ROS-MAGNA, a ROS-based framework for the definition and management of multi-UAS cooperative missions

Jose A. Millan-Romera, Hector Perez-Leon, Alejandro Castillejo-Calle, Ivan Maza and Anibal Ollero

***Abstract*— This paper presents a general framework for the definition and management of cooperative missions for multiple Unmanned Aircraft Systems (UAS) based on the Robot Operating System (ROS). This framework makes transparent the type of autopilot on-board and creates the state machines that control the behaviour of the different UAS from the specification of the multi-UAS mission. In addition, it integrates a virtual world generation tool to manage the information of the environment and visualize the geometrical objects of interest to properly follow the progress of the mission. The framework supports the coexistence of software-in-the-loop, hardware-in-the-loop and real UAS cooperating in the same arena, being a very useful testing tool for the developer of UAS advanced functionalities. To the best of our knowledge, it is the first framework which endows all these capabilities. The paper also includes simulations and real experiments which show the main features of the framework.**

***Index Terms*— Mutiple UAS, Robot Operating System, Simulation**

I. INTRODUCTION

The applicability of Unmanned Aerial Systems (UAS) in civil missions such as firefighting, critical infrastructure protection or remote surveillance, among many others, is clear nowadays. The multiple variety of platforms, control systems and ground-based equipments, and the heterogeneity of the communication devices have made difficult the interoperability among different systems. Each manufacturer has produced its own infrastructure which manage the information of the mission in a native format. This drawback implies that complex missions with a broad range of autonomous vehicles are highly complicated for planning, execution and monitoring.

However, the Robot Operating System (ROS) [1] is the 'de facto' standard for robot application development and it is also used for the development of autonomous vehicles. Its architecture and communications, based on topics, services and actions, makes it ideal to control a distributed network of sensor and actuator in a common work frame with well-defined data structures. ROS is scalable and suitable for a range of different platforms that can be easily integrated into a single environment. Then, ROS is a very useful tool in the context of multi-UAS cooperative systems.

This work is partially supported by the MULTIDRONE (H2020-ICT-731667) and AEROARMS (H2020-ICT-644271) European projects.

The authors are with the Robotics, Vision and Control Group, University of Seville, Avda. de los Descubrimientos s/n, 41092, Sevilla, Spain. Email: josmilrom@gmail.com, hectorperez@us.es, a.castillejocalle@gmail.com, imaza@us.es, aollero@us.es

Until now, no many works can be found about multi-UAS architectures implemented under ROS for the definition and management of missions. For example, ATLAS [2] is a framework developed for ROS that only addresses cooperative localization for Unmanned Aerial Vehicles (UAVs) based on cameras and fiduciary markers. On the other hand, the multi-UAV control testbed presented in [3] is focused on a GPS waypoint tracking package and a centralized task allocation network system (CTANS) developed under ROS. In the field of central control frameworks, an interesting event-based real-time Nonlinear Model Predictive Control (NMPC) Framework with ROS Interface for multi-robot systems is presented in [4].

Regarding teleoperation, the TeleKyb framework [5] is an end-to-end ROS software framework for the development of bilateral teleoperation systems between human interfaces and groups of quadrotor UAVs. Also in the area of teleoperation, considering the fact that the operation of multi-UAV system may need multiple cooperative operators, an algorithm based on position information and color information is described in [6] to identify multiple operators. The results of hand gesture recognition of multiple operators with UAV control under ROS are also presented.

Some works related to virtual reality and multiple-UAS can be also found. In [7], Unity-based virtual reality interfaces are developed for immersive monitoring and commanding interfaces, able to improve the operators situational awareness without increasing its workload. Three applications are presented: an interface for monitoring a fleet of drones, another interface for commanding a robot manipulator and an integration of multiple ground and aerial robots. Also for Unity, ROSUnitySim [8] presents an efficient high-fidelity 3D multi-UAV navigation and control simulator in GPS-denied environments.

Closer to the topic of our paper, some works present an approach for the control of an UAS swarm that performs a task in a coordinated way. A guide to use ROS to fly a Bitcraze Crazyflie 2.0, a small quadcopter platfrom, both individually and as a group is presented in [9]. In [10], it is proposed a support system for supervision of multiple unmanned aerial vehicles by a single operator supervising its cooperative behaviour. The FlyMASTER project [11] develops a software platform for cooperative swarming by high-level control and supervision of multi-agent UAS systems that is platform agnostic, capable of communicating with any flight controller that implements the MAVLink.

Our paper presents ROS-MAGNA (ROS-based Multi-

AGent mission maNAgement), which is a general framework to manage cooperative missions for multiple UAS based on ROS. Its main contribution is to enhance ROS built-in tools to offer the developer an abstracion layer for several features required for a mission so to focus on obtaining research results with less time and effort consumption. Multi-UAS mission specifications are implemented as state machines that control the behaviour of the different UAS independently of its autopilot on-board. In addition, a virtual world generation tool has been designed to offer an overall visual understanding of the state of the main geometrical elements and its progress on the mission. The framework is focused on offering useful testing tools on the whole development pipeline of UAS advanced functionalities, from simulated software-in-the-loop and hardware-in-the-loop to real UAS even at the same time and arena. As far as we are concerned, no other framework endows all these capabilities

This paper is structured as follows. Section II describes the general framework and the main role of each component. Section III presents the modules running on the ground station that controls the mission, except the component concerning the modelling of the world, which is presented in Sect. IV. The software running on-board the UAV is described in Sect. V. Simulations and real experiments which show the main features of the framework are included in Sect. VI. Finally, Section VII closes the paper with the conclusions and future work.

II. OVERALL FRAMEWORK

The general framework is mainly derived by the structure of ROS and its communications, with different nodes distributed on the Ground Station (GS) or on-board the aerial vehicles. Every node is subdivided into components. Therefore, every node is governed by a parent component and a network of other diverse, adjacent ones. Each component is implemented as a Python class. The standard way to introduce the different parameters of the mission is via JavaScript Object Notation (JSON) files or on the top front-end script.

Our software development is based on different tools. Regarding the low level, which interacts with the on-board autopilot, we have used a previous software development of our research group called UAV Abstraction Layer (UAL) [12]. Thanks to its versatility, this framework can control UAS with autopilots supporting the MAVLink protocol, and or autopilots by manufacturers such as DJI and Crazyflie. SMACH [13] is the ROS library applied for the mission and UAS internal state machines. Visualisation tools such as RViz [14] and SMACH viewer [15] provide a full comprehensive insight into the state of the mission.

Figure 1 offers a general view of the framework built upon an architecture divided into the ground segment, which is executed on land devices, and the aerial node, which is designed to be run onboard the UAS but would also be executed on land for simulation purposes. On top of it, the master node is the front-end where the main features of the mission are defined. It remains active from the beginning of the collection of missions to be performed. For every new mission, the master node spawns its corresponding GS, the storage folders for the generated data and to start, if it is required, the simulator.

A. Ground Segment Nodes

The next node in the hierarchy is the Ground Station (GS) node, which embeds several utilities for the management and control of the multi-UAS mission. The GS node is the central node which sends coordinated commands to the rest of nodes based on a State Machine (SM) that decides actions depending on the current state of the UAS and the status of the mission. It also reports about the performance and success or failure once the mission is over. If an UAS is simulated, the GS node also spawns the UAV nodes and autopilot bridges.

The SM is built by using the SMACH library within ROS. Coordination of the tasks of each UAS is achieved by combining a sort of states corresponding to basic motion primitives such as take-off, behaviours like following another UAV and more complex state machine structures such as the concurrence of various UAVs following different paths synchronized to start at the same time. In addition to the "nominal" behaviour during the mission, the actions to be executed in case of emergency during any of the possible states are also included in the state machine.

On the other hand, the GS node creates a world object containing different geometrical elements of the scenario: not only obstacles or 3D models of the objects in the environment, but also "logical" objects related to the goals of the mission or other abstractions that could be useful to monitor the execution of the mission. Those geometries are implemented with geometrical distributions of positions referenced to accessible Free Space Poses (FSP) or obstacles of variable shape. This is a more straightforward way to reference poses related to zones or geometries instead of using the global reference frame.

B. Aerial Segment Nodes

Regarding the nodes running on the computer on-board the UAV, the main one is the UAV master node, which manages its state and motion, receiving information from the on-board sensors nodes as well as communicating with the UAV Abstraction Layer (UAL) to receive the autopilot state and to send navigation commands. The core node component of this node receives the same name.

The UAV master node is also governed by a state machine, which receives mission commands from the GS node and splits them into single actions according to the current mission status. It listens and manages data from all its neighbour UAS. The capabilities of this module are distributed into four node components: Manager, Navigation Algorithm Interface (NAI), Data and Configuration. These components will be detailed in Sect. V.

The type of implementation for each UAS can be chosen between Software-in-the-loop (SIL), Hardware-in-the-loop (HIL) or real flight, so this node can be running on-board the

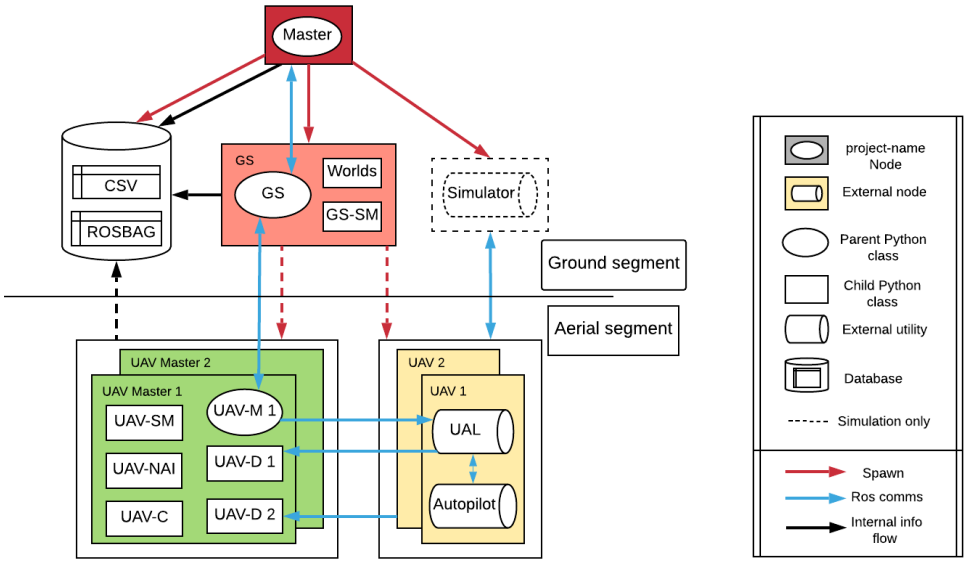


Fig. 1. General framework divided into the ground segment, and the aerial nodes designed to run on-board the UAVs although they can be also be executed on ground if the UAV is simulated. On one hand, the ground segment presented in Sect. III, is composed of the master and ground station nodes, the data storage folders and, in case it would be required, a simulator. On the other hand, the autopilot, UAL and the UAV nodes constitute the aerial segment, exposed in Sect. V. The red arrows represent the generation of a new architecture component following a parent-child tree architecture. Any kind of communication using the ROS network is represented by blue arrows. Black arrows correspond to the rest of existing data exchange. The simulation possibilities are contemplated with striped lines, which highlight the components of the architecture which would be used or not depending on the simulation level of each UAS.

UAV or on a ground computer. On the other hand, different types of autopilots are supported. For all these reasons, different parameters should be configured before starting the simulation/flights.

Along with the UAV master node, auxiliary external nodes such as the UAL and the autopilot nodes are also present onboard. The level of simulation required defines which of them are launched on-board or on the ground segment.

III. GROUND SEGMENT NODES

The nodes running on the Ground Control Station (GCS) are described in this section except the world geometry generation tool which is in the next section for the sake of clarity.

A. Master Node

The master node is a single component that serves as a front-end where the user selects the global parameters that define such as the name of the world and mission. In addition, different visualisation tools such as the Gazebo simulator client, RViz and the SMACH viewer can be activated by the user. The master node creates the storage folders where all the data gathered during a batch of missions will be saved. Hence, the identifiers of the dataset and identifier of the mission to be started within that dataset are also requested to the user. In the initialization process, the master node always spawns the GS node and then waits until the GS node declares the mission finished and reports success or failure and the critical events that may have happened during the execution. Finally, the master node purges any remaining opened node and stores the mission termination information in the dataset.

B. Ground Station Node

Two different components are endowed in this node. The central component controls the node, manages communication and information, and provides functions to implement different behaviours. A second component implements the state machine (SM) using the utilities offered by the central one.

The core node component retrieves information from the global parameters and the mission description file, where the user has predefined configuration fields such as the UAS model or the type of implementation for each UAS: Software-in-the-loop (SIL), Hardware-in-the-loop (HIL) or real flight. The world is created as part of this node and remains accessible for future interactions. The SM is built and executed while the core node component remains listening to information from the UAS (its state, critical events, etc.), which is logged in the mission folder using the rosbag mechanism for debriefing purposes.

Each task accomplished by the GS node is referenced to a unique state. Those tasks would entail an internal flow of information or communications exchanged with any of the UAS. Given that diversification, two SMACH state types are employed to accomplish each of them. For the duties that do not require ROS communications, CallBack States (CBS) are employed, which only deploy a standalone execution. Some examples of CBS are:

- New world. Execution of the world modelling tool to

create the virtual 3D environment and let it available for later access.

- Spawn UAS. Several spawn features such as the identification number, the location in the simulation or model previously defined in the mission SM are now employed to launch and activate the corresponding nodes and its control.
- Waiting. A period of idle time or waiting for user interaction via the keyboard.

In contrast, in order to implement states that require communication with the aerial segment nodes, Simple Action States (SAS) are employed. SAS fulfil a goal message with the objectives that the target UAS must accomplish and send it as a ROS action. After the UAS has performed the action, it sends back a response that is finally analysed by the SAS. Consequently, for each possible SAS action sent from the GS, there exists another state on the UAS SM that acts as a server. The main examples of SAS are:

- Take-off & Land.
- Basic Move. A movement in a single axis with a position or velocity setpoint.
- Follow Path. A list of static waypoints is followed based on a given algorithm which can control the UAV in position or velocity.
- Follow UAS at a distance. The UAV pursuits a dynamic goal until the target bounding sphere of a defined radius is reached.
- Follow UAS at a position. The goal pose is translated to a defined relative position from the followed UAS.
- Store data. The information retrieved during the simulation is saved in CSV format on disk.

The GS node state machine is a child node component which specifies how a mission should be executed taking as input the mission file described in JSON format. Then, this node component acts as a translator from JSON to the SMACH tool mapping also the required functions on each state. Every state callback is defined and grouped into state types. Those callbacks are employed while assembling them into nested structures such as simple state machines, concurrences or sequences. The outcome mapping must be according to the possible outcomes of each state, and it is directly retrieved from the mission definition file. In Fig. 2, a sequence state machine nesting the concurrence of three UAS following waypoints for their coordination is shown.

IV. WORLD MODELLING

The third module implemented on the GS node accomplishes the task of creating the virtual 3D environment to monitor the execution of the mission. Every geometrical element involved in the mission apart from UAS is generated: space segmentation, real or simulated objects and available positions of the space that can be accessed for different purposes called Free Space Poses (FSP). Figure 3 depicts the hierarchy where every new nested lower level offers definition independence from its upward grandparent. Provided that, for instance, a geometry would not need to be referred to the world frame, but only to its associated volume.

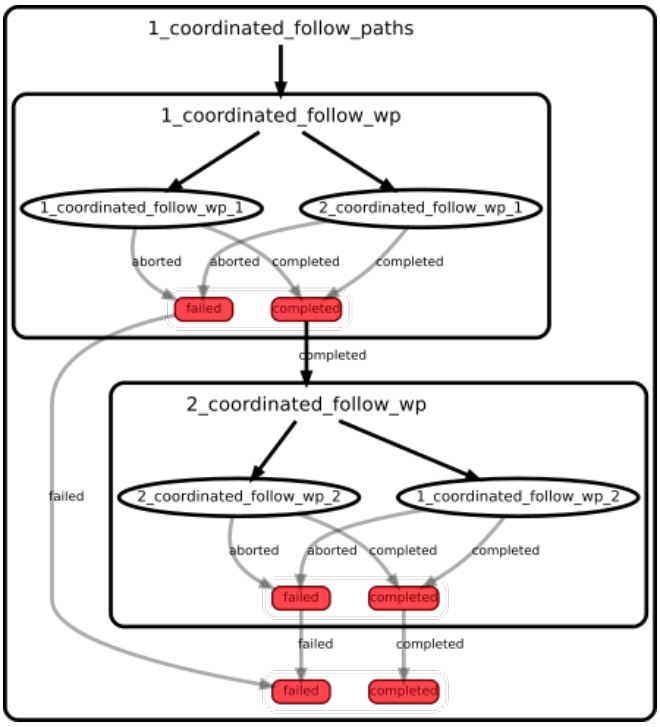


Fig. 2. The state machine for the coordination of two UAS following waypoints concurrently taken from SMACH Viewer. The outer box encloses a Sequence SM that includes two nested SM in a row defined by the "completed" outcome. Each of those nested SM correspond to the n-th waypoint of the path and is built out of a Concurrence SM. Inside each Concurrence SM, two states are activated at the same time. Each state corresponds to a different UAS and are of Follow Path SAS type.

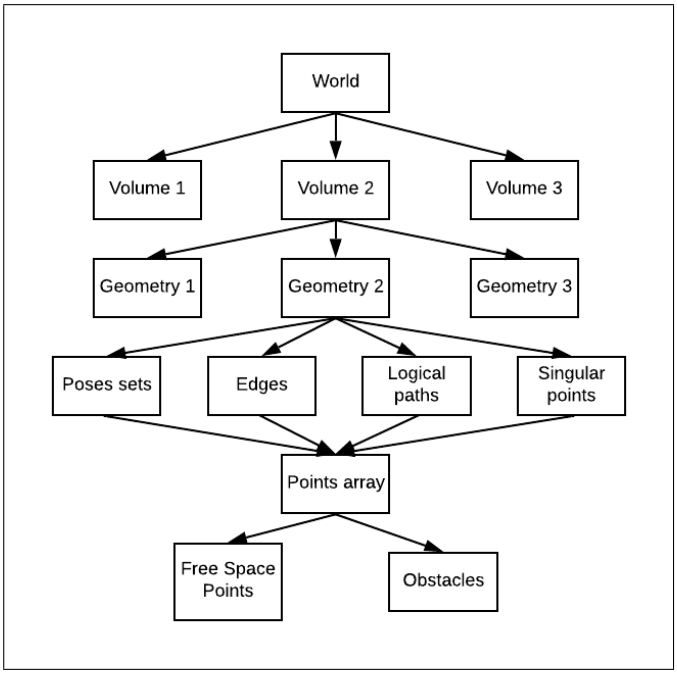


Fig. 3. World element tree modelling architecture. Each of the elements of a level would contain several of the elements of the lower level. A single world is the unique, global reference frame. Several volumes segment the scenario as parents of nested child groups of geometries that compose the shape to be defined. Geometries would be provided of different shape types and enclosure a sort of logically located poses that may be used to create obstacles or free space poses provided to poses of a path on the mission.

For any new element on each level, there exists a new world component so every parent would contain any number of objects of every type of child. A JSON file describes the whole world architecture and the characteristics of every element. The top world component retrieves that definition and splits it into the definition of the volumes. On the same way, each parent element splits that definition on the corresponding for itself and every child and provides it on its creation. Every world component provides functions to retrieve information from itself and its child objects in order to make every element accessible from the GS node.

A. Volume

The first level under the world frame is the volume, whose function is to gather various geometries around a single frame so all of them would be relocated without redefining the volume construction, as long as giving some standard features to all of them.

B. Geometry

Inside every volume, geometries with different shapes are used to model different restrictions within the scenario. The basic primitives included are a cube, a sphere, a cylinder and a prism.

Specific world components concerning the shape inherit a primary component for common utilities of a general geometry. The general geometry component manages standard features such as dimensions, name or origin. It also provides core functionalities for the transmission of data and the auxiliary generation of geometric structures that are later specifically employed by each shape. A list of the main functionalities is provided in Table I.

TABLE I

THE GENERIC INHERITED COMPONENT OFFERS CORE FUNCTIONALITIES FOR THE TRANSMISSION OF DATA AND THE AUXILIARY GENERATION OF GEOMETRIC STRUCTURES. SPECIFIC COMPONENTS EXTEND FUNCTIONALITIES DEPENDING ON THE SHAPE.

Function	**Provider world component**
Raw random values Matrix	Generic
Generate Obstacle/FSP from Matrix/List/Coords	Generic
Generate Poses sets Matrix	Generic
Generate Random Poses	Generic
Generate Random Dimensional Values	Generic
Generate Path	Generic
Lines intersection	Generic
Polygon-lines intersection	Generic
Zigzag from matrix poses	Generic
Get Obstacles	Generic
Get FS Global Pose from Matrix/List/Path/Coo	Generic
Make/Erase RViz Marker	Generic
Poses Matrix	Specific
Random Poses	Specific
Perimeter Poses	Specific
Edges Poses	Specific

C. Pose arrays

Geometries provide groups of 3D poses gathered into arrays. Those would be extracted from features of the shape, such its edges or 3D matrices that fulfill the interior of the shape along certain axes. Besides, other point arrays would be independent of the shape as it would be the coordinates from the Geometry origin. In addition, both kind of poses may be used to generate geometric elements that produce new pose arrays, such as intersections.

All those poses would be later used to define obstacles or FSP on it. Besides, poses arrays are also employed as supporting structures, that combine with each other with the purpose of constructing or finding more complex geometrical compositions. An example of it would be the use of a matrix of poses generated by a Generic Geometry function in addition to the perimeter of a prism provided by a Specific Geometry function. As a result, only the poses of the matrix enclosed by the perimeter would be extracted.

D. FSP and obstacles

The Free Space Pose is an available position of the space that can be accessed for different purposes. A single world component is the interface to operate with it. A transform is employed to be able to refer the FSP for its parent frame and the global frame.

More complex is the obstacle component, which is focused on managing the information of a single obstacle and spawning it where required. Thus, everything concerned with that obstacle deals with this world component or with information generated by it. Also using a transform, RViz marker and gazebo model would be spawned on its corresponding place and orientation. Offered shapes of obstacles in Gazebo are cubes, spheres and cylinders and their dimensions are fully customizable. Some functions are also required to deal with the information of the obstacle as the removal of the model or its global position computation.

E. Auxiliary world components

Transformation broadcasters, RViz markers and RViz polygon arrays are used for world components at every level of the world hierarchy. They would be thought as the leaf nodes of the world hierarchy tree.

Transforms are used to cope with pose generation on the ROS network and are useful to reference them to other frames, not only for visualisation purposes. RViz markers and polygon arrays must define and dynamically manage the message of generation, update and removal that defines any shape that the world works with.

V. AERIAL SEGMENT NODES

In this section, the nodes running on-board the UAV are described. However, it should be noticed that the UAS can be also simulated or the UAV can be so light that it is not possible to put a computer on-board. In these cases, these nodes can be also launched on a ground computer.

The presented framework has a single node designed to manage the UAV. This node receives through ROS information from its on-board sensors as well as the state of other UAVs. It also interacts with the UAV Abstraction Layer (UAL) [12] to control UAVs with autopilots supporting the

MAVLink protocol, and/or autopilots by manufacturers such as DJI and Crazyflie. As a consequence, the UAL node must be also running on-board the UAV along with the autopilot driver. This section is focused on the UAV master node and the different node components employed: core management of the UAV, the state machine which coordinates the actions, the Navigation Algorithm Interface to implement different strategies, an object that deals with the data corresponding to each UAS and another which controls the configuration of the communication and other features.

A. UAV Manager

This core class initiates the node and creates the required instances of the rest of classes. A single UAV state machine and a single UAV Navigation Algorithm Interface are generated. Functions are offered to accomplish the different roles, which are similar to those explained in Sect. III-B and selected as SimpleActionNodes.

The data from the GS is retrieved and mapped to internal variables. Any auxiliary function is called if needed to transform the goal into useful data such as translation to ROS variables, the desired path to be smoothed sent to the corresponding nodes or translation of the goal from the target UAV in a "Follow UAV At Position" behaviour. The NAI provides the reference for the velocity based on the algorithm chosen. Once the goal and the strategy to arrive are appropriate, the UAL is used to give the next command to the autopilot.

Before the state machine is executed, possible preemption messages are listened from the GS node. The communication is based on states, but some of them are automatically started when internal features are changed such as the desired goal and followed path sent to the RViz viewer or the state update to the GS node. The most important of them is the an evaluator, whose duty is to check if any collision, GS node notification or battery warning has occurred. In case any critical event is detected, the state is changed, and the GS node informed. It also gathers all the data for mission debriefing purposes.

B. UAV State Machine

How all the possible events fit in a timeline is the problem solved by this class. A mission is an ordered set of the already known actions performed by each of the UAS, coordinated by the GS state machine defined by the user in the JSON mission file. Since an UAV can perform only one task at a time, the current state must be assessed to let new commands from the GS affect just in case the UAV has finished the previous behaviour.

Every UAS has the same state machine since all of them are expected to execute the standard movements, behaviours and management tasks. Every state callback is available since the beginning to be requested. Hence, the basic structure can be seen as a star-shaped state machine with a central static state as the core and different Simple Action Server states located on each corner. SMACH wrappers are employed to deal with the Action Servers.

On the central state, every action is offered through ROS, and when it is being performed, only a preemption notification would be accepted from the state machine. Once the action has been accomplished, and the response sent, the state machine drives back to the central state.

Available wrapped SimpleActionStates correspond to the ones referred to in the GS node section. When the GS node has requested a specified action, the goal is translated to comprehensive variables, and the corresponding UAV master function is executed.

C. UAV Navigation Algorithm Interface

The Navigation Algorithm Interface (NAI) module provides a powerful tool to the researcher as it allows to easily test new navigation algorithms. In addition, it endows some already built-in modules as a reference, such as a simple greedy guidance algorithm, the Optimal Reciprocal Collision Avoidance (ORCA) algorithm [16] and an interface to TensorFlow [17] sessions. The simple greedy guidance calculates the distance and direction to the target from the current position and applies the desired speed to compute the velocity vector. ORCA receives the current position and velocity of every UAV on the scenario as well as the own desired velocity to provide the optimal velocity free of collisions by implementing a velocity field. To work with TensorFlow's machine learning techniques, a function uploads a pre-trained graph and start a session to evaluate the current conditions of the scenario as inputs, and a neural network returns the optimal velocity to apply as output.

As it was mentioned above, the NAI module is designed to be an interface to any new algorithm to be tested. On the scope of this class, all the information about the UAS and its environment is continuously available during the mission as input to the algorithm to be tested. In addition, various outputs can be mapped into control signals to the UAL apart from the velocity. For instance, the selection of adjacent FSP as next target optimizing any cost function is straightforward.

The NAI class can offer its maximum potential when different algorithms are employed at the same time to optimize various steps of the decision-making chain. The fact that all of them would work inside the same data structure would make it easier to concatenate different task solver algorithms uniformly.

Other auxiliary functions are offered to make easier the implementation of the algorithms. A hovering function returns a zero value velocity message. Saturation upper and lower strains a value provided the limits. A neighbor selector function calculates the distances to all the UAVs and static obstacles in the environment and provides a sorted list with the identification of the nearest along with the distance.

D. UAV Data and Configuration

Each of the objects created from these classes are focused on a single UAV. Them, the UAV manager needs a list of the available UAVs on the scene with their associated relevant information.

TABLE II
MAIN FEATURES OF THE UAV MANAGED BY THE UAV DATA PYTHON CLASS.

Feature	Description
UAL state	Codification about landed, armed or flying among others.
Pose	Current stamped local, Cartesian pose.
Velocity	Current stamped Cartesian twist.
Battery level	Percentage of remaining battery.
GS critical event/preempt command	Command from GS node to finish current behaviour.
Path smoothing velocity	Optimal velocity according to the path smoother nodes.
Sensor driver comms	For instance, depth camera pixel info.

The UAV Data is mostly passive: it gets data which is generated outside the ROS network. That information is processed and translated to the UAV data so the UAV manager can get access to it. It would also be used to treat that data and republish it to, for instance, improve its visualization. The main information that UAV Data copes with is listed in Table II.

In an ideal situation, a fully standardised interface as UAL would be enough to deal with each model and autopilot and its singularities. Nonetheless, it is a better option to rely on an intermediate bridge more customizable inside this framework.

UAV Configuration takes the information about the UAL selected on the mission definition and extracts all the model information from a specific JSON file. After combining both information sources, particular features as the security radius, maximum speed or implementation (SIL, HIL or real) are mapped to internal values of the UAV master node.

VI. SIMULATIONS AND EXPERIMENTS

Simulations and real experiments which show the main features of the framework are described in this section. Several videos of them are available at `https://grvc.us.es/icuas19multi`: For each setup, two videos (full mission and edited) are provided with a recording of the simulation/reality, RViz visualization and state machine transitions.

A. Setup I: Battery level safety with collision-avoidance

The first setup involves two PX4 Iris UAS simulated software-in-the-loop in Gazebo. The role of the first UAS is to follow a segmented delivery path whereas the task of the second one is to film the first UAV at different relative positions during the mission. Elapsed in periods of 15 seconds, the second UAS tries to follow at 3 meters on the x-axis, at 3 meters on the z-axis and at 2 meters on the y and x-axes. Given the fact that both of them traverse a volume with obstacles, the collision avoidance algorithm ORCA is employed continuously. The smoothing path nodes are used to select the desired velocity input for ORCA and a SMACH state machine is designed to perform correctly the tasks explained as well as to manage any critical event occurred.

The first segment of the path followed by the first UAS is collision-free and the second one achieves the different relative positions successfully. During the execution of the second segment, the first UAV runs out of battery. Hence, an adjacent recharge state machine is activated inside the path following state. That recharging state machine consists on a return to home, land, wait for the pilot to notify charge completed, take-off and return to the prior state. During the process, the first UAV is still performing its filming task at the last relative position.

Figure 4 shows a screenshot taken during the mission. The situation of successful execution and recharge landing are included along with the battery recharge nested state machine. Full videos can be found at `https://grvc.us.es/icuas19multi#SimulationI`.

B. Setup II: Collision safety

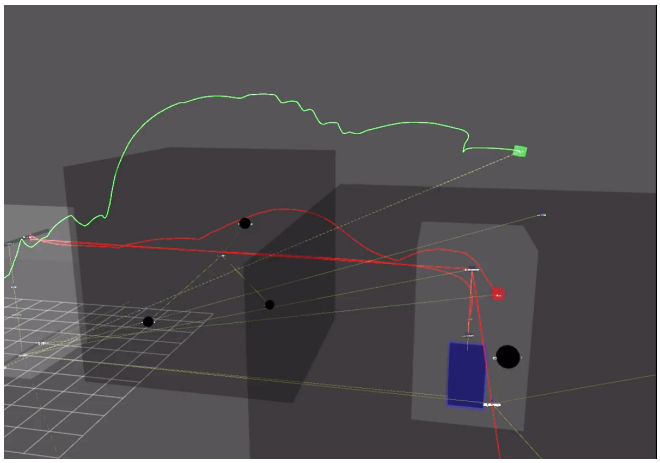
The simulation conditions are the same as well as the scenario. The role of the first UAS is also delivering. However, the second UAS should maintain a distance of two meters with respect to the first one disregarding its relative direction. The state machine used is the same, although other branches will be relevant in this setup. The main difference is that no algorithm to avoid collisions is employed. Thus, the first obstacle in the path is detected to collide in a short time with the first UAV, which stops its current state and sends to the GS node a notification of a critical event. The notification is resent to the other UAS involved in the mission. Both of them start hovering and, after a safety period for the pilots to take their control, they land.

Figure 5 shows a screenshot of the mission abort instant and its concerned state machine branch. Full videos of the simulation can be downloaded from `https://grvc.us.es/icuas19multi#SimulationII`.

C. Setup III: Multi-platform coordination

This larger scale experiment is designed to show the main features of this framework. First, platform agnosticism is achieved thanks to the connection to the UAL. A PX4 (drone 1) and two DJI UAS (drones 2 and 3) are employed. Then, different levels of simulation are used: Drone 1 is SIL simulated with Gazebo, drone 2 is real with every dedicated node running on an Intel NUC on-board a DJI F550 frame and drone 3 is another DJI whose dynamics are simulated on a computer but a real autopilot device is used HIL. Finally, the potential of the World module is applied to build a solar photovoltaic plant and locate panels on a sorted position beneath an inspection zig-zag path.

The photovoltaic plant Cañamero, in Caceres (Spain) has been modelled. The western group of three zones of panels has been extracted to allocate each zone to an available UAS. Each panel zone is applied to a volume of the World module. Inside each volume, three different prismatic geometries are used to define the panel height, the inspection altitude and



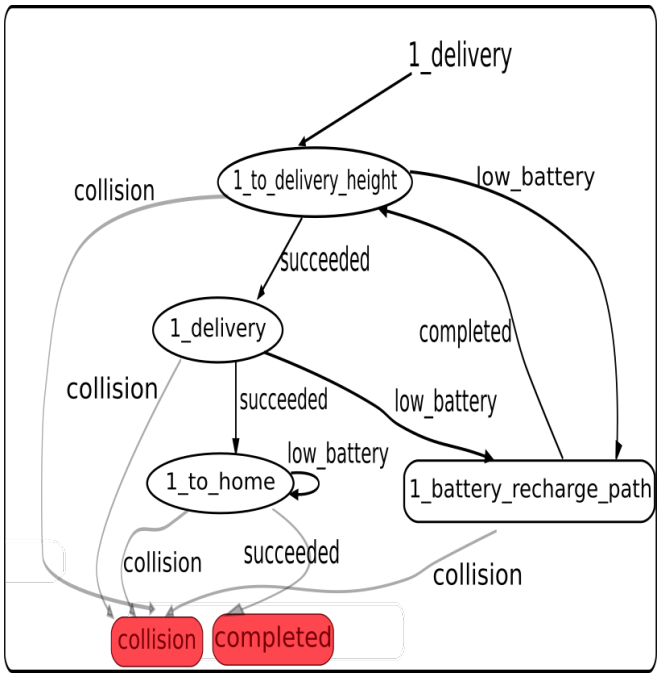
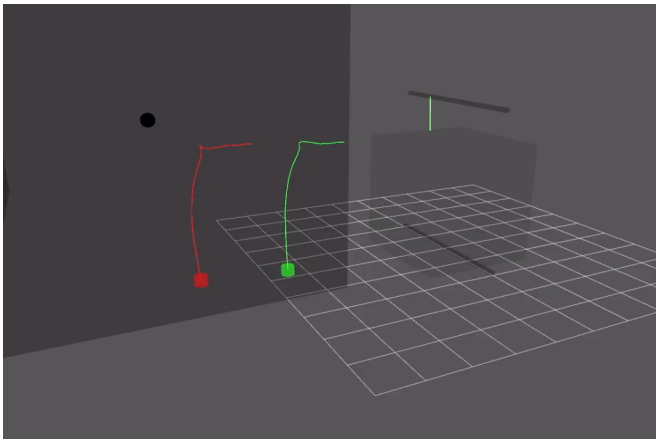


Fig. 4. Setup I screenshot and state machine. The top image presents an RViz visualization of both UAVs crossing an obstacle volume. The first UAV follows a delivery path, whereas the second one films it at 3 meters above. The ORCA algorithm assists both UAVs for obstacle avoidance. The bottom image shows a safety state machine to deal with the possibility of an UAV in a low battery state. The state machine that implements the delivery path of the first UAV is composed of altitude checking, delivery path and return to home. For each of those states, independently from the actions of the rest of UAS, if the UAV receives a low battery warning, it enters in the state machine "battery recharge path". That state machine is not shown for the sake of clarity and it is composed by return to home, land and wait for the user to notify the full recharge. Finally, a take-off is commanded, and the mission continues, exiting "battery recharge path" and coming back to the "delivery state".



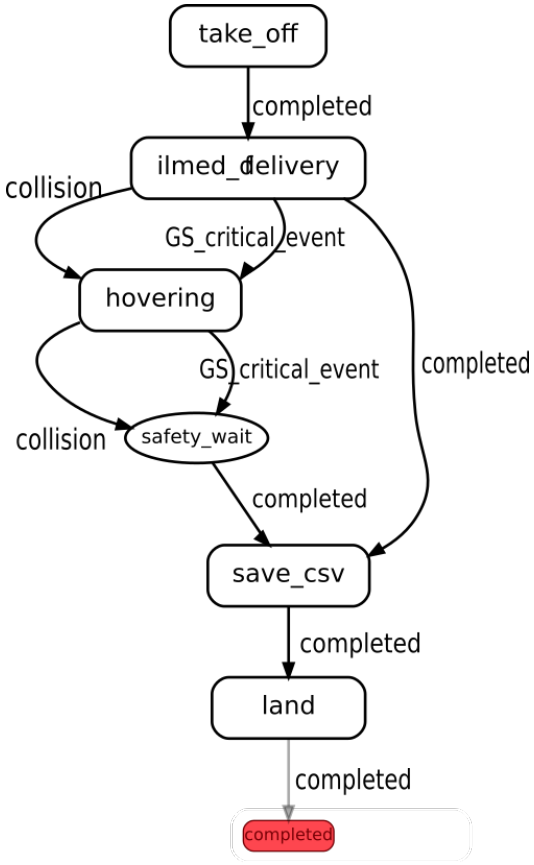


Fig. 5. Setup II screenshot and state machine. The top image shows a RViz visualization of the UAVs without collision avoidance algorithm navigating towards an obstacle. The first one (UAS 1) follows a path and the second UAV (UAS 2) pursuits the first one at 2 meters when an imminent collision is detected by UAS 1 and an alert is sent to the GS node which broadcasts it. Both UAVs hover and safely land. The bottom image presents the state machine for the safety collision detection. Both UAS were into the normal mission state. When UAS 1 detects the collision and UAS 2 receives the alert, that state outcomes a collision and a concurrence state machine makes them both hover by a Basic Move SAS. A safety wait CBS lets the pilots take control over the drones. In case it does not happen, another two concurrence state machines with SAS let the UAVs store the data and safely land.

the approximation altitude. Panel height geometry is fulfilled with a pose set built from a matrix and its poses included into a polygon. The inspection altitude geometry is used to construct a zigzag with the intersections of parallel lines with the perimeter. A different approximation altitude is used for each drone. Drones 1 and 3 employ a path follower node to smooth their trajectory (velocity control), whereas drone 2 directly uses position commands.

At the beginning of the mission, every UAS takes off at the same time and follows its own path independently. The state machine counts with both safety branches described on the first two setups.

Figure 6 shows a photo taken during the flight along with the visualization of the constructed virtual scenario in RViz. Videos of the full setup are available at `https://grvc.us.es/icuas19multi#ExperimentI`.

VII. CONCLUSIONS AND FUTURE WORK

The implementation of the framework along with several built-in use examples can be found in the GitHub repository `https://github.com/JoseAndresMR/ros_magna` under the MIT License. The framework allows to implement different cooperative strategies while maintaining group and individual safety thanks to the different state machines that can be implemented. The interface on the UAS on-board node has been tested with the implementation of different navigation algorithms. In addition, different virtual worlds has been modelled and used for the simulations and experiments.

Future work covers the full range of presented modules since it is on going work. It would be interesting to implement also a real-time teleoperation interface within the framework as well as a graphical user interface for the definition of the mission global parameters and real-time monitoring. Regarding the world modelling, more shapes and geometrical elements as well as the possibility to dynamic transform them during the execution of the mission are considered as next steps.

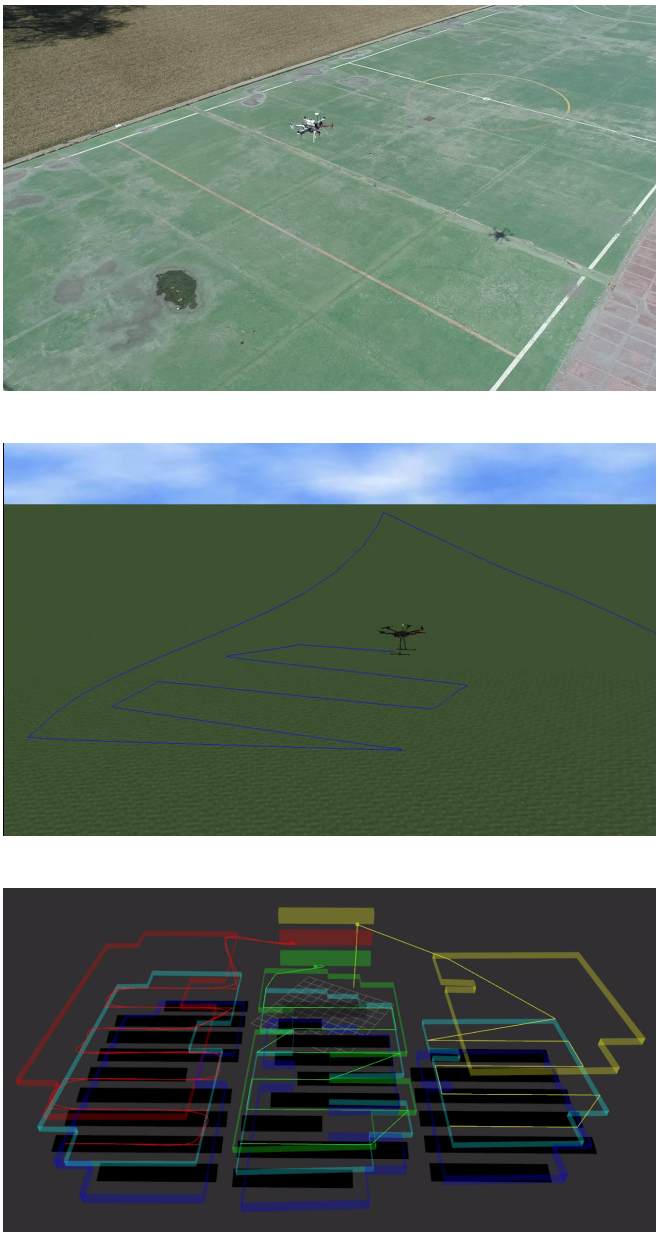


Fig. 6. Setup III with different autopilots and simulation levels. The top image is a photograph of the DJI F550 real UAS (drone 2) during the flight. At the central image, a screenshot of the DJI Assistant 2 simulator of the HIL drone 3 is shown. The bottom image is an RViz visualisation of the scaled modelled Cañamero photovoltaic plant: Three panel volumes within its panels zone with obstacles and two height prisms used. The blue prism corresponds to the panels, the cyan prism represents inspection altitude, and the rest of the coloured prisms correspond to the security altitude of the UAS of the same colour. The UAS are approximating to their corresponding volumes and their target zigzag paths of inspection are also rendered.

REFERENCES

[1] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[2] P. Kremer, J. Dentler, S. Kannan, and H. Voos, "Cooperative localization of unmanned aerial vehicles in ros - the atlas node," in *Proceedings - 2017 IEEE 15th International Conference on Industrial Informatics, INDIN 2017*, 2017, pp. 319–325.

[3] B. H. . Lee, J. R. Morrison, and R. Sharma, "Multi-uav control testbed for persistent uav presence: Ros gps waypoint tracking package and centralized task allocation capability," in *2017 International Conference on Unmanned Aircraft Systems, ICUAS 2017*, 2017, pp. 1742–1750.

[4] J. Dentler, S. Kannan, M. A. Olivares-Mendez, and H. Voos, "Implementation and validation of an event-based real-time nonlinear model predictive control framework with ros interface for single and multi-robot systems," in *1st Annual IEEE Conference on Control Technology and Applications, CCTA 2017*, vol. 2017-January, 2017, pp. 1000–1006.

[5] V. Grabe, M. Riedel, H. H. Bulthoff, P. R. Giordano, and A. Franchi, "The telekyb framework for a modular and extendible ros-based quadrotor control," in *2013 European Conference on Mobile Robots, ECMR 2013 - Conference Proceedings*, 2013, pp. 19–25.

[6] Y. Yu, X. Wang, Z. Zhong, and Y. Zhang, "Ros-based uav control using hand gesture recognition," in *Proceedings of the 29th Chinese Control and Decision Conference, CCDC 2017*, 2017, pp. 6795–6799.

[7] J. J. Roldán, E. Peña-Tapia, D. Garzón-Ramos, J. deLeón, M. Garzón, J. delCerro, and A. Barrientos, *Multi-robot Systems, Virtual Reality and ROS: Developing a New Generation of Operator Interfaces*. Cham: Springer International Publishing, 2019, pp. 29–64. [Online]. Available: https://doi.org/10.1007/978-3-319-91590-6_2

[8] W. Meng, Y. Hu, J. Lin, F. Lin, and R. Teo, "Ros+unity: An efficient high-fidelity 3d multi-uav navigation and control simulator in gps-

denied environments," in *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, 2015, pp. 2562–2567.

[9] W. Hönig and N. Ayanian, *Flying Multiple UAVs Using ROS*. Cham: Springer International Publishing, 2017, pp. 83–118. [Online]. Available: https://doi.org/10.1007/978-3-319-54927-9_3

[10] H. Hayakawa, T. Azumi, A. Sakaguchi, and T. Ushio, "Ros-based support system for supervision of multiple uavs by a single operator," in *Proceedings - 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018*, 2018, pp. 341–342.

[11] A. P. Lamping, J. N. Ouwerkerk, N. O. Stockton, K. Cohen, M. Kumar, and D. W. Casbeer, "Flymaster: Multi-uav control and supervision with ros," in *2018 Aviation Technology, Integration, and Operations Conference*, 2018.

[12] F. Real, A. Torres-Gonzalez, P. Ramon-Soria, J. Capitan, and A. Ollero, "UAL: An abstraction layer for unmanned aerial vehicles," in *Proceedings of the 2nd International Symposium on Aerial Robotics*, Philadelphia, USA, June 2018.

[13] "SMACH - a task-level architecture for rapidly creating complex robot behavior in ROS," 2019, http://wiki.ros.org/smach, Last accessed on 2019-02-26.

[14] "RViz - 3D visualization tool for ROS," 2019, http://wiki.ros.org/rviz, Last accessed on 2019-02-26.

[15] "SMACH Viewer - a GUI that shows the state of hierarchical SMACH state machines in ROS," 2019, http://wiki.ros.org/smach_viewer, Last accessed on 2019-02-26.

[16] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–19.

[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/