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:

Acknowledgment

To my tutor, Jesús, for guiding me in this project, for trusting me to join the research group to which he belongs, for supporting me in my decision to join a doctoral program, for always seeking the best for us despite of his preferences and for his kindness.

To all those department mates who have helped me every time I have needed it and all those who have ever volunteered to help. In particular, I would like to thank Fran and Arturo for all the time they have dedicated to helping me.

To Damián, for accompanying me in the easy and difficult moments, but above all, for being my friend, and for being there unconditionally for whatever I needed.

To my classmates, who despite being a difficult year with social distancing, have been as close as ever.

To all my friends, for being such good friends.

To my entire family, for their unconditional love and support, and for their patience and understanding.

Thanks for everything

Álvaro Calvo Matos

Sevilla, 2021

Abstract

This master's thesis has addressed problems arising from the recent increase in the applications of cooperative Unmanned Aerial Vehicle (UAV) teams, which are the autonomy to operate over a long period of time with robustness to possible failures, and the difficulty of providing the team with cognitive capabilities to be able to operate in dynamic environments with humans.

Many of these applications are currently being executed by humans, making the activities much more expensive, time-consuming, and in some cases even dangerous. This is why there is currently a great deal of interest and effort being put into developing solutions to the problems posed.

The aim of the work was to develop cognitive planning techniques for coordinating fleets of quadrotors to assist human operators in inspection and maintenance tasks on high-voltage power lines. These techniques should also extend the autonomy of the system, ensure that safety requirements between drones and human workers are met, and ensure the success of the mission.

A software architecture has been proposed based on a central planner and a distributed behaviour manager. To carry out the planning, a cost has been defined, which is calculated for each task. Thus, each one is assigned to the UAV that consumes the least executing it. On the other hand, to control the behaviour of the drones and ensure the safety of the aerial equipment, a behaviour tree has been implemented.

As a result, it has been possible to develop a software architecture capable of dynamically planning missions while ensuring the safety of the equipment involved. This provides a good base that can be easily adapted and from which more complex planners can be developed in the future. Compared to the typical way of implementing behaviour managers, involving complex finite state machines that are difficult to read, reuse and extend, the use of behaviour trees is a great improvement and will allow the creation of increasingly complex behaviours.

Resumen

Este Trabajo de Fin de Máster ha afrontado problemas que surgen del reciente aumento de las aplicaciones de equipos cooperativos de 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 era desarrollar técnicas cognitvas de planificación para coordinar flotas de drones que asistan a operarios humanos en tareas de inspección y mantenimiento en líneas eléctricas de alta tensión. Estas técnicas debían además extender la autonomía del sistema, garantizar que se cumplen los requisitos de seguridad entre drones 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 comportamiento distribuido. Para llevar a cabo la planificación se ha definido un coste, que es calculado para cada tarea. De esta forma, cada una se asigna al UAV al que cueste menos. Por el otro lado, para controlar el comportamiento de los drones y asegurar la seguridad de los equipos aéreos, se ha implementado un árbol de comportamiento.

Como resultado, se ha conseguido desarrollar una arquitectura de software capaz 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.

Short Outline

Contents

1 Introduction

The use of UAVs has grown considerably in recent years for numerous applications including real-time monitoring, search and rescue, providing wireless coverage, security and surveillance, precision agriculture, package delivery and infrastructure inspection [1]. With the rapidly developing technology in this area, and demonstrations of what UAVs can do, there are increasing efforts to bring this technology to other applications. With the expected increase in applications for this technology, new problems and challenges arise, including autonomy, safety, obstacle avoidance and coordination of multi-UAV teams. Developing the technology to solve these problems will be a major effort, but as UAVs have proven to be critical in situations where humans are at high risk or highly inefficient, and they have proven their capacity to evolve and develop even more potential in the short term, companies are investing in developing all sort of UAV-based solutions.

1.1 Motivation

With the increase in global electricity demand, a challenge has arisen for electricity supply companies to maintain and repair power grids in a way that minimizes the frequency of outages. According to [2], one of the main causes of power outages is damage to transmission lines due to bad weather or inefficient inspection campaigns.

Figure 1.1 Operators getting off the helicopter during a maintenance mission.

The strategy often used by electric companies to reduce power outages is to schedule periodic maintenance operations on active lines. This is the most suitable method if the correct functioning of the system is to be ensured and when replacing a circuit is unacceptable [2]. These maintenance missions are carried out by experienced crews on board helicopters and equipped with safety suits and harnesses among other things that prevent the operators from receiving an electric shock (see figure 1.1). The problem with this solution is that these activities are dangerous for the operators, as they are working at high altitude and on electrified lines, are extremely time-consuming and expensive ($1500 per hour) and are subject to human error [3].

These are the reasons why distribution companies have the need to develop more efficient and safer maintenance methods. Multiple solutions have been proposed to automate this task [4], but the best of them seems to be the use of UAVs because of their flexibility and ability to inspect at different levels [2]. To achieve this, there are still some important barriers to overcome, such as the limited autonomy of these devices, the strong electromagnetic interference to which they would be subjected due to being close to power lines and the ability to detect and avoid obstacles of different nature that can be found in this type of environment [3]. Providing UAVs with the cognitive capability to operate autonomously in such dynamic environments and with the presence of humans, and providing them with a rapid on-line planning method [5], is key to address these complexities and to safely and successfully accomplish the assigned mission with UAV fleets.

A versatile and reliable software architecture will be essential to integrate and interconnect all the heterogeneous components that compose these cognitive multi-UAV systems. In [6], as part of the AERIAL-CORE European project[1], a multi-layer software architecture is presented for carrying out such missions cooperatively between human operators and a fleet of quadrotors. One of the layers of software that compose it is a high-level task planner. Its function is to coordinate the entire fleet of UAVs to generate high-level behaviours in order to efficiently, safely and successfully complete the maintenance or inspection mission. This type of work has the characteristic of being dynamic, since it is not possible to know in advance what the outcome of the inspection as such will be in order to plan offline, but rather, as the mission develops, new tasks will arise that the fleet will have to attend to. Therefore, the task planner should be able to react to unexpected events (new task, failure of a UAV, loss of connection, less autonomy than calculated, etc.) and to replan online. Thus, this layer will be the main cognitive block of the system [6].

1.2 Objectives

The overall objective of this project was to develop a cognitive task planner that would be in charge of governing the behaviour of multi-UAV teams for the inspection and maintenance of power lines in a collaborative way with human operators, being one of the software layers that compose the aforementioned software architecture [6] developed for the AERIAL-CORE European project. The fleet of governed UAVs acts as an aerial co-worker and can perform various tasks such as delivering a tool to an operator, inspecting regions of the power line or monitoring a worker while operating to ensure his safety. The planner receives both high-level input and feedback from the different equipments that make up the fleet, and processes all the information to elaborate a plan to manage the UAV team or modify it in reaction to an unforeseen event. To achieve this, the following objectives were defined:

- Ensure that resources are used and tasks are executed efficiently.
- Comply with all security requirements and ensure the integrity of equipment and mission success.
- Be able to replan online to react to unforeseen events.

[1] AERIAL-CORE European project homepage: *https://aerial-core.eu/*

- Implement the software layer in Robot Operating System (ROS) and manage the necessary communications with the rest of the software layers and modules that make up the architecture.
- Carry out Software In The Loop (SITL) simulations to prove that the algorithm is able to govern the behaviour of the fleet efficiently and safely, and that it is able to react to unforeseen events dynamically, demonstrating cognitive capabilities.
- Design the task planner in such a way that it is easy to maintain, modify or extend, seeking to make it modular and reusable so that it can serve as a basis for the construction of planners for other applications.

2 Preliminaries

This chapter focuses on the current state of the art of those technologies related to this project, as well as on the tools used for the development of the task planner as a software layer of a multi-layer architecture. In addition, the research work carried out on the state of the art in work related to the technologies and techniques used in this project is presented.

2.1 Current technology

Although in the last decade the use of UAVs has spread to a large number of applications, the origin of this technology dates back to 1898 with the invention of radio control and the appearance of the first unmanned aircraft, baptised with the name of drone. These were not yet unmanned aerial vehicles, and were mainly used for military purposes.

Figure 2.1 General Atomics MQ-1 Predator. A Remotely Piloted Aircraft (RPA). Source: Wikipedia.

Later, with the development of technology, the first computers of sufficient size and computing power to run the software necessary to operate a UAV autonomously and even to control aircraft with more complex and even unstable dynamics (gliders [7, 8], airships [9], quadrotors [10, 11, 12, 13], multirotors [14], flapping wings [15, 16, 17], etc.) appeared. Even though computational capacity was still insufficient for some applications, the development of UAV systems was made possible by performing calculations on the ground. What was done was to run the critical and most important systems for autonomous flight on the on-board computer (controls, data acquisition, obstacle

avoidance, etc.), and to run the more demanding calculations that are not necessary in real time on the ground computers [18].

Figure 2.2 GRIFFIN's flapping wing robot [17].

For an aerial vehicle to operate autonomously, it is necessary to acquire data from the environment and process it in real time. A large number of different sensor configurations as well as numerous data acquisition and processing techniques can be found in the current literature [19, 20, 21].

Once UAV technology reached sufficient capacity and autonomy, the first applications for both single [22, 23, 24] and multi-UAV [25, 26, 27] equipment began to appear. There is great interest in the latter, as they can be configured in different ways [28], collect and process data in a distributed way, increasing the computational capacity of the equipment [29, 30], and generate global collective behaviour emerging from interactions between a large number of UAVs that individually are relatively simple, known as swarming [31, 32, 33].

Current applications often require human presence to carry out certain decisions, with the human pilot overseeing that everything runs smoothly and providing the cognitive capacity to analyse the generally dynamic environment and react to unforeseen situations [34, 35, 36]. This is because providing a UAV with sufficient cognitive capacity to operate fully autonomously in dynamic environments is a very complicated task and requires a great deal of processing power. In recent years, UAV technology has evolved rapidly, benefiting from advances in computing and artificial intelligence. As processors are becoming more powerful, efficient and smaller, UAVs are becoming more and more powerful without increasing their weight or compromising their autonomy. With the increase in the number of operations per second that UAVs can perform, this opens up the possibility of using drones for previously unthinkable applications, applications that require a large amount of processing and usually have to be performed in real time [1, 37]. At the same time, advances in artificial intelligence mean that the perception, analysis and sensory fusion capabilities of UAVs are getting better and better. Advances in technology are breaking down one of the barriers preventing UAV technology from achieving this level of autonomy, and with it, more and more research effort is being devoted to breaking down the other barrier, developing software that enables UAVs to have cognitive capabilities.

Mentioning some of the research that is currently being carried out, we can recall the well-known AERIAL-CORE European project[1], in which major European robotics teams are jointly participating with the aim of developing a fully autonomous robotic system with sufficient cognitive capabilities to work together with human operators in inspection and maintenance work on electrical networks [38]. The PILOTING European project[2] aims to develop a complete inspection platform that will

[1] AERIAL-CORE European project homepage: *https://aerial-core.eu/*

[2] PILOTING European project homepage: *https://piloting-project.eu/*

provide its users with the information they need to draw up maintenance plans for structures [39]. HYFLIERS [3] is a completed European project that focused on the inspection of long pipe arrays in hard-to-reach areas. This, unlike the previous two, is not fully autonomous, but needs a pilot to indicate the inspection points along the pipes, and to supervise the aerial robot while it operates [40]. It is also worth mentioning a recent NASA (National Aeronautics and Space Administration) achievement, which is no less than the first flight of an UAV outside the Earth [41, 42]. This is specifically the Martian helicopter called Ingenuity, whose mission was simply to take off, move around and land in the Martian atmosphere with the added difficulty that, due to the distance between the two planets, this had to be done completely autonomously.

2.2 Related work

According to the literature review conducted by Hazim Shakhatreh et al. in 2019 [1], the market value of UAVs for civil infrastructure inspection is expected to be more than $45 billion, representing 45% of the total UAV market. The development of heterogeneous UAV fleets and efficient algorithms for their communication and coordination is important to have multi-UAV teams capable of carrying out a successful inspection and maintenance mission. If the inspection equipment is to be fully autonomous, so that it can be operated by personnel not specialized in the piloting of aerial vehicles, a module capable of reacting to any unforeseen event and modifying the planning in real time if necessary is required. This module, which could be called a task planner, is usually part of a larger software architecture in charge of fleet management and which tries to provide intelligence to the equipment. The task planner developed in this thesis consists of two distinct modules, the task planner itself and the behaviour manager.

2.2.1 Task planning in multi-drone teams

There are numerous proposals for solving the task planning problem, each with its strengths and weaknesses. Given the lack of a rule of when one planner or another is better, Jiang et al. [43] compared the performance of the different planners in the literature. The conclusion they reached was that Planning Domain Description Language (PDDL)-based planners are better on problems requiring long solutions, while Answer Set Programming (ASP)-based planners are less susceptible to domain object augmentation. When complex reasoning is required, ASP-based solutions can be considerably faster than PDDL-based solutions. On the other hand, in [44] they present a standardized integration of probabilistic planners in ROSPlan, which is a framework for task planning in the ROS. By integrating RDDL (Relational Dynamic Influence Diagram Language) into ROSPlan, they allow combining deterministic and probabilistic planning within the same system.

An example of probabilistic planning is presented in [45], where they use an improved multi-objective particle swarm optimization algorithm to solve the task allocation problem for multiple UAVs. The system consists of two phases with which they try to accelerate convergence and avoid the algorithm falling into local minimum. The results shown by this study reveal a good performance in solving this kind of problems. From a completely different approach, [46] proposes an intelligent task planner focused on fuzzy neural networks. This system selects the best action from a set of possible actions. Time-constrained planning is also something that has been researched, e.g. [47] incorporates time constraints into the planning of multi-agent systems. The proposed solution consists of two phases, first all execution times are pre-computed in a decentralized way and then the possible configurations are tested so that the collective execution time is guaranteed. Finally, mention can be made of [48], where the final planning is done by humans, but they have a series of control interfaces and coordination algorithms that assist them in the decision-making process and compute the shortest path for each UAV taking into account the priorities of the tasks and the

[3] HYFLIERS European project homepage: *https://www.oulu.fi/hyfliers/*

type of UAV required in each of them. The human commander can modify the plan or include constraints, such as assigning a task to a particular UAV for example, and will have to review and accept the plan suggested by the algorithm once it is finished.

The task planner proposed in this work tries to group in the same system strong points such as robustness to failures, the capacity to react and replan online in case of any unforeseen event, the incorporation of restrictions and the consideration of factors such as the type of UAV, the priorities of each task or the battery level of each equipment in an automatic way, without the need for human supervision.

2.2.2 Drone behavior management

In this work it has been called drone behaviour manager to what would commonly be controllers and safety modules executed at different times and levels. Once the aerial vehicle has concrete orders of what to do, in this case, once it has an assigned plan, it is time to execute the controls in charge of carrying out that plan. On the one hand, it will be necessary to execute some control in charge of supervising and ensuring the integrity of the airborne equipment. In [49] a Finite State Machine (FSM) is used in which one of the states is in charge of managing emergency situations, to which it transitions when another module detects and communicates the emergency condition. On the other hand, a high-level controller has to be executed in charge of calling the corresponding low-level controllers at any given moment. In this last study, they develop the complete flight control using FSM that transition from one state to another depending on the information provided by the sensors. The use of finite state machines is in fact the most common. In [50] for example, an automatic landing system programmed in this way is presented. Vitor de Araujo et al. [51] present a solution for the control of UAVs in search and rescue tasks based on a Parallel Hierarchical Finite State Machine (PHFSM) with which they claim to achieve many improvements with respect to other typical implementations.

The problem with this approach is that state machines are difficult to scale and reuse. Moreover, there comes a point where it becomes even difficult for a human to interpret. There is therefore a problem with further increasing the capabilities of UAVs using state machines. As discussed in [52], Behaviour Trees (BTs) are an alternative that provide, among other advantages, scalability, modularity and readability, and could be used for UAV mission management. [53] also emphasizes the advantages of using this type of system for UAV control.

Although BTs are already widespread in the videogame industry, there are still not many proposals that use them for the management of autonomous systems. The module in charge of controlling the behaviour of each of the UAVs in this project has been developed using BTs.

2.3 Tools

This section discusses the most relevant software tools used in this project and briefly explains what role they have played in it.

2.3.1 ROS

The Robot Operating System (ROS)[4] is more of a framework than an operating system. It is a collection of tools, libraries and conventions that aim to facilitate the development of software for robots. The aim of this tool is to enable research teams from all over the world to collaborate with each other and to take advantage of each other's work. ROS is present in most robotics projects today. In this project, it forms the basis on which everything else is programmed, as it allows the use of many other very useful tools developed by the community, such as those mentioned below.

[4] ROS homepage: *https://www.ros.org/*

2.3.2 Gazebo

This is a simulation tool usually used by the robotics community to accurately simulate and test their systems. In addition to being open-source, it allows the entire system to be tested, from the design of the robot itself to its programming. In this project, Gazebo has been used to test the developed software safely both in the development phases and in the final stages of testing and verification.

2.3.3 Rviz

Rviz is a tool for visualizing information in 3D. It also allows to graphically represent the information received by a robotic system. It can be integrated with ROS applications, being very useful for debugging tasks. Rviz is used in this project to visualize the position of each of the UAVs as well as different details about the simulation for monitoring and debugging purposes.

2.3.4 UAL

As its name suggests, UAV Abstraction Layer (UAL)[5] is a software layer that simplifies the process of developing and testing high-level algorithms for aerial robots by standardizing and simplifying the interfaces with these robots. In addition, UAL can work with both simulated and real platforms [54]. This software is also available in ROS for free use in any robotics project. Although in this thesis a higher level software layer has been developed, which does not have to communicate directly with the autopilot of any of the aerial vehicles, it has been necessary to communicate with them in order to test the correct functioning of the task planning approach in simulation. Therefore, UAL has been used in this project to programme at a low level the movement of the UAVs, ignoring which autopilot they will incorporate in the future.

2.3.5 Behaviour Trees

BehaviorTree.CPP[6] is a C++ 14 library for creating behaviour trees. Its design features include flexibility, ease of use and speed. This library, among other things, allows the creation of asynchronous Actions, enables reactive behaviours that execute multiple Actions at the same time, permit to load trees at runtime and provides a type-safe and flexible mechanism to do Dataflow between Nodes of the Tree. This library has been chosen over others that fulfil the same function because of its documentation[7] and support. Its function in this project has been the creation of BTs for the programming of the drone behaviour manager in a modular, scalable and easily reusable way.

2.3.6 Groot

Groot[8] is a graphical editor to create BTs. It is compliant with the library used in this project to create behavior trees, BehaviorTree.CPP. This graphic interface can be also used to monitor a running behavior tree. For programming BTs in this project a simple XML file editor will be used, so Groot's role in this project will be to monitor the execution of the BTs during testing.

[5] UAL repository: *https://github.com/grvcTeam/grvc-ual*
[6] BehaviorTree.CPP repository: *https://github.com/BehaviorTree/BehaviorTree.CPP/*
[7] BehaviorTree.CPP documentation: *https://www.behaviortree.dev/*
[8] Groot repository: *https://github.com/BehaviorTree/Groot*

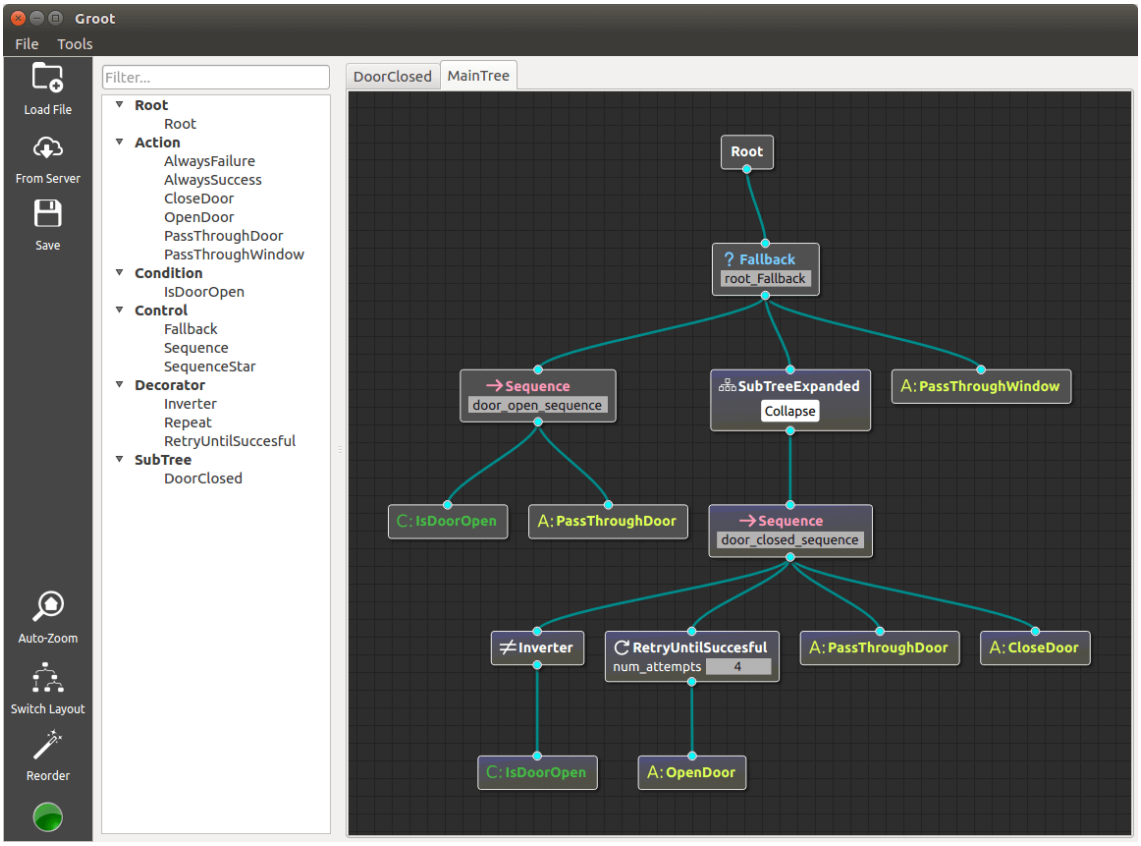

Figure 2.3 Groot graphical interface for editing behavior trees. Source: Github.

3 Problem Formulation

As mentioned in the chapter 1, the context around which this cognitive task planner is being developed is the inspection and maintenance of electrical networks. Although one of the objectives is to build a task planner whose characteristics allow its easy reuse and adaptation for other applications, it is relevant to state the problem for which it is being originally prepared.

The AERIAL-CORE project (H2020-ICT-2019-871479) aims to develop different technologies for the use of multi-UAV equipment in inspection and maintenance tasks in high-voltage electrical installations. In particular, one of the technologies proposed is the use of Aerial Co-Worker, i.e. small teams of cooperative UAVs to safely support maintenance workers while working at height on power lines. These systems would have to interact with humans (see Fig. 3.1) to inspect certain parts that are indicated to them, monitor worker safety during operation and deliver tools or other light equipment, in order to make the work more efficient and safer. In addition, to have a greater impact, the system would need to operate over extended periods of time, being able to autonomously deal with certain faults or recharges.

Figure 3.1 Multi-UAV team supporting an operator. Source: Aerial-Core.

Three types of ACWs are referred, each intended to provide different functionality: *Inspection-ACW*, *Safety-ACW*, and *Physical-ACW*. The use case scenarios can be summarized as follows:

- *Inspection*, where a fleet of ACWs (i.e., *Inspection-ACWs*) carries out a detailed investigation of power equipment autonomously, helping the human workers to acquire views of the power tower that are not easily accessible (see Fig. 3.2);
- *Safety*, where a formation of ACWs (i.e., *Safety-ACWs*) provides the supervising team with a view of the humans working on the power tower in order to monitor their status and to ensure their safety (see Fig. 3.3);
- *Physical*, where an ACW (i.e., *Physical-ACW*) physically interacts with the human worker and provides physical assistance to it, i.e., while in contact with the human it flies stably, reliably, and accomplishes the required physical task (e.g., handover of a tool) without becoming harmful for the human worker (see Fig. 3.4).

Even if there is a specific type of ACW for each of the tasks (inspection, monitoring and tool delivery), this does not mean that a UAV can at any given time undertake a task for which it is not the best. It will therefore be the planner's task to take into account which ACWs are best suited for each task, which are not but could perform it without problem, and which do not have the capacity to perform it at all. As a consequence, the number of ways in which the mission planning can be carried out multiplies, thus considerably increasing the difficulty of the problem that the task planner has to solve.

This mission planning problem with multiple UAVs with battery constraints can be posed as an optimization problem, the solution of which indicates the most efficient way to allocate the different tasks and plan recharges. To react to possible failures, one of the most widespread options is to come up with dynamic methods that can replan in real time as certain events occur. Although there are many variants, most formulations for missions where multiple vehicles visit multiple locations to inspect or make deliveries give rise to NP-hard optimization problems and, therefore, the most widespread approach is to solve them using heuristic algorithms.

Uncertainty planning methods are appropriate for adding cognitive capabilities to a system that has to interact with humans in dynamic environments, as they allow optimizing plans by predicting the most likely intentions of humans and the outcomes of future actions. The main problem is their computational complexity, as the plan search space would grow exponentially with the number of UAVs and with the future time horizon over which planning is to take place.

It is in this context and with these ideas in mind that the cognitive task planner was developed. As this is one of the software layers that make up the architecture that solve the problem, in order to present the details on which the planner has been designed it is necessary to at least talk about what information exchanges exist between the upper and lower layers of the software architecture, describe the interfaces by which this information travels and highlight when control is passed to lower levels of the software architecture. In the following section this information is presented by individually explaining the different tasks contemplated in the project.

On the other hand, a review will be made of other important considerations that the planner must take into account such as battery recharges, connection losses and task rescheduling; analyzing the different situations in which each of them can occur and their different causes.

3.1 Description of tasks

As mentioned above, three different types of tasks are envisaged in the project. These tasks are requested at any time by human workers through gestures. There will be a higher level software layer that processes the information contained in the gestures so that the planner receives an asynchronous communication from the upper layer containing the specifications of a new task. At this point, it is the planner's job to process the new information together with the information it already had in order to elaborate and implement a new plan. The same planner is also in charge of calling the

low-level controllers when necessary and ensuring the safety of the equipment and the fulfillment of the mission. Each task is explained in detail below.

3.1.1 Inspection tasks

This task can be performed by all three types of ACWs. It is the second highest priority task, with the tool delivery task being the only one that exceeds it. It consists of carrying out a detailed inspection of the specified areas of the power equipment. The layer immediately above the task planner is responsible for passing it a list of waypoints (WPs) that define the inspection task, and the planner is responsible for deciding how many ACWs it recruits to execute the task, which of the available ACWs it assigns it to, and which subset of the list of WPs it assigns to each one. In turn, once the planning is executed, the tasks of this type are transmitted to the lower level layers containing the subset of WPs and the identifications (IDs) of the selected UAVs.

All the above-mentioned communications will be carried out asynchronously, as the creation of the task by the workers, which triggers the whole sequence of actions, is done in this way.

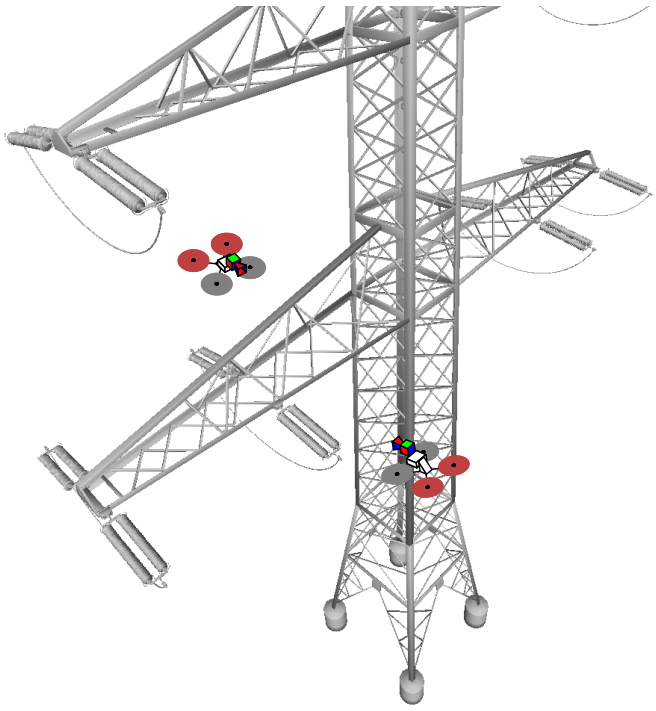

Figure 3.2 *Inspection-ACW* carrying out an inspection task.

3.1.2 Monitoring tasks

This task can also be executed by all three types of ACWs. It is the lowest priority task. Monitor worker's safety consists of providing the supervisory team with a view of the people working in the power tower to monitor their status and ensure their safety. The layer immediately above the task planner communicates this time the ID of the worker to be monitored, the number of UAVs wanted and the distance they should keep from the worker. It is the task planner's responsibility to decide once again which of the available ACWs to assign this task to and the formation they should maintain during the flight. Once the planning has been executed, the tasks of this type are passed on to the lower level layers with both the original information and the information resulting from the planning.

The above-mentioned communications will also be carried out asynchronously for the same reason.

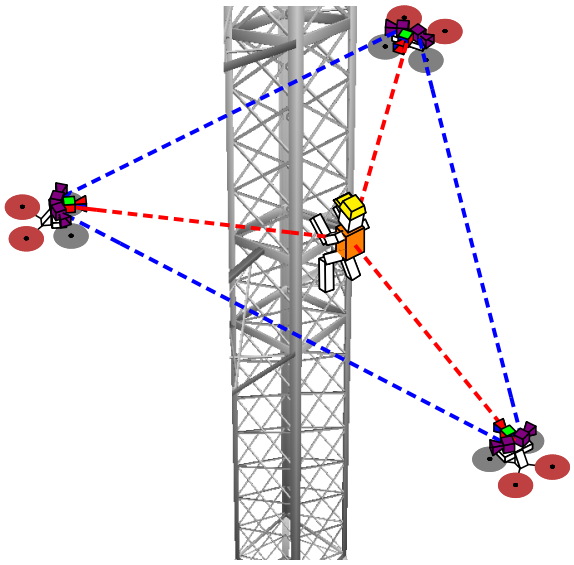

Figure 3.3 *Safety-ACW* carrying out a monitoring task.

3.1.3 Tool delivery tasks

This task can be performed only by *Physical-ACW* UAVs, as special hardware is required to perform the physical interaction with the low-weight objects and the human. This is the highest priority task. Delivering a tool consists of picking up a tool and transporting it to the worker, with whom a physical interaction will take place through which the delivery of the tool will take place. Low-level controllers will have to be especially precise and careful not to hurt the worker. This time, the layer immediately above the task planner communicates the ID of the worker to whom the tool is to be delivered and the ID of the requested tool. Again, the task planner's mission is to decide which of the available ACWs to give this task to. Once the planning is executed, tasks of this type are passed on to the lower level layers with the same information as originally.

The above-mentioned communications, once again, will be carried out asynchronously.

Figure 3.4 *Physical-ACW* carrying out a tool delivery task.

3.2 Battery recharges

Given the current autonomy problem with UAV technology, eventually each of the ACWs involved in the mission will run out of battery power. The moment when the battery of each one will run out can be estimated from the mission planning itself, so the planner can in turn take this into account when distributing the tasks so that the ACW itself anticipates this event. Recharging does not necessarily have to take place when the UAV is about to run out of battery, nor does it have to take place until the battery reaches its maximum, but both will be parameters to be taken into account during the mission planning and optimization process.

On the other hand, it is possible that the calculations may fail for some reason, and the battery may run out sooner than expected. It will therefore be necessary to periodically read the battery status and perform emergency recharging and replanning if necessary. One possible scenario is that the aerial agent runs out of battery during a loss of connection. Since the planner is centralised at a ground station, there must also be a battery check and action module on board each aerial vehicle, and an emergency protocol in case this happens.

In the absence of specifications, it is assumed that battery recharging does not occur instantaneously (it is not a battery change), so reaching the desired battery level takes time and should be considered in the plan.

Also, the task planning algorithm must be able to handle without blocking in case all ACWs are simultaneously without sufficient battery power and therefore there is no UAV with which to execute a task immediately.

3.3 Connection losses

Another important consideration is the possible loss of connection between the centralised planner, where most of the cognitive capacity is concentrated, and one of the ACWs. Since a loss of connection is an unforeseen event, it is most likely that the planner will recalculate the optimal task allocation once the UAV fleet is updated, so that the tasks previously assigned to the disconnected ACW will be executed on another one. This is a potentially dangerous situation because the disconnected UAV could act autonomously according to its last plan and cause an accident with the rest of the agents that are still online.

It is therefore important to implement, on the one hand, a system to detect disconnections both from one end of the communication and the other, and on the other hand, to establish a common action protocol so that both modules know how the other is going to act, thus ensuring the integrity of all equipment and the safety of workers.

3.4 Task replanning situations

Once the mission is underway, any unforeseen event has the potential to completely change what the optimal plan is. Therefore, even if there is a possibility that the event will not affect the mission at all, it will always be necessary to execute a mission replanning in case of an unforeseen event.

The following is a list of the unforeseen events that have been contemplated in this problem:

- Arrival of a new task.
- Modification of a task's parameters.
- Connection of a new ACW.
- Disconnection of an ACW.
- Unplanned insufficient battery in one of the ACWs

- Battery drain faster than expected and therefore will not be enough for the current plan.
- An ACW finishes recharging ahead of schedule.
- A task is successfully completed.
- A task is completed unsuccessfully.

It should be explained that some of the events considered are not really unexpected. Successful completion of a task is what is desired, for example, so it should not imply a change of plans. However, this event is included in the list because it is a good moment to check if there is a better plan and to modify the current one if necessary. As the planner is pursuing the optimal plan, the result of the replanning will keep the previous plan unchanged if it is still optimal.

4 Design of the proposed solution

This section provides more details about the implementation of the solution to the problem: node diagram, pseudocode and inter-module communications. All the code is available online[1], and was developed under the Ubuntu 18.04 operating system and ROS Melodic.

The solution proposed for the problem formulated in the previous section (see section 3) follows a hierarchical approach, with a high-level planner in charge of activating different low-level controllers. The high-level planner detects the tasks required by the operators, and distributes them from the ground in a centralised way among the available ACWs, planning the necessary reloads throughout the mission. In addition, this planner reacts in real time to possible failures by reassigning tasks. The low-level planners are on board each UAV and are responsible for executing contingency plans for these failures while the central planner calculates and communicates the new plan. They will also be in charge of controlling the movement of the ACWs to execute the different tasks assigned by the higher-level module (e.g. flying to a location to be inspected or to the position of an operator waiting for a tool). From now on, the low-level module on board each UAV will be called the *Agent Behaviour Manager*, and the centralised module on the ground will be called the *High-Level Planner*. Together, these modules have cognitive capabilities enough to interact with humans efficiently.

4.1 Node diagram

As stated in chapter 1, the developed task planner is part of a software architecture consisting of different layers, being the main cognitive block the central layer, the *high-level cognitive task planner*. Figure 4.1 shows a schematic of the software architecture from the perspective of the module implemented in this project, including the nodes that form it and their interfaces. The part of the diagram painted in grey would be the complete software architecture, including from the high-level module in charge of analyzing the gestures made by the operators to extract the tasks from them; to the low-level controllers in charge of executing those tasks. The software layer corresponding to this thesis, in charge of high-level decision-making, is marked in blue-green. It is composed of the *High-Level Planner*, which is centralised and runs on a ground station, colored in orange; and the *Agent Behaviour Manager*, distributed on board each ACW, painted in lime.

In the software architecture scheme, although some communications are bidirectional, it can be seen that there is a main flow of information. Starting with the information arriving at the node *Gesture Recognition*, this propagates to the last layer, where the *Lower-Level Controllers* use the already processed information to command the ACWs. The table 4.1 shows the type of data that each of the nodes in the figure 4.1 receives as input and the type of data that each of them emits as output. Additionally, table 4.2 explains what each one of the data mentioned in the previous table consists of.

[1] Human aware collaboration planner source code: *https://github.com/grvcTeam/aerialcore_planning*

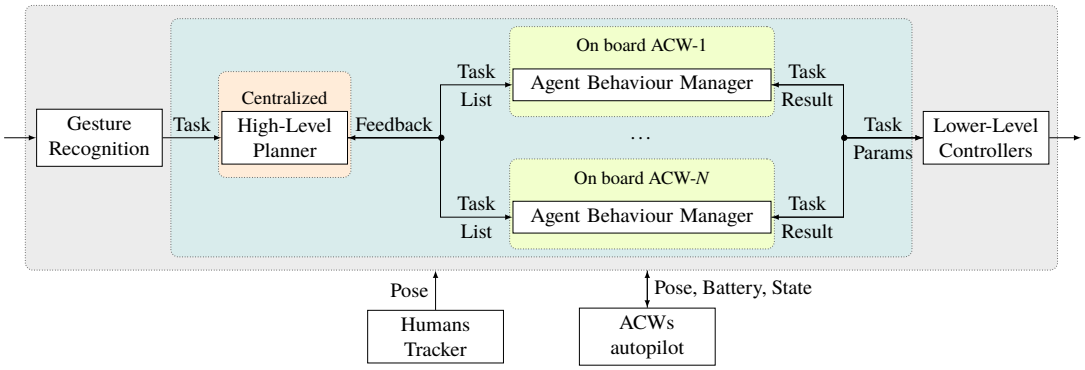

Figure 4.1 Software architecture: nodes and interfaces. Node diagram from the high-level cognitive task planner perspective.

Table 4.1 Description of the data interfaces for each software module.

Module Name	Input Data	Output Data
Gesture Recognition	Images	**Task, defined by:** Task ID, Task Type, Monitoring Distance, Monitoring Number, WP List, Tool ID (some task parameters will be ignored depending on Task Type)
High-Level Planner	Task, Feedback (Task Result, BatteryEnough, BT info), Humans' Pose, ACWs' Pose, Battery and State, and Agent Beacon	Task List adding to each one its extra parameters result of the planning (Formation and/or List of ACWs' IDs) and Planner Beacon
Agent Behaviour Manager	Task List, Low-Level's Result, Human Pose, ACWs Pose, Battery and State	Params needed by Low-Level Controllers (depending on Task Type), Feedback (Task Result, BatteryEnough, BT info) and Agent Beacon
Low-Level Controllers	Params (depending on Task Type)	Result
Humans Tracker		Pose
ACW autopilot	Low-Level orders	Pose, Battery and State

The first node is constantly checking the images captured by the UAVs for a gesture that is indicating a new task or the modification of an existing task. When this occurs, it asynchronously emits a task, which will be picked up by the centralised planner. As shown in the table 4.1, this communication includes the unique ID that differentiates this task from the others, the type of task and the parameters that define it.

On the other hand, the *High-Level Planner*, when it receives this information, proceeds to re-evaluate the optimal plan taking into account the task received, the information it receives from the *ACWs' autopilot*, and the position of the operators, which is periodically published by the *Human Tracker*. The aforementioned data set constitutes the input data for the *High-Level Planner* together with the feedback coming from each *Agent Behaviour Manager*. Its output data being a list of tasks for each glsACW.

On board each ACW is the *Agent Behaviour Manager*. This node is in charge of collecting the corresponding task list provided by the centralised planner. With this input information and the information coming from the *Human Tracker* and the *ACW's autopilot*, this module is in charge of calling the *Lower-Level Controllers* to carry out the execution of the assigned plan. The information emitted by the *ACW's autopilot* is also used to check that everything is working correctly and to execute the security protocols in case they are necessary. If this happens, the corresponding

Table 4.2 Description of data types.

Data name	Data type	Comment
Task ID	String	Unique identifier of each task
Task Type	Integer	Task type indicator: m/M, i/I or d/D
Human Target ID	String	Unique identifier of each human worker. The position of the human target and other needed info is supposed to be known and accessible via its ID.
Monitoring Distance	Float	Distance from which the ACW surveil the worker during a safety monitoring task
Monitoring Number	Integer	Number of ACWs that are required in formation for a certain safety monitoring task
WP List	List of 3 float tuples (x, y, and z)	List of waypoints to be inspected
List of ACWs' IDs	List of Strings	List of the unique identifiers of the ACWs that have been selected for a task that requires multiple ACWs
Formation	Integer	Indicates which of the predefined types of formations should be used for monitoring (e.g., circle, triangle)
Tool ID	String	Unique identifier of the tool to be delivered
ACW's Pose	geometry_msgs /PoseStamped	ACW's Position and orientation
ACW's Battery	sensors_msgs /BatteryState	Percentage of battery in the ACW
Task Result	String, Boolean	First one is the task unique ID and second one its result once it's finished
Battery Enough	Boolean	Result of computing if an ACW will have enough battery for its current task
BT info	String list	Status of each BT's node in its last execution (Running, IDLE, SUCCESS or FAILURE)
Agent Beacon	String, String	First one is the ACW's unique ID while the second one defines ACW's type (SafetyACW, InspectACW, or PhysicalACW). It is used as heartbeat and to detect new ACWs in Planner
Planner Beacon	Time	ROS::Time message containing the time when the beacon was sended. It is used to check the status of the connection from Agent's side.
Lower-Level's Result	Boolean	Result of the Lower-Level Controllers once they have finished after being called

communication would be issued back to the *High-Level Planner* node in order to calculate a new plan. This node also receives the *Lower-Level Controllers*' result after calling each of them, and publish back to the *High-Level Planner* some feedback.

In addition to these communications, the nodes *High-Level Planner* and *Agent Behaviour Manager* periodically exchange beacons that are used to detect both the connection of a new ACW and its disconnection in case of failure. Finally, there is an asynchronous communication that is broadcast to all nodes indicating the end of the mission when this happens.

Finally, it is worth mentioning that the *Gesture Recognition* node does not have a communication aimed at modifying parameters of a task already contemplated within the *High-Level Planner*. However, this is possible because tasks have a unique identifier. Once a task has been delivered to the *High-Level Planner*, in order to change any of its parameters, the *Gesture Recognition* node just has to submit the task again, keeping the same task ID and updating only the desired parameters. Thus, the *High-Level Planner* would overwrite it and allocate it again with the new parameters.

4.2 Centralized module: High-Level Planner

As mentioned above, the *High-Level Planner* is a centralised module running on a ground station and constitutes the main cognitive block of the software architecture of which this project is a part. Its purpose is to plan the mission in an optimal way, i.e., to distribute the pending tasks among the available ACWs by specifying the order in which they are executed and taking into account the time it takes to complete each one, the type of each UAVs, the distance each one will have to travel, the battery they have available, the task each one was executing, the priority of each task, the battery consumed by each task, the recharges that will be needed and when it is best to carry out the recharges.

The general pseudocode for this node from launch to termination is contained in the code 4.1.

Code 4.1 General operation of *High-Level Planner*'s code.

```
1. Read from a ros::param the address of the configuration file.
2. Read from the configuration file all necessary information.
3. Configure ROS communications (Publishers, Subscribers and
   ActionServers).
4. Set the loop rate.
5. Main "while" loop. While ros::ok() and not mission over do:
  5.1. Check the timeout of the Agents' beacons.
  5.2. Publish a new Planner beacon.
  5.3. Check for pending incoming communications (ros::spinOnce).
  5.4. Sleep the remaining time to send the next beacon.
6. Wait until all UAVs have finished and disconnected. While there is
   any agent connected do:
  6.1. Check the timeout of the Agents' beacons.
  6.2. Check for pending incoming communications (ros::spinOnce).
  6.3. Sleep for a while.
```

Since the environment in which the UAVs operates is dynamic, this module has been programmed in such a way that it can react to unforeseen events and recalculate the optimal plan. As can be deduced from the 4.1 pseudocode, everything works through callback functions. Every time a communication arrives from another node, a response is triggered on that node. The information contained in the message is analyzed and it is decided whether a replanning is necessary or not. The situations in which a replanning has been deemed necessary are listed in section 3.4. The communications summarized in the tables 4.1 and 4.2 and in the figure 4.1 are sufficient to detect these unforeseen events and to be able to respond to them in the best possible way.

Code 4.2 Task callback pseudocode.

```
1. Check if the task already exists and delete it in order to create
   it with the new parameters.
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.
```

Firstly, there is the callback that is executed when the node *Gesture Recognition* sends a task, which always ends up calling the function in charge of calculating the optimal plan (see code 4.2); 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 the code 4.3.

Code 4.3 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.

Code 4.4 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* node 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 the code 4.4 consequently 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 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, on the other hand, conditions have changed since the last planning and a better plan now exists, it is at this point that the plan is updated. On the other hand, 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 4.5.

Code 4.5 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 the code 4.1, 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 the code 4.6. Simply, 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 the agents, the number of connected UAVs has decreased, i.e. if any of the previously connected UAVs has disconnected, a mission replanning is executed.

Code 4.6 Beacons' timeout check function.

```
1. For each agent connected:
  1.1. If the elapsed time since the last beacon is grater 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 node'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 summarized in the code 4.7. 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 be more urgent, and therefore will be given higher priority. Therefore, when a new task is received, it is stored both in the *std::map* that contains all the pending tasks to facilitate the access to the information, and in the *std::vector* of its task type, where the order of arrival is maintained. What this simplification allows is to assign tasks one at a time. By having a prioritized 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 must be remembered 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 a simple scenario, it makes sense to make this simplification 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 4.4.

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 penalizes the assignment of tasks to those UAVs designed for another type. It penalizes 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 is at 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 penalizes 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.

Code 4.7 Simplified 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
       battery enough.
    1.2.2. Assign the task to the agent for who the task cost the
       least (from those who has battery enough).
    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) tha has battery enough:
      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 battery enough, continue.
      1.3.5. Else, if the number of agents to select is equal to zero,
         assign the task to the agent that cost 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 battery enough.
      1.4.2. If 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
       Behavior Manager.
  2. Else:
    2.1. Warn operators that any 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* node returns to the main while loop to continue waiting until an event that triggers a replanning occurs again.

4.3 Distributed module: Agent Behavior Manager

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

The general structure of this node is quite similar to that of the central node. The pseudocode is summarized in the code 4.8. Upon initialization, the *Agent Behaviour Manager* prepares the necessary information to start its operation, configures the necessary communications, declares and initializes the behaviour tree and, once the UAV it controls has finished initializing, starts sending beacons to the central node to inform it of its joining the mission. Once the code finishes initializing 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.

Code 4.8 General operation of *Agent Behaviour Manager*'s code.

```
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. Initialize each BT node.
8. Start BT loggers to to facilitate debugging and monitoring of the
   node's performance.
9. Wait until the ACW fully initializes.
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. BT can be defined as an improved FSM. They are a more advanced mechanism to implement behaviors, 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 nothing like 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 yet have enough intuition to know which one is better. Taking advantage of the fact that the use of BT 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. The examples found in [55, 56, 57] 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 and the functioning of each of them will be briefly discussed.

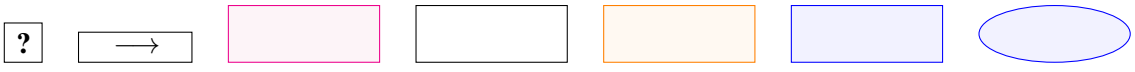

Figure 4.2 Different types of nodes that can be present in an BT.

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 4.2a), which try success calling one by one each of their children; or *Sequence* nodes, represented with an arrow (see subfigure 4.2b), which call their children in order if the previous one have succeeded. *Fallback* nodes return *SUCCESS* if one of its children does it, *FAILURE* if none of them return *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*, *Sequence* node does it too. *Control* nodes are represented in a black

rectangular box when they are the standard ones (see subfigure 4.2d), but they could also be *Reactive* control nodes, represented by a magenta box (see subfigure 4.2c), 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 necessary 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 4.2e), 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 4.2g), that check a condition and return either *SUCCESS* or *FAILURE*; or *Action* nodes, represented in a blue rectangular box (see subfigure 4.2f), that execute code that take longer to execute and therefore these nodes could also return *RUNNING*.

4.3.1 Main tree

The design of both the behaviour tree and the *Agent Behaviour Manager* node in general has been made with the aim of concentrating the minimum necessary intelligence in it to ensure the success of the mission and the safety of the equipment 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 variable.

Not all intelligence and decision-making can be placed in the ground station node. There must 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 4.8). 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 power, 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. On the other hand, 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, for example. 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 the section 4.3 until eventually a *leaf* node returns one of three possible responses (*SUCCESS*, *RUNNING* or *FAILURE*), which will be propagated back, being the result that the root node returns to the main while loop.

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 an 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 BT taking advantage of the modularity offered by this approach. The main tree is represented in the figure 4.3, being the node named as *Main Tree* the root of the complete tree.

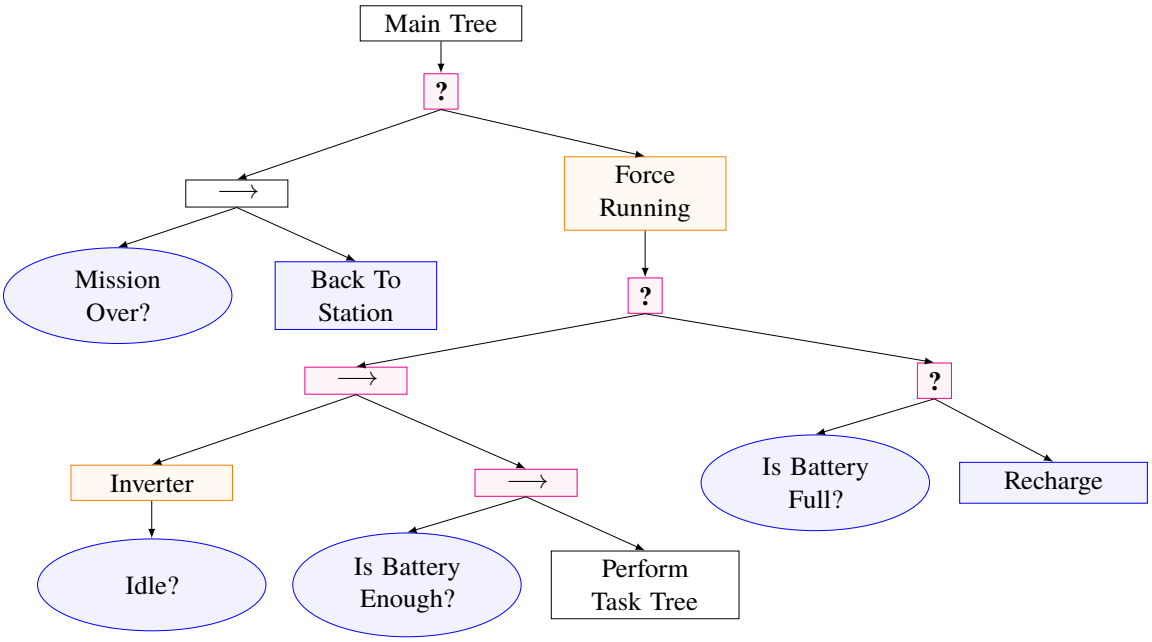

Figure 4.3 Behavior Tree: Main tree.

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 Fig. 4.4).

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 start 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.

The *Perform Task Tree* checks which is the first task in the queue, which task 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

[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 point". The location of the charging stations is known in advance or provided as input by the High-Level Planner/Behavior Tree.

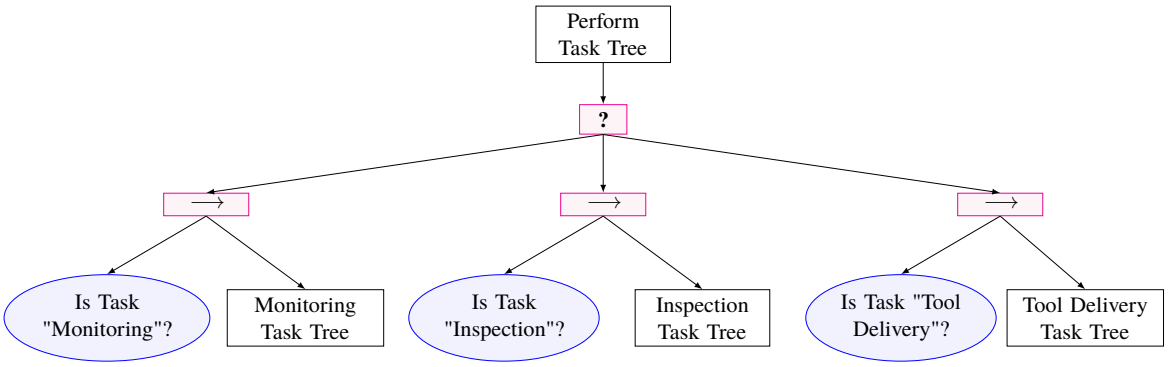

Figure 4.4 Behavior Tree: Perform Task Tree.

execute the task directly. However, it was decided that 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 4.6, 4.5 and 4.7 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 must communicate their results (success or failure) asynchronously back to the *Agent Behavior Manager*, so that the BT could continue running.

In the following, a brief description on how to carry out each of the available tasks in the system. It is assumed that there are Low-Level Controllers running on the ACWs to perform basic navigation actions, formation control for human worker monitoring, inspection operations, and physical interaction with the human worker (e.g., to pick or deliver a tool). These Low-Level Controllers operate in a known environment, represented by a map (i.e., an occupancy grid map) that also includes the position of obstacles and the power tower.

4.3.2 Inspection task tree

High-Level Planner inputs: Task ID, Task Type and Waypoint List (the rest will be ignored). **Description**: the High-Level Planner, from the list of waypoints, will decide how many ACWs are required and which part of the waypoint list is assigned to each one. Then, it would send to each corresponding Agent Behavior Manager the task parameters, including a list with the IDs of the selected glsplACW. The same information is forwarded to the Lower-Level Controllers when the BT calls them (see Fig. 4.5). Basically, the list of waypoints to be inspected are covered by the assigned ACWs, stopping at each of them to take images.

4.3.3 Monitoring task tree

High-Level Planner inputs: Task ID, Task Type, Human Target ID, Monitoring Distance, and Monitoring Number (the rest will be ignored). **Description**: the High-Level Planner will assign this task to as many Safety-ACWs as specified Monitoring Number. The formation will be chosen by the High-Level Planner from a set of fixed formations (to be listed) depending on the number of ACW. Each selected ACW will know a list with the IDs of the ACWs selected for the task and the formation that they must take. As shown in Fig. 4.6, each Agent Behaviour Manager would individually navigate each ACW near the human target and then, it would call the corresponding Lower-Level Controller for formation control. Extra ACWs could even be added to the formation at any time, just updating the task parameters sending a new task from Gesture Recognition.

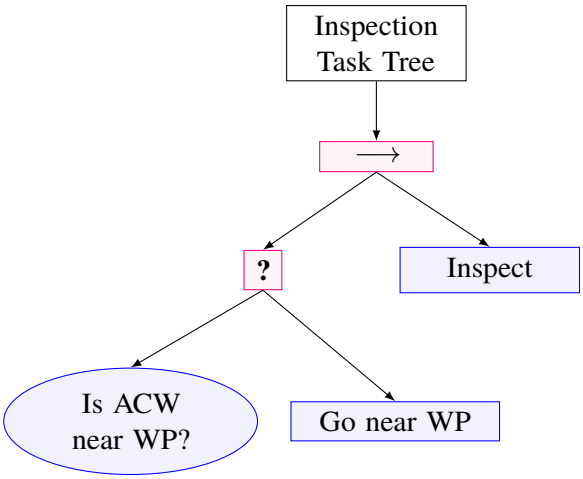

Figure 4.5 Behavior Tree: sub-tree that controls the inspect tasks.

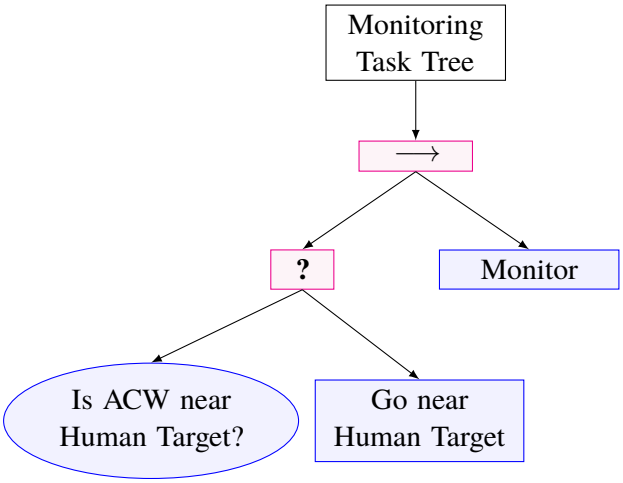

Figure 4.6 Behavior Tree: sub-tree that controls the safety monitoring tasks.

4.3.4 Tool delivery task tree

High-Level Planner inputs: Task ID, Task Type, Human Target ID and Tool ID (the rest will be ignored). **Description**: After task allocation, the High-Level Planner will send the information to the corresponding Agent Behavior Manager and from there, the Lower-Level Controllers will be called sequentially as shown in Fig. 4.7. Basically, the ACW needs to navigate to the station where the tool is, pick it up, navigate back to where the worker is, and start physical interaction to deliver the tool.

4.4 Lower and upper level modules faker

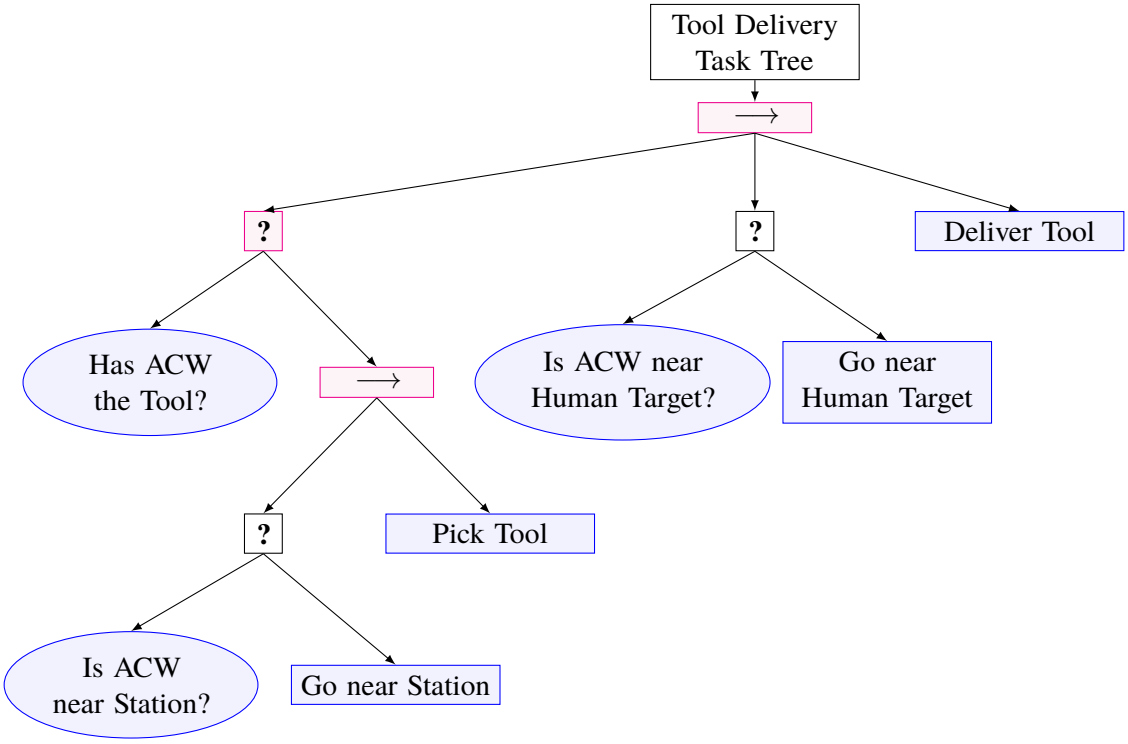

Figure 4.7 Behavior Tree: sub-tree that controls the tool delivery tasks.

5 Results

Lorem itsum

5.1 Task planning

5.1.1 Battery

5.1.2 Connection lost

5.1.3 Replanning

5.2 Drone behaviour manager results

5.2.1 Battery management

5.2.2 Connection lost management

5.2.3 Replanning management

6 Conclusions and future work

6.1 Conclusions

6.2 Future work

6.2.1 Augmented reality

List of Figures

List of Tables

List of Codes

Bibliography

[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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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].

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

[35] 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].

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[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] 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.

[50] 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.

[51] 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.

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

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

[54] 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

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

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

[57] 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].

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

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

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

[61] 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.

Glossary

ACW Aerial Co-Worker. 11–23, 25–30

ASP Answer Set Programming. 7

BT Behaviour Tree. 8, 9, 18, 19, 25–28, 35

FSM Finite State Machine. 8, 25

ID identification. 13, 14, 18, 19

NASA National Aeronautics and Space Administration. 7

PDDL Planning Domain Description Language. 7

PHFSM Parallel Hierarchical Finite State Machine. 8

RDDL Relational Dynamic Influence Diagram Language. 7

ROS Robot Operating System. 3, 7–9

RPA Remotely Piloted Aircraft. 5, 35

SITL Software In The Loop. 3

UAL UAV Abstraction Layer. 9

UAV Unmanned Aerial Vehicle. III, V, 1, 2, 5–9, 11–15, 17, 18, 20–24, 26, 27, 35

WP waypoint. 13, 23