batería se agote antes de lo previsto. Por lo tanto, será necesario leer periódicamente el estado de la batería y realizar una recarga de emergencia y una replanificación si es necesario, reaccionando ante fallos inesperados.

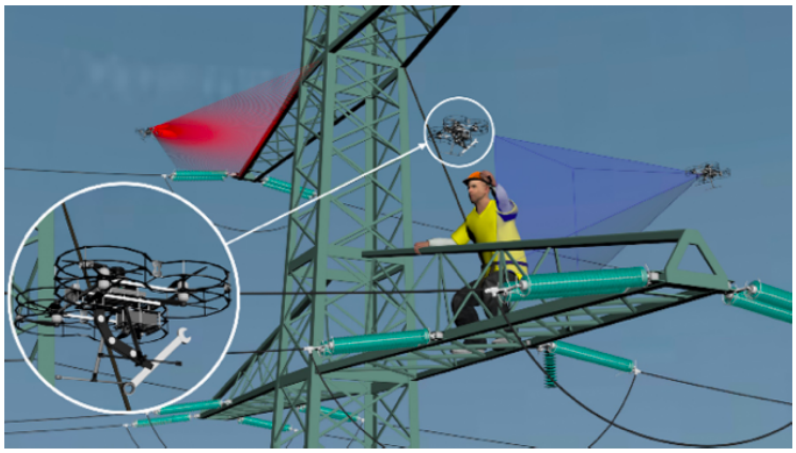

Figura 2.4 *Physical-ACW* llevando a cabo una tarea de entrega de herramienta.

Un escenario posible es que el robot aéreo se quede sin batería durante una pérdida de conexión. Dado que el planificador está centralizado en una estación terrestre, también debería haber un módulo de comprobación de la batería a bordo de cada vehículo aéreo, y un protocolo de emergencia en caso de que esto ocurra.

A falta de especificaciones, se asume que la recarga de la batería no se produce de forma instantánea (no es un cambio de batería), por lo que alcanzar el nivel de batería deseado lleva un cierto tiempo que debe ser considerado en el plan.

Además, el algoritmo de planificación de tareas tiene que ser capaz de manejar sin bloqueos situaciones en las que todos los ACWs están simultáneamente sin batería suficiente y, por tanto, no hay UAVs con los que ejecutar una tarea de forma inmediata.

2.3 Connection losses

Otra consideración importante es la posible pérdida de conexión entre el planificador centralizado, donde se concentra la mayor parte de la capacidad cognitiva, y alguno de los ACWs. Dado que una pérdida de conexión es un evento imprevisto, lo más probable es que el planificador vuelva a calcular la asignación óptima de tareas una vez que se actualice la flota de UAVs, de modo que las tareas previamente asignadas al ACW desconectado se ejecuten en otro. Esta es una situación potencialmente peligrosa, ya que el UAV desconectado podría actuar de forma autónoma según su último plan y provocar un accidente con el resto de agentes que aún están en línea.

Por ello, es importante: (i) implementar un sistema de detección de desconexiones desde ambos lados de la comunicación, y (ii) establecer un protocolo de actuación común para que ambos módulos sepan cómo va a actuar el otro, garantizando así la integridad de todos los vehículos y la seguridad de los trabajadores

2.4 Situaciones de replanificación de tareas

Una vez que la misión está en marcha, cualquier acontecimiento imprevisto tiene el potencial de cambiar por completo cuál es el plan óptimo. Por lo tanto, aunque exista la posibilidad de que el evento no afecte en absoluto a la misión, siempre será necesario ejecutar una replanificación de la misión en caso de que se produzca un imprevisto.

A continuación se enumeran los imprevistos que se han contemplado en este trabajo:

- Llegada de una nueva tarea.

- Modificación de los parámetros de una tarea.
- Conexión de un nuevo ACW.
- Desconexión de un ACW.
- Batería insuficiente imprevista en uno de los ACWs.
- La batería de algún ACW se consume más rápido de lo esperado y por lo tanto no será suficiente para el plan actual.
- Un ACW termina de recargar antes de los esperado.
- Una tarea termina exitosamente.
- Una tarea termina no exitosamente.

Notar que algunos de los acontecimientos considerados no son realmente inesperados. Por ejemplo, la finalización con éxito de una tarea es lo que se desea, por lo que no debería implicar un cambio de planes. Sin embargo, este acontecimiento se incluye en la lista porque es un buen momento para comprobar si existe un plan mejor y modificar el actual si es necesario. Como el planificador persigue el plan óptimo, el resultado de la replanificación mantendrá el plan anterior sin cambios si sigue siendo óptimo.

3 Diseño de la solución propuesta

Este capítulo proporciona los detalles sobre la implementación de la solución al problema del capítulo 2: diagrama de bloques, pseudocódigo y comunicaciones entre módulos. Todo el código está disponible en línea [1], y fue desarrollado bajo el sistema operativo Ubuntu 18.04 y ROS Melodic.

La solución propuesta sigue un enfoque jerárquico, con un planificador de alto nivel encargado de activar diferentes controladores de bajo nivel. El planificador de alto nivel detecta las tareas requeridas por los operadores, y las distribuye desde tierra de forma centralizada entre los ACWs disponibles, planificando las recargas necesarias a lo largo de la misión. Además, este planificador reacciona en tiempo real ante posibles eventos reasignando las tareas. Los planificadores de bajo nivel están a bordo de cada UAV y se encargan de ejecutar los planes de contingencia para estos eventos mientras el planificador central calcula y comunica el nuevo plan. También se encargarán de controlar el movimiento del ACWs para ejecutar las diferentes tareas asignadas por el módulo de nivel superior (por ejemplo, volar hasta un lugar que debe ser inspeccionado o hasta la posición de un operario que espera una herramienta). A partir de ahora, el módulo de bajo nivel a bordo de cada UAV se llamará *Gestor de comportamiento*, y el módulo centralizado en tierra se llamará *Planificador de alto nivel*. Juntos, estos módulos proporcionarán capacidades cognitivas para interactuar con los humanos de forma eficiente en un escenario dinámico.

3.1 Diagrama de bloques

Tal y como se indica en el capítulo 1, el planificador de tareas desarrollado forma parte de una arquitectura de software compuesta por diferentes capas, siendo el bloque cognitivo principal la capa central, el *Planificador de tareas cognitivas de alto nivel*. La figura 3.1 muestra un esquema de la arquitectura software desde la perspectiva del módulo implementado en esta tesis, incluyendo los diferentes bloques y sus interfaces. La parte del diagrama en gris sería la arquitectura software completa, incluyendo desde el módulo de alto nivel encargado de analizar los gestos realizados por los operarios para extraer las tareas de los mismos, hasta los controladores de bajo nivel encargados de ejecutar dichas tareas. La capa de software correspondiente a esta tesis, encargada de la toma de decisiones de alto nivel, está marcada en azul-verde. Está compuesta por el *Planificador de alto nivel*, que está centralizado y se ejecuta en una estación terrestre (en naranja) y el *Gestor de comportamiento*, distribuido a bordo de cada ACW (en color lima).

En el esquema de la arquitectura de software, aunque algunas comunicaciones son bidireccionales, se puede observar que existe un flujo principal de información. Empezando por la información que llega al módulo de *Reconocimiento de Gestos*, ésta se propaga hasta la última capa, donde los *Controladores de nivel inferior* utilizan la información ya procesada para dar órdenes al ACWs. La

[1] Código fuente del Human aware collaboration planner: *https://github.com/grvcTeam/aerialcore_planning*

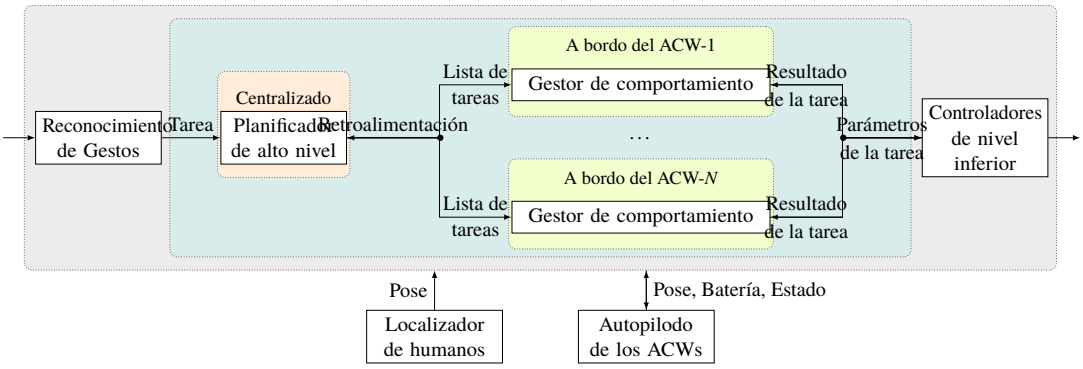

Figura 3.1 Arquitectura de software: bloques e interfaces. Diagrama de bloques desde la perspectiva del planificador de tareas cognitivas de alto nivel.

tabla 3.1 muestra el tipo de datos que cada uno de los módulos de la figura 3.1 recibe como entrada y el tipo de datos que cada uno de ellos envía como salida. Además, la tabla 3.2 explica los detalles de los tipos de datos.

Tabla 3.1 Descripción de las interfaces de datos para cada módulo de software.

Nombre del módulo	Dato de entrada	Dato de salida
Reconocimiento de Gestos	Imágenes	**Task, defined by:** ID de la tarea, Tipo de la tarea, Distancia de monitorización, Número de monitorización, Lista de WPs, ID de la herramienta (algunos parámetros de tarea serán ignorados dependiendo del tipo de tarea)
Planificador de alto nivel	Tarea, Retroalimentación (Resultado de la tarea, Batería suficiente, información del Behaviour Tree (BT)), Pose del humano, Pose del ACWs, Batería y Estado, y Baliza del agente	Lista de tareas añadiendo a cada una sus parámetros extra resultado de la planificación (Formación y/o Lista de IDs de los ACWs) y Baliza del planificador
Gestor de comportamiento	Lista de tareas, Resultado del nivel inferior, Pose del humano, Pose del ACWs, Batería y Estado	Parámetros necesarios para los controladores de bajo nivel (en función del tipo de tarea), Retroalimentación (resultado de la tarea, batería suficiente, información del BT) y Baliza del agente
Controladores de nivel inferior	Parámetros (dependiendo del tipo de la tarea)	Resultado
Localizador de humanos		Pose
Autopiloto de los ACWs	Órdenes de bajo nivel	Pose, Batería y Estado

El primer módulo comprueba constantemente las imágenes captadas por el UAVs en busca de un gesto que indique una nueva tarea o la modificación de una tarea existente. Cuando esto ocurre, envía asíncronamente una tarea, que será recogida por el planificador centralizado. Como se muestra en la tabla 3.1, esta comunicación incluye el ID único que diferencia esta tarea de las demás, el tipo de tarea y los parámetros que la definen.

El *Planificador de alto nivel*, cuando recibe esta información, procede a reevaluar el plan óptimo teniendo en cuenta la tarea recibida, la información que recibe de los *Autopilotos de los ACWs*, y la posición de los operarios, que es publicada periódicamente por el *Localizador de humanos*. Estos datos constituyen la entrada para el *Planificador de alto nivel*, junto con la información procedente

Tabla 3.2 Descripción de los tipos de datos.

Nombre del dato	Tipo del dato	Comentario
ID de la tarea	Cadena	Identificador único de cada tarea
Tipo de tarea	Entero	Indicador del tipo de tarea: m/M, i/I or d/D
ID del humano	Cadena	Identificador único de cada trabajador humano. Se supone que la posición del objetivo humano y otra información necesaria es conocida y accesible a través de su ID.
Distancia de monitorización	Flotante	Distancia desde la que los ACWs vigilan al trabajador durante una tarea de monitorización de seguridad
Número de monitorización	Entero	Número de ACWs que se requieren en la formación para una determinada tarea de monitorización de seguridad
Lista de WPs	Lista de tuplas de 3 flotantes (x, y, and z)	Lista de puntos a inspeccionar
Lista de IDs de ACWs	Lists de Cadenas	Lista de los identificadores únicos de los ACWs que han sido seleccionados para una tarea que requiere múltiples ACWs
Formación	Entero	Indica cuál de los tipos de formaciones predefinidas debe utilizarse para la supervisión (por ejemplo, círculo, triángulo)
ID de herramienta	Cadena	Identificador único de la herramienta a entregar
Pode del ACW	geometry_msgs /PoseStamped	Posición y orientación del ACW
Batería del ACW	sensors_msgs /BatteryState	Porcentaje de batería del ACW
Resultado de la tarea	Cadena, Booleano	El primero es ID único de la tarea y el segundo su resultado una vez terminada
Batería suficiente	Booleano	Resultado de calcular si un ACW tendrá suficiente batería para su tarea actual
Información del BT	Lista de Cadenas	Estado de cada nodo de BT en su última ejecución (Running, IDLE, SUCCESS or FAILURE)
Baliza del Agente	Cadena, Cadena	El primero es el ID único del ACW mientras que el segundo define el tipo de ACW (SafetyACW, InspectACW, o PhysicalACW). Se utiliza como latido y para detectar nuevos ACWs en el Planificador de alto nivel
Baliza del planificador	Tiempo	Mensaje de tipo ROS::Time que contiene el tiempo cuando la baliza fue mandada. Es usado para comprobar el estade de la conexión desde el lado del Agente.
Resultado del nivel inferior	Booleano	Resultado de los controladores de nivel inferior una vez que han terminado después de ser llamados

de cada *Gestor de comportamiento* de los agentes. Su salida es una lista de tareas para cada ACW.

A bordo de cada ACW hay un *Gestor de comportamiento* de agente. Este módulo se encarga de recoger la correspondiente lista de tareas proporcionada por el planificador centralizado. Con esta entrada y la información procedente del *Localizador de humanos* y del *Autopiloto del ACW*, este módulo se encarga de llamar a los *Contoladores de nivel inferior* para llevar a cabo la ejecución del plan asignado. La información emitida por el *Autopiloto del ACW* se utiliza también para comprobar que todo funciona correctamente y para ejecutar los protocolos de seguridad en caso de que sean necesarios. Si esto ocurriera, se emitiría la correspondiente comunicación de vuelta al *Planificador de alto nivel* para calcular un nuevo plan. Estos módulos también reciben el resultado de los *Controladores de nivel inferior* después de llamar a cada uno de ellos, y lo publican de vuelta al *Planificador de alto nivel* como retroalimentación.

Además de estas comunicaciones, los módulos *Planificador de alto nivel* y *Gestor de comportamiento* de agente intercambian periódicamente balizas que sirven para detectar tanto la conexión de

un nuevo ACW como su desconexión en caso de fallo. Además, existe una comunicación asíncrona que se emite a todos los componentes indicando el final de la misión cuando ésta se produce.

Por último, cabe mencionar que el módulo *Reconocimiento de gestos* no tiene una comunicación destinada a modificar los parámetros de una tarea ya contemplada dentro del *Planificador de alto nivel*. Sin embargo, esto es posible porque las tareas tienen un identificador único. Una vez que una tarea ha sido entregada al *Planificador de alto nivel*, para cambiar alguno de sus parámetros, el módulo *Reconocimiento de gestos* sólo tiene que enviar la tarea de nuevo, manteniendo el mismo ID de la tarea y actualizando sólo los parámetros deseados. Así, dentro de la función que se ejecuta cuando se comunica una nueva tarea, se llama a otra función para actualizar los parámetros de las tareas ya registradas.

3.2 Módulo centalizado: Planificador de alto nivel

Como se ha mencionado anteriormente, el *Planificador de alto nivel* es un módulo centralizado que se ejecuta en una estación terrestre y constituye el principal módulo cognitivo de la arquitectura de software. Su objetivo es planificar la misión de forma óptima, es decir distribuir las tareas pendientes entre los ACWs disponibles especificando el orden en el que se van a ejecutar, teniendo en cuenta el tiempo que se tarda en completar cada una, el tipo de cada UAVs, la distancia que tendrá que recorrer cada uno, la batería que tienen disponible, la tarea que estaba ejecutando cada uno, la prioridad de cada tarea, la batería consumida por cada tarea, las recargas que serán necesarias, y cuándo es mejor realizar esas recargas.

El pseudocódigo general de este componente, desde el lanzamiento hasta la terminación, está representado en el código 3.1.

Salida 3.1 General operation of *High-Level Planner*'s code.

```
1. Leer de un ros::param la dirección del archivo de configuración.
2. Leer del archivo de configuración toda la información necesaria.
3. Configurar las comunicaciones ROS (Publishers, Subscribers y
   ActionServers).
4. Configurar la frecuencia de bucle.
5. Bucle "while" principal". Mientras que ros::ok() y no misión
   terminada hacer:
  5.1. Comprobar el tiempo transcurrido de las balizas de los Agentes.
  5.2. Publicar una nueva baliza del Planificador.
  5.3. Comprobar si hay comunicaciones entrantes pendientes (ros::
     spinOnce).
  5.4. Dormir el tiempo restante para enviar la siguiente baliza.
6. 6. Esperar a que todos los UAVs terminen y se desconecten. Mientras
    que haya algún agente conectado hacer:
  6.1. Comprobar el tiempo transcurrido de las balizas de los agentes.
  6.2. Comprobar si hay comunicaciones entrantes pendientes (ros::
     spinOnce).
  6.3. Dormir un rato.
```

Dado que el entorno en el que operan los UAVs es dinámico, este módulo se ha programado de forma que pueda reaccionar ante los imprevistos y recalcular el plan óptimo. Como se puede deducir del código 3.1, todo funciona a través de funciones de respuesta. Cada vez que llega una comunicación desde otro nodo de ROS, se activa una respuesta en este nodo. En ella se analiza la información contenida en el mensaje y se decide si es necesaria una replanificación o no. Las

situaciones en las que se ha considerado necesaria una replanificación se enumeran en la sección 2.4. Las comunicaciones resumidas en las tablas 3.1 y 3.2 y en la figura 3.1 son suficientes para detectar estos imprevistos y poder responder a ellos de la mejor manera posible.

Salida 3.2 Task callback pseudocode.

```
1. If the task already exists:
  1.1. If the new task's type is the same as old one's type:
    1.1.1. Update parameters, perform a task planning and return.
  1.2. Else: Warn operators that a pending task is going to be deleted
       and delete old task.
2. Read the type of task and the parameters that apply to it.
3. Add the new task to the pending task list.
4. Perform a task planning.
```

There is a callback that is executed when the node *Gesture Recognition* sends a task, which in case the given task is correct, always ends up calling the function in charge of calculating the optimal plan (see Code 3.2); the mission over callback, whose only action is to change the value of a variable so that the node exits the main while loop; and finally the agent's beacon callback, which is executed every time a UAV beacon is received and whose pseudocode is in Code 3.3.

Salida 3.3 Agent's beacon callback.

```
1. Read the information contained in the beacon.
2. If it is a connection of a new UAV:
  2.1. Register it in the database.
  2.2. Perform a task planning.
3. Else, if it is the heartbeat of an already known UAV:
  3.1. Reset the timeout timer.
```

The action carried out by the agent's beacon callback varies depending on whether it is the beacon of a new UAV or the heartbeat of a known UAV. For each agent there will be an object in the database that will contain another series of callbacks that will be in charge of receiving the messages coming from the ACWs and respond accordingly.

Salida 3.4 Callback that runs when an *Agent Behaviour Manager* sends battery feedback.

```
1. Update the value of the internal flag associated with the battery.
2. Perform a task planning.
```

The *Agent Behaviour Manager* block only sends communications messages indicating the battery status when it is due to an unplanned event. This event can be either an early battery depletion or a faster than expected recharge. In both cases, the callback function, whose pseudocode is Code 3.4, updates the value of an internal variable used during planning, and recalculates the optimal plan.

The other possible communication coming from a node of type *Agent Behaviour Manager* with the ability to trigger a reaction in the planner is due to the termination of a task. When a task finishes successfully, it is simply removed from the list of pending tasks. In this case, the emphAgent Behaviour Manager block also removes the task from its queue, which is the only case where it does so. In addition, this moment is used to re-evaluate the optimal plan. It is expected that the mission is still within the optimal plan, so in that case the planning result should be the same as the plan that was already being executed. If, instead, conditions have changed since the last planning and there

exists a better plan now, it is at this point that the plan is updated. If the task ends with a failure, the callback action will depend on the causes of the failure (note that the interruption of a task will result in a failure). If the interruption is due to the UAV battery, it may be planned, in which case no action is required, or it may be unexpected, in which case the corresponding actions are taken by the battery callback. Once it has been verified that the task has not finished due to the battery, a check is made to see if the task was at the beginning of the queue. If so, a failure has indeed occurred, so the operators are warned, the task is removed from the list, and a replanning is executed. Otherwise the task in question would have been moved from the top of the queue due to a change of plans and therefore no action would have to be taken either. The pseudocode corresponding to what has just been explained is in Code 3.5.

Salida 3.5 Callback that runs when an *Agent Behaviour Manager* sends a task result.

```
1. Read the information contained in the task result.
2. If the task result is SUCCESS:
  2.1. Delete it from the pending tasks list.
  2.2. Perform a task planning.
3. Else, if the task result is FAILURE:
  3.1. If the task has been halted because of not having battery
     enough:
    3.1.1. Return.
  3.2. Else, if the task is on the front of that ACW's task queue:
    3.2.1. Notify operators that a task has failed and is going to be
       deleted.
    3.2.2. Delete task from the pending tasks list.
    3.2.3. Perform a task planning.
  3.3. Else:
    3.3.1. Return.
```

The other two communications received by the *High-Level Planner* from the ACWs are sensor readings corresponding to the UAVs'position and battery percentage. In both cases the only action of the corresponding callback is to update the information with the new values.

The last function that remains to be explained of those that can potentially request a replanning of the mission is the one in charge of checking the timeout of the agents'beacons. As shown in Code 3.1, this function is not a callback like the previous ones, instead it is executed periodically in the main while loop. Its operation is shown in Code 3.6. Basically, for each agent connected, it checks that the timeout amount of time has not elapsed since its last beacon was received. If a timeout has occurred, that ACW is considered disconnected and is removed from the centralised node data. If, after checking all agents, the number of connected UAVs has decreased, i.e. if any of the previously connected UAVs has disconnected, a mission replanning is executed.

Salida 3.6 Beacons' timeout check function.

```
1. For each agent connected:
  1.1. If the elapsed time since the last beacon is greater than the
     timeout time:
    1.1.1. Add that agent's ID to the list of disconnected agents.
2. While the list of disconnected agents is not empty:
  2.1. Take first ID from the list.
  2.2. Erase from the block's data all information related to that ID.
```

```
3. If any agent has been disconnected:
  3.1. Perform a task planning.
```

The pseudocode that is executed when one of these functions deems it necessary to perform a new task planning is summarised in Code 3.7. It is important to remember that some tasks have a higher priority than others, and this depends only on the type of task. To simplify the process, it has been decided to allocate the tasks in order of arrival, assuming that between two tasks of the same type, the one that arrived first will have priority. When a new task is received, it is stored both in a *std::map* that contains all the pending tasks to facilitate the access to the information, and in a *std::vector* with task types, where the order of arrival is maintained. What this simplification allows is to assign tasks one at a time. By having a prioritised list of tasks and assuming that no task can be assigned before a task with a higher priority, the mission planning problem is reduced to calculating the cost of each task individually for each UAV with the ability to execute it and assign it to the one with the lowest cost. For monitoring-type tasks, the selection of the required number of agents is strictly cost-based. The N agents that cost the least to execute the task are selected. The same is a little more complex for the tasks of type inspect, where the number of agents to select is a parameter to be defined by the planner itself. This value is first set according to the number of points to be inspected. Up to three points, a single ACW is selected; up to six points, two are selected; and from seven points onwards, three agents are selected, this being the maximum number imposed by the low-level controller. Moreover, as the low-level controller in charge of this task works, all the ACWs selected for this task are required to start executing it simultaneously, so a second approximation of this number is made according to the number of idle UAVs. Thus, if they are assigned this as the first task, they will start executing it simultaneously. Academically, this simplification seems to deviate from the optimal solution, but it should be recalled that this work is part of a software architecture that will operate in real situations. In such situations, it is not expected that there will be a large number of UAVs connected simultaneously, nor a long list of pending tasks. In such simplified scenarios, this assumption makes sense without deviating too much from the optimal solution. Finally, the number of agents to be selected will be the smaller of the two above, being equal to one when there is no UAV idle and zero in case there is no ACW with enough battery. In the latter case, the task would be assigned after recharging. Once the number of agents to be selected has been defined, the agents that have the least cost to execute the task are selected from among those that meet the conditions described. Having selected the ACW that will carry out the task, all that remains is to distribute among them the WPs to be inspected. Although the algorithm in charge of performing the optimal distribution is in the low-level controller of this task, as the rest of the modules that make up the software architecture are not yet available, it has been necessary to program a distribution algorithm in order to be able to carry out the experiments. More details on this will be given in Section 3.4.

The cost for each UAV is calculated as the weighted sum of three different types of costs. A first cost assesses the type of ACW and penalises the assignment of tasks to those UAVs designed for another type. It penalises especially the assignment of lower priority tasks to agents designed to perform higher priority tasks. The second cost evaluates the total distance the UAV will have to travel from where it is at the beginning of the task to where it should be by the end of the task. This cost is an approximation of the expected battery consumption, although it does not take into account intermediate travel and hoovering times during the mission. The last cost penalises the interruption of the task that was being executed according to the previous plan and rewards the assignment of the same task. This cost is intended to ensure that a task is preferentially assigned to an idle UAV, to an UAV that is executing a lower priority task, or even to an UAV of a different type, rather than interrupting a task unnecessarily just because that ACW has to travel a shorter distance, for example.

Salida 3.7 Task planning function's pseudocode.

```
1. If there is any agent connected:
  1.1. For each agent connected:
    1.1.1. Make a copy of the current task queue.
    1.1.2. Empty the task queue.
  1.2. For each Tool Delivery task:
    1.2.1. Compute the cost of the task for each PhysicalACW that has
       enough battery.
    1.2.2. Assign the task to the agent for whom the task costs the
       least (from those who has enough battery).
    1.2.3. Add the task to that agent's task queue.
  1.3. For each Inspection task:
    1.3.1. Extract from the task parameters the list of WP to inspect.
    1.3.2. For each ACW (any type) that has enough battery:
      1.3.2.1. Compute the cost of the task for that ACW.
      1.3.2.2. Check if that ACW is still idle.
    1.3.3. Calculate the number of agents to select for the task based
        on the number of WP and the number of idle agents.
    1.3.4. If no agent has enough battery, continue.
    1.3.5. Else, if the number of agents to select is equal to zero,
       assign the task to the agent that costs the least.
    1.3.6. Else, select the calculated number of agents for whom the
       task costs the least.
    1.3.7. Divide the WP to inspect among the selected agents.
    1.3.8. For each selected agent:
      1.3.8.1. Set the remaining task parameters (List of selected
         ACWs' IDs and divided WP list).
      1.3.8.2. Add the task to the agent's task queue.
  1.4. For each Monitoring task:
    1.4.1. Compute the cost of the task for each ACW (any type) that
       has enough battery.
    1.4.2. If the required number of ACWs for the task is zero:
      1.4.2.1. Warn operators that this parameter can not be zero.
      1.4.2.2. Delete task from pending tasks.
    1.4.3. Else, select the requested number of agents for whom the
       task costs the least.
    1.4.4. Set the remaining task parameter (List of selected ACWs'
       IDs)
    1.4.4. Add the task to each selected agent's task queue.
  1.5. For each ACW connected, send the new task queue to its Agent
     Behaviour Manager.
2. Else:
  2.1. Warn operators that no agent is connected.
```

Once the calculation of the mission plan has been completed, the new task queues are sent to the corresponding distributed modules. Each *Agent Behaviour Manager* will react to this communication and will take care of executing the newly assigned plan. In the meantime, the *High-Level Planner* block returns to the main while loop to continue waiting until an event that triggers a replanning occurs again.

3.3 Distributed module: Agent Behaviour Manager

This component is in charge of executing the plan assigned by the *High-Level Planner*, checking the security of the UAVs at all times, detecting unforeseen events and communicating them to the centralised node so that it can make a change of plans if needed. The *Agent Behaviour Manager* will communicate with the low-level controllers, handing over control when necessary to complete the assigned plan.

The general structure of this module is quite similar to that of the central module. The pseudocode is summarised in Code 3.8. Upon initialisation, the *Agent Behaviour Manager* prepares the necessary information to start its operation, configures the necessary communications, declares and initialises the behaviour tree and, once the UAV has finished initialising, starts sending beacons to the central node to notify that it is joining the mission. Once the code finishes initialising and reaches the main while loop, the activity of the *Agent Behaviour Manager* concentrates on the execution of callbacks in response to incoming messages, as in the *High-Level Planner*, and on the execution of the behaviour tree, which directs and supervises the UAV movement.

Salida 3.8 General operation of *Agent Behaviour Manager*.

```
1. Read from a ros::param the beacon's content (ACW's ID and type).
2. Read from a ros::param the address of the configuration file.
3. Read from the configuration file all necessary information.
4. Configure ROS communications (Publishers, Subscribers and
   ActionServers).
5. Set the loop rate.
6. Declare the behaviour tree.
7. Initialise each BT node.
8. Start BT loggers to facilitate debugging and monitoring of the node
   's performance.
9. Wait until the ACW fully initialises.
10. Main "while" loop. While ros::ok() and BT status is running:
  10.1. If a timeout of Planner's beacons has not ocurred:
    10.1.1. Publish a new Agent beacon.
  10.2. Check if battery is enough for the current task.
  10.3. Check for pending incoming communications (ros::spinOnce).
  10.4. Sleep the remaining time to send the next beacon.
```

The BTs are who governs the ACWs to perform each of the assigned tasks. Each BT monitors its ACW's battery and task status and reacts to any possible failure or unexpected event, requesting a new re-planning to the *High-Level Planner* in case of need. A BT can be defined as an improved Finite State Machine (FSM). They are a more advanced mechanism to implement behaviours, especially because of their advantages in terms of scalability, modularity, readability and reusability, facilitating the creation of more complex behaviours with less effort.

Despite this, the process of designing a state machine is quite different from the process of designing a behaviour tree. Designing behaviour trees without ever having done it before is not a trivial task. Moreover, there will be more than one valid implementation to achieve the same behaviour, which makes it more complicated to design this type of solution when you do not still have enough intuition to know which one is better. Taking advantage of the fact that the use of BTs is widespread in the videogame industry, information about them was gathered and studied to try to develop enough knowledge and intuition to design from scratch a BT that meets the needs of the mission. For that, previous examples found in [5, 6, 7] were very useful.

Before proceeding with the explanation of the designed BT, the types of nodes that can be found in the selected C++ library (see Figure 3.2) and the functioning of each of them will be briefly discussed.

Figura 3.2 Different types of nodes that can be present in an BT.

Behaviour Trees are made up of *Control* nodes, *Decorator* nodes, and *Leaf* nodes. *Control* nodes could be either *Fallback* nodes, represented with a question mark (see subfigure 3.2a), which try success calling one by one each of their children; or *Sequence* nodes, represented with an arrow (see subfigure 3.2b), which call their children in order if the previous one has succeeded. On the one hand, *Fallback* nodes return *SUCCESS* if one of its children does it, *FAILURE* if none of them success, and *RUNNING* if one of its children returns *RUNNING*. On the other hand, *Sequence* nodes return *SUCCESS* when all children have been called in order and have returned *SUCCESS*. If any of them returns *FAILURE*, the sequence is broken and the *Sequence* node returns *FAILURE* too. When a child returns *RUNNING*, the *Sequence* node does it too. *Control* nodes are represented in a black rectangular box when they are the standard ones (see subfigure 3.2d), but they could also be *Reactive* control nodes, represented by a magenta box (see subfigure 3.2c), which means that its already called children will be called again in the next iteration. This is very useful for generating behaviours where an action is constantly reattempted, or where it is necessary to check that the required conditions are still met. A *Child* node could be another *Control* node, a *Decorator* node, a *Leaf* node or a whole sub-tree. A *Decorator* node, represented in an orange box (see subfigure 3.2e), can only have one child (of any type) and its function is programmable (e.g., modifying its child result or retrying calling its child a number of times). *Leaf* nodes, represented in blue, could be *Condition* nodes, represented in a blue elliptical shaped box (see subfigure 3.2g), that check a condition and return either *SUCCESS* or *FAILURE*; or *Action* nodes, represented in a blue rectangular box (see subfigure 3.2f), that execute code that takes longer and therefore these nodes could also return *RUNNING*.

3.3.1 Main tree

In general, the design of both the behaviour tree and the *Agent Behaviour Manager* has been made with the aim of concentrating as less intelligence as possible, to ensure the success of the mission and the safety of the UAVs and the workers. That is why the only task of the callback functions that are executed when different messages come in is to update the value of the corresponding internal variables.

However, not all intelligence and decision-making can be placed in the ground station node. There has to be some decision-making capability on board UAVs in case the connection to the central node is lost. That is why there is a predefined protocol to act when this happens or when the battery runs out of power earlier than expected. These two factors are periodically checked in the main while loop (see Code 3.8). In the case of the battery, if the function in charge determines that there is not enough battery to complete the current task, what happens is that the task queue is emptied and the value of the internal flag associated with the battery is updated. In addition, the event is communicated to the task planner in case the connection is still alive to generate a new plan. Similarly, if a connection loss is detected, the task queue is emptied and the corresponding flag is updated. In this case the *High-Level Planner* node will execute a replanning when it also detects the connection loss.

The behaviour tree is designed in such a way that, when the task queue is emptied and the respective flags are updated, the corresponding UAV goes to the battery charging station, which is the established emergency protocol. In order to justify this decision, each of the cases will be analysed separately below.

In case the connection between the two nodes is still active but there is not enough battery, the aim of the contingency plan is to eliminate risks to the UAV while the *High-Level Planner* generates new instructions. In addition, the new plan is likely to involve recharging the battery as a first step. Besides, there is a possibility that the connection may be lost at this point.

The danger of connection loss is that the *High-Level Planner* will replan the mission without the disconnected ACW, so that the tasks previously assigned to it will now be executed by others. If in this scenario the disconnected ACW continues with the last assigned plan, collisions could occur. The emergency protocol ensures that the disconnected UAV does not interfere with the new plans. In addition, the time until the connection is re-established is used by recharging the battery, which is positive for the mission.

BTs operate recursively. All nodes, regardless of their type, have a function that executes their content, the *tick* function. When the root node is *ticked* from the main while loop, it propagates the *tick* among its children following the operation rules described in Section 3.3 until eventually a *leaf* node returns one of three possible responses (*SUCCESS*, *RUNNING* or *FAILURE*), which will be propagated back, to obtain a final result that the root node will return.

Typically, a BT is executed to achieve a goal, and therefore the executor keeps on doing *tick* to the tree root until the response is either *SUCCESS* or *FAILURE*. As the function of this BT is to control during the whole mission the movement of a UAV, it is of interest that the result of the root is *RUNNING* until the mission ends. This is why a *Decorator* node that always returns *RUNNING* regardless of the result of its child node has been defined. The BT implemented as a solution for the described problem is further divided into several BTs, taking advantage of the modularity offered by this approach. The main tree is represented in Figure 3.3, being the node named as *Main Tree* the root of the complete tree.

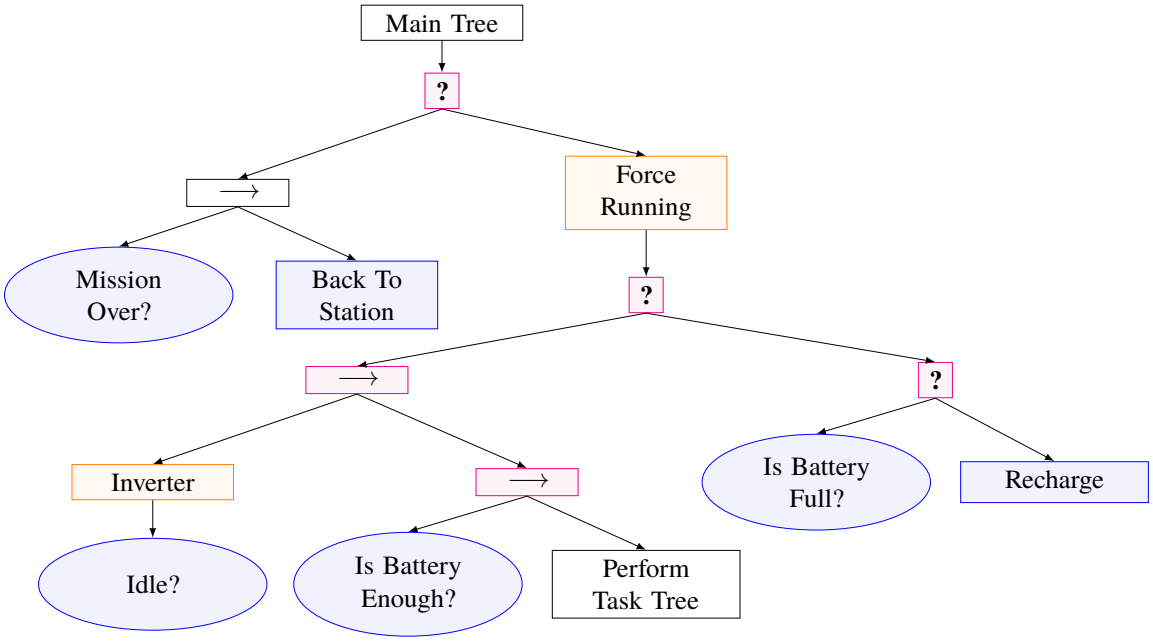

Figura 3.3 Behaviour Tree: Main tree.

This BT checks whether the mission is over (reminder: a mission would represent the working session, not a single task, i.e., whether the ACW is ready to be turned off) and if so directs the ACW

back to the base station. If not, the main *Fallback* ticks to the right branch of the tree, entering the *Recursive Fallback* node that controls the mission. This branch checks if any tasks are assigned. If it turns out that the ACW is idle and the battery is not at hundred percent, the ACW is guided to a recharging station [2]. If a task is assigned and the corresponding flag indicates that the battery is enough, it enters directly into the *Perform Task Tree* (sub-tree represented in Figure 3.4).

This *Recursive Fallback* is where is coded the behaviour that prepares the BT to be safe against a loss of connection or an unexpected battery event. As both unexpected events are managed flushing the task queue, the ACW reacts recharging, while giving the High-Level Planner control to decide when it is the best time to stop recharging (the High-Level Planner just needs to assign tasks again so that the ACW starts working back). Note that, thanks to the presence of *Recursive Control* nodes, the *Leaf Condition* nodes are constantly being re-evaluated. Thus, in case of any unforeseen event or change of plan, the BT will react by instantly stopping the executing branch and switching to the appropriate branch.

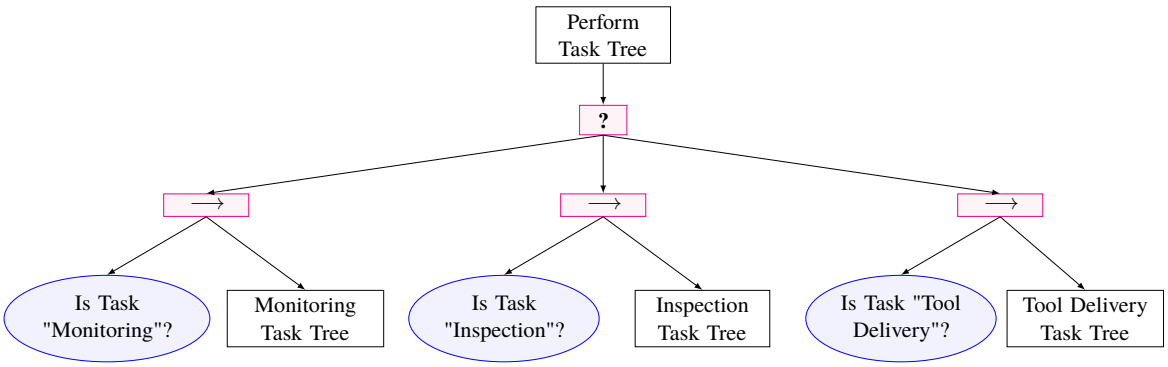

Figura 3.4 Behaviour Tree: Perform Task Tree.

The *Perform Task Tree* checks which is the first task in the queue, which is the task that should be executed at that moment. This tree does not require much more explanation, it simply connects the *Main Tree* with the corresponding *Task sub-tree*. At this point, instead of sub-trees, it would be possible to directly place *Leaf* nodes that give control to the appropriate low-level controller, starting to execute the task directly. However, it was decided that the full control would not be given to the low-level controllers until the ACW is close enough to the area where the task takes place.

In Figures 3.6, 3.5 and 3.7 are depicted the sub-trees that run Safety, Inspection, and Physical tasks, respectively. They all guide the ACW close enough to where the low-level controllers need to be called (e.g., close to a worker to monitor or a place to inspect) and then, control is given to the corresponding one. These low-level controllers run on board the corresponding ACWs and communicate their results (success or failure) asynchronously back to the *Agent Behaviour Manager*, so that the BT can continue running.

In the following, the functioning of each of these sub-trees is described, as well as the details of each of the tasks that have not yet been explained.

3.3.2 Inspection task tree

This BT is quite simple as the task is just to visit a series of points and stop at each one to take pictures (see Fig. 3.5).

[2] Both Safety, Inspection and Physical-ACW provide an input interface to guide the ACW to the charging station. In other words, among the low-level controller capabilities, there is a "reach this waypoint". The location of the charging stations is known in advance or provided as input by the High-Level Planner/Behaviour Tree.

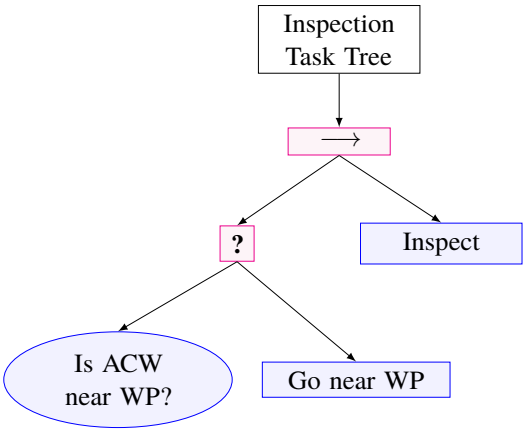

Figura 3.5 Behaviour Tree: sub-tree that controls the inspection tasks.

As mentioned above, control is handed over to the low-level controllers when the ACW is close enough to the area where the task takes place. The *Reactive Sequence Control* node behind the root of this sub-tree is in charge of checking this condition at any moment, stopping the low-level controller if it is no longer fulfilled.

All the information that the BT needs to operate is known in advance by the *Agent Behaviour Manager*, either because it has received it through one of the interfaces represented in Figure 3.1, or because it has calculated it itself in one of the functions that are executed in the main while loop. In particular, to check if the ACW is close to the points to be inspected, the information received from the *ACW's autopilot* is used against the points in the list of WPs to be inspected. If necessary, the *Action* node *Go near WP* is executed, which connects to the low-level controller in charge and gives as target location the nearest WP in the list. Once there, the low-level controller that executes this task is called.

3.3.3 Monitoring task tree

As can it be seen in Figure 3.6, this BT follows the same structure as the previous one. The only differences are the *Leaf* nodes.

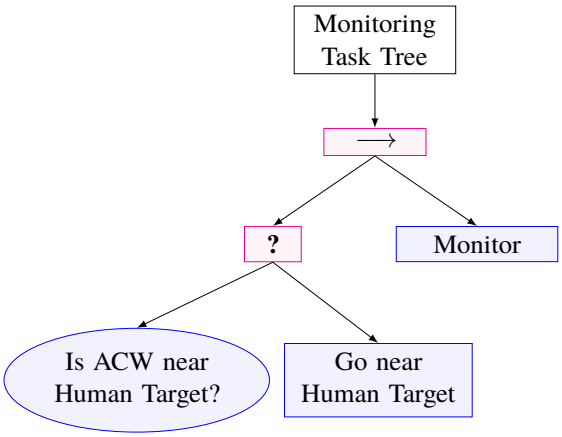

Figura 3.6 Behaviour Tree: sub-tree that controls the safety monitoring tasks.

In this case, the position of the operator to be monitored is known from communications with the *Human Tracker* node (see Figure 3.1). If the ACW is not close enough, the BT will call the low-level controller giving it this time the position of the operator as a target. Once the UAV is in

a position close to the worker, control is passed to the low-level controller in charge of this task. The information needed by this node is the *Monitoring Number*, the *Monitoring Distance*, the *list of ACWs'IDs*, and the formation to be maintained during the flight, which will be chosen by the *High-Level Planner* from a set of predefined formations based on the monitoring number (see Table 3.2). Finally, it is worth mentioning that an ACW could be added or removed from the formation at any time by updating the parameters of the task, or even due to availability.

3.3.4 Tool delivery task tree

This tree is a bit more complex, as the task involves several steps (see Fig. 3.7). First, it picks up the requested tool in case the ACW does not have it yet. This part involves travelling to the station to pick up the tool in case the UAV is not already there. The second part of the task is the process of approaching the operator. After all this, the low-level controller can be activated to carry out the delivery of the tool through a physical interaction between the operator and the ACW.

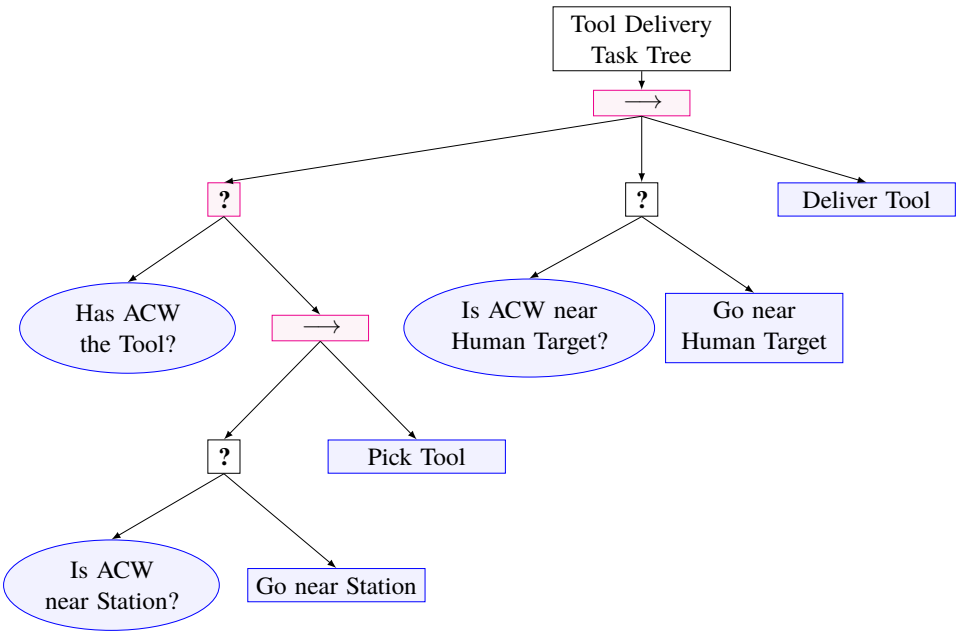

Figura 3.7 Behaviour Tree: sub-tree that controls the tool delivery tasks.

Again, the location of the operator and the position of the ACW is provided by the *ACW's autopilot* and *Human Tracker* blocks. The position of the fixed elements, in this case, the station where the tools are placed, is available among the information read by the *Agent Behaviour Manager* node from the configuration file during its initialisation.

In this case, the low-level controller that delivers the tool will be in charge of requesting the operator's permission to approach. In the meantime, it will have to wait. When it gets the permission, it is time to make the delivery. If too much time elapses without the operator giving the order, the task will be aborted and the tool will be returned to the station.

3.4 High- and low-level blocks faking

The work in this thesis consisted of programming one of the software layers that make up a software architecture. As there are still parts of the software architecture that are not yet available for integration, temporary solutions have had to be programmed to simulate their action during testing. This section will discuss those parts of the code whose mission is to fool both *High-Level Planner*

and *Agent Behaviour Manager* blocks into believing that they are communicating with the real blocks, or to provide functions necessary for the execution to progress.

Low-level controllers have been faked in different ways. For BT's *Action* nodes of the *"Go To"* type, the *GoToWaypoint* ROS service available in the UAV Abstraction Layer (UAL) tool is called instead, which leads the UAV to the entered coordinates (although it does not take obstacles into account). For more complex *Action* nodes such as *Inspect*, *Monitor*, *Deliver Tool* or *Pick Tool*, a function is simply called which sleeps the *Action* node for a while simulating that the low-level controller has been called and is waiting for a response. For these *Action* nodes, the response of the *tick* function will be *RUNNING* while sleeping, and either *FAILURE* or *SUCCESS* depending on whether their execution is halted or not. During sleeping time the rest of the tree continues running. The *Recharge* node also calls some low-level controllers. In this particular case it was decided to ignore the call and simply land the UAV on the charging station.

On the other hand, since UAL does not allow battery control, and the battery level that is available for reading remains static throughout the simulation, it has been necessary to create a block that simulates the evolution of the battery both during flight and during recharging. This block is programmed in such a way that at initialisation it reads the configuration file, thus having the position of the charging stations, and also subscribes to the information published by the *ACW's autopilot* to which it is faking the battery in order to know its position and status. If the UAV is in the air, the battery will be periodically decremented. Otherwise, the battery will remain static unless the UAV is over a charging station, in which case the battery will periodically increment. The battery percentage and charge/discharge rate is externally configurable at any time during the simulation. In addition, for ease of testing, this block allows for different modes of operation that can among other things make the battery static. The false battery level is periodically published in a similar direction to the one used for the real battery.

Finally, as mentioned in the section 3.2, the algorithm in charge of performing the distribution of WPs for an inspection task among the different selected ACWs has been forged. Normally, this algorithm is executed inside the low-level controller itself which is called by the *Inspect Action* node which can be found in the tree of the figure 3.5. However, as this *Action* node has been completely forged, a distribution of WPs has been made within the task planning algorithm. It simply assigns, in order, one WP from the list to each selected ACW until the list of WPs is exhausted.

4 Conclusiones and trabajo futuro

4.1 Conclusiones

En este trabajo se ha desarrolado un planificador de tareas con capacidad para realizar la planificación de misiones para equipos multi-UAV. El sistema tiene la capacidad cognitiva suficiente para controlar a múltiples UAVs que funcionen como co-trabajadores en entornos dinámicos de forma segura. Las simulaciones realizadas han demostrado la capacidad que tiene el sistema para detectar situaciones de emergencia y actuar de forma segura ejecutando planes de contingencia de forma autónoma mientras se calcula un nuevo plan a seguir que tenga en cuenta los imprevistos que hayan acontecido. El diseño del sistema dividido en un bloque centralizado en tierra encargado de realizar la planificación óptima de la misión y de bloques distribuidos a bordo de cada uno de los ACWs permite que el sistema presente robustez ante fallos y capacidad cognitiva suficiente para reaccionar ante eventos imprevistos recalculando el plan óptimo. De esta forma, se consigue una ejecución eficiente de las tareas y un mejor aprovechamiento de los recursos, que se traduce en una mayor autonomía conjunta del equipo de UAVs.

El sistema se ha diseñado en ROS y las comunicacioens entre las diferentes capas de software y los diferentes bloques de cada capa se han realizado empleando las herramientas que este ofrece. Esto facilita la integración del sistema desarrollado en otros proyectos de robótica que necesiten de un planificador de tareas con estas características o similares. El uso de BT para el diseño del UAV behaviour manager presenta grandes ventajas frente a las FSM convencionales. Esta técnica permite generar comportamientos complejos con numerosos estados sin que haya que preocuparse por tener en cuenta cada una de las transiciones entre esos estados como pasa con las FSM, en las que el número de transiciones crece exponencialmente con el número de estados. Las características de la librería empleada para programar esta parte del sistema hacen que sea facil de mantener, de modificar o de amplicar. Además, gracias a su caracter modular, este bloque puede ser reutilizado tanto a partes como en su totalidad en cualquier otro proyecto. El BT diseñado, aunque mejorable, ha demostrado en las simulaciones realizadas que funciona bastante bien, sentando las bases para programar comportamientos más complejos en el futuro y sirviendo de ejemplo para la comunidad de robótica aérea, que podrá emplearlo como punto de partida para otras aplicaciones.

Respecto al bloque que se encarga de la planificación de la misión, el High-Level Planner, decir que hasta el momento ha demostrado tener la capacidad para general planes con sentido en las condiciones en las que se le ha puesto a prueba y que además ha demostrado ser capaz de recalcular dichos planes en línea como reacción a eventos imprevistos de diferente naturaleza. La solición alcanzada es capaz de planificar la misión teniento en cuenta resticciones impuestas como el tipo de cada ACW, la prioridad de cada una de las tareas y el nivel de autonomía de cada uno de los UAVs, siendo capaz de calcular planes para misiones formadas por una cantidad indefinida de tareas y ACWs. Respecto a la optimalidad de los planes generados por este bloque hay que decir que no

se ha implementado ningún algoritmo que realice una búsqueda o aproximación del plan óptimo, sino que se ha diseñado una solución basada en una funcion de costes que se calcula para cada uno de los ACWs con cada una de las tareas. Sin embargo, este trabajo forma parte de un proyecto que tendrá aplicaciones reales en unas condiciones definidas. Por tanto, está justificado alejarse de análisis académicos en los que se busca aproximar el plan óptimo en misiones con un número indeterminado de tareas y de ACWs y analizar en su lugar al planificador en escenarios compuestos por pocas tareas y ACWs. La solución alcanzada, en este contexto, es una aproximación válida hacia un algoritmo de planificación que genere un plan próximo al óptimo.

4.2 Trabajo futuro

Como parte del trabajo futuro se realizará una validación en un entorno real con equivos reales de las técnicas desarrolladas en este trabajo. Además, el sistema desarrollado en este trabajo se empleará como punto de partida de una tesis doctoral en la que se tratará de pulir y mejorar el diseño del Agent Behaviour Manager, así como de desarrollar un algoritmo de planificación que genere una aproximación real al plan óptimo de cada situación. Para ello se introducirán al sistema algoritmos heurísticos aleatorios, así como la capacidad para aprender en tiempo real características como el consumo de la batería de los UAVs, anticipándose de esta forma a eventos imprevistos aplicando planes de contingencia, consiguiendo así una mayor robustez ante fallos y extendiendo la autonomía del sistema aún más.

Una primera mejora para el planificador respecto a la versión actual podría ser la incorporación de tareas de tipo Recarga que, en vez de ser solicitadas por operarios humanos como el resto de las tareas, serían incorporadas por el High-Level Planner a la cola de tareas, separando de esta forma las recargas de emergencia o las recargas que se ejecutan cuando un agente se encuentra ocioso, de las recargas llevadas a cabo como parte del plan. Implementar este cambio en el BT supondría modificar el Perform Task Tree para contemplar esta nueva tarea en el diseño del árbol, cambio que se podría llevar a cabo reutilizando y adaptando ligeramente los árboles empleados para las tareas de Inspection y Safety Monitoring aprovechando la reusabilidad de los BTs.

Además, durante la futura tésis, se pretende investigar el uso de tacnologías de realidad mixta también para aplicaciones de inspección con equipos multi-UAV, combinando vistas tomadas desde distintos puntos para recrear entornos visuales más completos para el operario, y mejorando la interacción hombre-máquina del sistema durante tareas colaborativas.

Índice de Figuras

Índice de Tablas

Índice de Salidas

Bibliografía

[1] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges," *IEEE Access*, vol. 7, pp. 48 572–48 634, 2019.

[2] J.-Y. Park, S.-T. Kim, J.-K. Lee, J.-W. Ham, and K.-Y. Oh, "Method of operating a gis-based autopilot drone to inspect ultrahigh voltage power lines and its field tests," *Journal of Field Robotics*, vol. 37, no. 3, pp. 345–361, 2020.

[3] R. Pěnička, J. Faigl, and M. Saska, "Physical orienteering problem for unmanned aerial vehicle data collection planning in environments with obstacles," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3005–3012, 2019.

[4] G. Silano, J. Bednar, T. Nascimento, J. Capitan, M. Saska, and A. Ollero, "A multi-layer software architecture for aerial cognitive multi-robot systems in power line inspection tasks," in *2021 International Conference on Unmanned Aircraft Systems*, 2021, pp. 1624–1629.

[5] D. Faconti, "Behaviortree.cpp," https://www.behaviortree.dev/, 27-May-2019, [Online; accessed 10-November-2020].

[6] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[7] C. Simpson, "Gamasutra. behavior trees for ai: How they work," https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php, 17-July-2014, [Online; accessed 15-November-2020].

[8] H. Baik and J. Valenzuela, "Unmanned aircraft system path planning for visually inspecting electric transmission towers," *Journal of Intelligent & Robotic Systems*, vol. 95, no. 3, pp. 1097–1111, 2019.

[9] C. Martinez, C. Sampedro, A. Chauhan, J. F. Collumeau, and P. Campoy, "The power line inspection software (polis): A versatile system for automating power line inspection," *Engineering applications of artificial intelligence*, vol. 71, pp. 293–314, 2018.

[10] K. P. Valavanis and G. J. Vachtsevanos, *Handbook of unmanned aerial vehicles*. Springer, 2015, vol. 2077.

[11] A. Roshanbin, H. Altartouri, M. Karásek, and A. Preumont, "Colibri: A hovering flapping twin-wing robot," *International Journal of Micro Air Vehicles*, vol. 9, no. 4, pp. 270–282, 2017.

[12] A. G. Eguíluz, J. Rodríguez-Gómez, J. Paneque, P. Grau, J. M. de Dios, and A. Ollero, "Towards flapping wing robot visual perception: Opportunities and challenges," in *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*. IEEE, 2019, pp. 335–343.

[13] A. Ollero and L. Merino, "Control and perception techniques for aerial robotics," *Annual reviews in Control*, vol. 28, no. 2, pp. 167–178, 2004.

[14] S. S. Bueno, J. R. Azinheira, J. Ramos, E. Paiva, P. Rives, A. Elfes, J. R. Carvalho, and G. F. Silveira, "Project aurora: Towards an autonomous robotic airship," in *Workshop on Aerial Robotics, IEEE International Conference on Intelligent Robots and Systems*, 2002, pp. 43–54.

[15] Wikipedia contributors, "General atomics mq-1 predator — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=General_Atomics_MQ-1_Predator&oldid=1041584540, 2021, [Online; accessed 27-September-2021].

[16] A. Simpson, O. Rawashdeh, S. Smith, J. Jacob, W. Smith, and J. Lumpp, "Big blue: high-altitude uav demonstrator of mars airplane technology," in *2005 IEEE Aerospace Conference*, 2005, pp. 4461–4471.

[17] E. Capello, A. Scola, G. Guglieri, and F. Quagliotti, "Mini quadrotor uav: design and experiment," *Journal of Aerospace Engineering*, vol. 25, no. 4, pp. 559–573, 2012.

[18] S. N. Ghazbi, Y. Aghli, M. Alimohammadi, and A. A. Akbari, "Quadrotors unmanned aerial vehicles: A review," *International journal on smart sensing and Intelligent Systems*, vol. 9, no. 1, 2016.

[19] I. Kroo, F. Prinz, M. Shantz, P. Kunz, G. Fay, S. Cheng, T. Fabian, and C. Partridge, "The mesicopter: A miniature rotorcraft concept phase ii interim report," *Stanford university*, 2000.

[20] P. Pounds, R. Mahony, P. Hynes, and J. M. Roberts, "Design of a four-rotor aerial robot," in *The Australian Conference on Robotics and Automation (ACRA 2002)*, 2002, pp. 145–150.

[21] R. Rashad, J. Goerres, R. Aarts, J. B. Engelen, and S. Stramigioli, "Fully actuated multirotor uavs: A literature review," *IEEE Robotics & Automation Magazine*, vol. 27, no. 3, pp. 97–107, 2020.

[22] R. Zufferey, J. Tormo-Barbero, M. M. Guzmán, F. J. Maldonado, E. Sanchez-Laulhe, P. Grau, M. Pérez, J. Á. Acosta, and A. Ollero, "Design of the high-payload flapping wing robot e-flap," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3097–3104, 2021.

[23] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue," *IEEE robotics & automation magazine*, vol. 19, no. 3, pp. 46–56, 2012.

[24] I. Maza, A. Ollero, E. Casado, and D. Scarlatti, "Classification of multi-uav architectures," *Handbook of unmanned aerial vehicles*, pp. 953–975, 2014.

[25] H. Ren, Y. Zhao, W. Xiao, and Z. Hu, "A review of uav monitoring in mining areas: Current status and future perspectives," *International Journal of Coal Science & Technology*, vol. 6, no. 3, pp. 320–333, 2019.

[26] B. N. Chand, P. Mahalakshmi, and V. Naidu, "Sense and avoid technology in unmanned aerial vehicles: A review," in *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*. IEEE, 2017, pp. 512–517.

[27] H. Aasen, E. Honkavaara, A. Lucieer, and P. J. Zarco-Tejada, "Quantitative remote sensing at ultra-high resolution with uav spectroscopy: A review of sensor technology, measurement procedures, and data correction workflows," *Remote Sensing*, vol. 10, no. 7, p. 1091, 2018.

[28] F. Nex and F. Remondino, "Uav for 3d mapping applications: a review," *Applied geomatics*, vol. 6, no. 1, pp. 1–15, 2014.

[29] P. Radoglou-Grammatikis, P. Sarigiannidis, T. Lagkas, and I. Moscholios, "A compilation of uav applications for precision agriculture," *Computer Networks*, vol. 172, p. 107148, 2020.

[30] C. D. Drummond, M. D. Harley, I. L. Turner, A. N. A Matheen, W. C. Glamore *et al.*, "Uav applications to coastal engineering," in *Australasian Coasts & Ports Conference 2015: 22nd Australasian Coastal and Ocean Engineering Conference and the 15th Australasian Port and Harbour Conference*. Engineers Australia and IPENZ, 2015, p. 267.

[31] J. Martínez-de Dios, L. Merino, A. Ollero, L. M. Ribeiro, and X. Viegas, "Multi-uav experiments: application to forest fires," in *Multiple heterogeneous unmanned aerial vehicles*. Springer, 2007, pp. 207–228.

[32] J. Gu, T. Su, Q. Wang, X. Du, and M. Guizani, "Multiple moving targets surveillance based on a cooperative network for multi-uav," *IEEE Communications Magazine*, vol. 56, no. 4, pp. 82–89, 2018.

[33] R. Shakeri, M. A. Al-Garadi, A. Badawy, A. Mohamed, T. Khattab, A. K. Al-Ali, K. A. Harras, and M. Guizani, "Design challenges of multi-uav systems in cyber-physical applications: A comprehensive survey and future directions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3340–3385, 2019.

[34] J. Scherer, S. Yahyanejad, S. Hayat, E. Yanmaz, T. Andre, A. Khan, V. Vukadinovic, C. Bettstetter, H. Hellwagner, and B. Rinner, "An autonomous multi-uav system for search and rescue," in *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, 2015, pp. 33–38.

[35] M. Campion, P. Ranganathan, and S. Faruque, "Uav swarm communication and control architectures: a review," *Journal of Unmanned Vehicle Systems*, vol. 7, no. 2, pp. 93–106, 2018.

[36] Y. Zhou, B. Rao, and W. Wang, "Uav swarm intelligence: Recent advances and future trends," *IEEE Access*, vol. 8, pp. 183 856–183 878, 2020.

[37] M. Chen, H. Wang, C.-Y. Chang, and X. Wei, "Sidr: a swarm intelligence-based damage-resilient mechanism for uav swarm networks," *IEEE Access*, vol. 8, pp. 77 089–77 105, 2020.

[38] D. Pascarella, S. Venticinque, R. Aversa, M. Mattei, and L. Blasi, "Parallel and distributed computing for uavs trajectory planning," *Journal of Ambient Intelligence and Humanized Computing*, vol. 6, no. 6, pp. 773–782, 2015.

[39] Y. Guo, S. Gu, Q. Zhang, N. Zhang, and W. Xiang, "A coded distributed computing framework for task offloading from multi-uav to edge servers," in *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2021, pp. 1–6.

[40] A. Kopeikin, A. Clare, O. Toupet, J. How, and M. Cummings, "Flight testing a heterogeneous multi-uav system with human supervision," in *AIAA Guidance, Navigation, and Control Conference*, 2012, p. 4825.

[41] Y. B. Sebbane, *Smart autonomous aircraft: flight control and planning for UAV*. Crc Press, 2015.

[42] Kristina Grifantini, "How to make uavs fully autonomous," https://www.technologyreview.com/2009/07/15/211604/how-to-make-uavs-fully-autonomous-2/, 2009, [Online; accessed 30-September-2021].

[43] A. Suarez, A. Caballero, A. Garofano, P. J. Sanchez-Cuevas, G. Heredia, and A. Ollero, "Aerial manipulator with rolling base for inspection of pipe arrays," *IEEE Access*, vol. 8, pp. 162 516–162 532, 2020.

[44] J. Cacace, S. M. Orozco-Soto, A. Suarez, A. Caballero, M. Orsag, S. Bogdan, G. Vasiljevic, E. Ebeid, J. A. A. Rodriguez, and A. Ollero, "Safe local aerial manipulation for the installation of devices on power lines: Aerial-core first year results and designs," *Applied Sciences*, vol. 11, no. 13, p. 6220, 2021.

[45] D. Benjumea, A. Alcántara, A. Ramos, A. Torres-Gonzalez, P. Sánchez-Cuevas, J. Capitan, G. Heredia, and A. Ollero, "Localization system for lightweight unmanned aerial vehicles in inspection tasks," *Sensors*, vol. 21, no. 17, p. 5937, 2021.

[46] G. Schroeder, "Nasa's ingenuity mars helicopter: The first attempt at powered flight on another world." *American Scientist*, vol. 108, no. 6, pp. 330–331, 2020.

[47] N. Potter, "A mars helicopter preps for launch: The first drone to fly on another planet will hitch a ride on nasa's perseverance rover-[news]," *IEEE Spectrum*, vol. 57, no. 7, pp. 06–07, 2020.

[48] S. D. Ramchurn, J. E. Fischer, Y. Ikuno, F. Wu, J. Flann, and A. Waldock, "A study of human-agent collaboration for multi-uav task allocation in dynamic environments," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[49] A. Nikou, J. Tumova, and D. V. Dimarogonas, "Cooperative task planning of multi-agent systems under timed temporal specifications," in *2016 American Control Conference (ACC)*. IEEE, 2016, pp. 7104–7109.

[50] K. Jolly, R. S. Kumar, and R. Vijayakumar, "Intelligent task planning and action selection of a mobile robot in a multi-agent system through a fuzzy neural network approach," *Engineering Applications of Artificial Intelligence*, vol. 23, no. 6, pp. 923–933, 2010.

[51] Y. Gao, Y. Zhang, S. Zhu, and Y. Sun, "Multi-uav task allocation based on improved algorithm of multi-objective particle swarm optimization," in *2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE, 2018, pp. 443–4437.

[52] S. Sanner, "Relational dynamic influence diagram language (rddl): Language description," *Unpublished ms. Australian National University*, vol. 32, p. 27, 2010.

[53] Y.-q. Jiang, S.-q. Zhang, P. Khandelwal, and P. Stone, "Task planning in robotics: an empirical comparison of pddl-and asp-based systems," *Frontiers of Information Technology & Electronic Engineering*, vol. 20, no. 3, pp. 363–373, 2019.

[54] G. Canal, M. Cashmore, S. Krivić, G. Alenyà, D. Magazzeni, and C. Torras, "Probabilistic planning for robotics with rosplan," in *Annual Conference Towards Autonomous Robotic Systems*. Springer, 2019, pp. 236–250.

[55] S. Emel'yanov, D. Makarov, A. I. Panov, and K. Yakovlev, "Multilayer cognitive architecture for uav control," *Cognitive Systems Research*, vol. 39, pp. 58–72, 2016.

[56] A. Klöckner, "Behavior trees for uav mission management," *INFORMATIK 2013: informatik angepasst an Mensch, Organisation und Umwelt*, pp. 57–68, 2013.

[57] N. Monterrosa, J. Montoya, F. Jarquín, and C. Bran, "Design, development and implementation of a uav flight controller based on a state machine approach using a fpga embedded system," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 2016, pp. 1–8.

[58] M. E. Kügler and F. Holzapfel, "Autoland for a novel uav as a state-machine-based extension to a modular automatic flight guidance and control system," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 2231–2236.

[59] V. de Araujo, A. P. G. Almeida, C. T. Miranda, and F. de Barros Vidal, "A parallel hierarchical finite state machine approach to uav control for search and rescue tasks," in *2014 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, vol. 1. IEEE, 2014, pp. 410–415.

[60] P. Ogren, "Increasing modularity of uav control systems using computer game behavior trees," in *Aiaa guidance, navigation, and control conference*, 2012, p. 4458.

[61] F. Real, A. Torres-González, P. R. Soria, J. Capitán, and A. Ollero, "Unmanned aerial vehicle abstraction layer: An abstraction layer to operate unmanned aerial vehicles," *International Journal of Advanced Robotic Systems*, vol. 17, no. 4, pp. 1–13, 2020. [Online]. Available: https://doi.org/10.1177/1729881420925011

Glosario

ACW Aerial Co-Worker. 5–17, 19, 21–25

BT Behaviour Tree. 12, 13, 19–23, 25, 29

FSM Finite State Machine. 19

ID identification. 7, 8, 12–14, 24

ROS Robot Operating System. 3, 14, 25

SITL Software In The Loop. 3

UAL UAV Abstraction Layer. 25

UAV Unmanned Aerial Vehicle. III, 1–3, 5–9, 11, 12, 14–17, 19–21, 23–25, 29

WP waypoint. 7, 17, 23, 25