# A Multi-Layered Component-Based Approach for the Development of Aerial Robotic Systems: The Aerostack Framework

**Jose Luis Sanchez-Lopez · Martin Molina · Hriday Bavle · Carlos Sampedro · Ramón A. Suárez Fernández · Pascual Campoy**

**Abstract** To achieve fully autonomous operation for Unmanned Aerial Systems (UAS) it is necessary to integrate multiple and heterogeneous technical solutions (e.g., control-based methods, computer vision methods, automated planning, coordination algorithms, etc.). The combination of such methods in an operational system is a technical challenge that requires efficient architectural solutions. In a robotic engineering context, where productivity is important, it is also important to minimize the effort for the development of new systems. As a response to these needs, this paper presents Aerostack, an open-source software framework for the development of aerial robotic systems. This framework facilitates the creation of UAS by providing a set of reusable components specialized in functional tasks of aerial robotics (trajectory planning, self localization, etc.) together with an integration method in a multi-layered cognitive architecture based on five layers: reactive, executive, deliberative, reflective and social. Compared to other software frameworks for UAS, Aerostack can provide higher degrees of autonomy and it is more versatile to be applied to different types of hardware (aerial platforms and sensors) and different types of missions (e.g. multi robot swarm systems). Aerostack has been validated during four years (since February 2013) by its successful use on many research projects, international competitions and public exhibitions. As a representative example of system development, this paper also presents how Aerostack was used to develop a system for a (fictional) fully autonomous indoors search and rescue mission.

**Keywords** Aerial robotics · Robot architecture · Autonomous behavior · Distributed robot systems · Multi-robot coordination · Quadrotor · Mobile robots · Remotely operated vehicles · MAV

## 1 Introduction

A high degree of autonomous operation of Unmanned Aerial Systems (UAS), is important to simplify their use by human operators and to increase the safety

of flights in dynamic environments. To provide fully autonomous operation, the research community in aerial robotics is proposing solutions for a number of specific problems such as localization and mapping on unstructured and dynamic environments, precise control of the aircraft with collision avoidance, trajectory and mission planning with a high level of cognition and intelligence, human-robot interaction, robot-robot interaction, safety and fault tolerance among others.

However, the development of a completely integrated solution for full autonomy requires to combine in a single architecture a number of highly interrelated and specialized building blocks. This integration creates new challenges (e.g., efficient multi-tasking execution, adaptability to be used in different problems, scalability, etc.).

A number of commercial and open-projects exist that aim to develop complete software architectures (see [11] for a complete survey). To the best of the authors' knowledge, the most well-known open-source commercial oriented projects are the "PX4 Flight Stack", and the "APM Flight Stack". These projects aim to create a full software stack that provide the UAS a high level of autonomy including for example controllers to command the drone giving a predefined set of waypoints, a state estimator that fuses the measurements given by different sensors to provide an estimation of the full state of the UAS (pose, velocity, acceleration), and a reactive obstacle avoidance that generates the appropriate motion commands to evade obstacles perceived. Nevertheless, despite of all these available components, the user is required to be in the loop while the UAS is performing a mission, because he or she has to take control of the UAS in case of unexpected changes of the environment, having to constantly monitor the mission. In addition, the UAS has limited interaction capabilities with the environment and with the users.

On the other hand, the activity of several research groups has produced some open-source research oriented architecture frameworks for UAS, being the most relevant ones, up to the authors' knowledge:

- "asctec_mav_framework", developed by ASL - ETHZ, has a special focus on autonomous navigation of Ascending Technologies Aircrafts and it is not compatible with any other aircraft platforms.
- "hector_quadrotor" [10] framework, developed by HECTOR - TU Darmstadt, focused on heterogeneous cooperation for search and rescue (SAR) tasks.
- "telekyb" [8] framework, developed by HRI - MPI. It allows the fully autonomous multi-UAS navigation. Although it is very powerful, the main drawback is its rigid architecture that even allowing to exchange for modules with similar functionalities depending on the user's need, it does not allow the user to easily change the architecture design for new capabilities.
- "Paparazzi" [3] project, developed and used by ENAC and MAVLAB - TUDelft. This project includes not only the software framework but also the hardware autopilot and sensors and it is not compatible with any other commercial hardware.
- "Twirre" [12] architecture, developed by NHL Computer Vision proposes a hardware and software design. It is focused mainly on hardware and it does not report a high level of autonomy.

Even though this line of research has produced important advances, the referred work shows that there are important remaining challenges related to: (1) level of autonomy, i.e. more complex hybrid architectures able to provide more degree of autonomy and (2) versatility, i.e. more versatile integrated solutions able to be used for different applications and physical aerial platforms.

In order to fulfill these needs, this paper shows the authors' recent progress and main results in this line of research. In [27], Aerostack[1] (a software framework for AErial RObotics) was firstly introduced. The authors named Aerostack to a framework that integrates and consolidates results of four years (since February 2013) of research work on both research of components for aerial systems and their integration with tests in efficient architectures [25, 29]. As a result, Aerostack was created as a more mature, robust, reliable, documented, tested and validated software framework.

The advantages of Aerostack with respect to other architectures and frameworks is twofold: (1) a complete multi-layered architectural organization to support fully autonomous flights and (2) a versatile

software framework for developing and integrating new software components.

The multi-layered architecture includes both low-level layers for reactive behavior and high-level layers for intelligent behaviors. At the low-level, Aerostack provides a number of specialized reusable components for visual perception, motion controllers, etc. At the high-level, Aerostack includes a number of components to provide a high degree of autonomy and self-adaptation in complex and dynamic environments with fault management procedures to increase the degree of safety.

The versatility of Aerostack is based on the following two main features. On the one hand, it is flexible enough for a wide range of applications from teleoperated flights of single UAS to highly autonomous missions of multi-robot UAS platforms. On the other hand, Aerostack is hardware-independent. It is focused on software development that is designed to work as part of the operative system, which means that it requires the appropriate hardware design as well as its proper firmware and middle-ware software components.

The main contribution of this paper is to describe Aerostack in depth, providing details of every subsystem of its architecture together with implementation examples that illustrate the general concepts. This papers shows as well a complex mission where a set of UAS navigate in a search and rescue mission in a complex environment, with emergent cooperation between the UAS and looking for a target. This mission allows to demonstrate the full level of autonomy achieved by Aerostack, with emergent cooperation, and adaptation to changing environments.

The remainder of the paper is organized as follows: Section 2 presents the key aspects of Aerostack framework and its architecture. Section 3 describes in detail the subsystems of Aerostack. In Section 4 a full mission demonstrates the capabilities and performance of Aerostack. Finally, Section 5 concludes the paper and points out some lines of future work.

## 2 Aerostack Framework

This section describes how the Aerostack framework is organized in a multi-layered architecture. The section also describes the library of reusable components provided by Aerostack and how the library can be used to build a new architecture for a specific UAS.

### 2.1 The Multi-Layered Model

An important contribution of Aerostack is a solution to organize robotic software components in an efficient architecture. For this purpose, we have defined a multi-layered organization model as an architectural pattern with subsystems and functional components.

The multi-layered organization model follows the hybrid reactive/deliberative paradigm, i.e., an architecture that integrates both a deliberative and reactive approaches [1] and [14]. The presented design (represented in Fig. 1) includes five layers: reactive, executive, deliberative, reflective and social.

The first three layers correspond to the popular hybrid design known as the three layer architecture [7] and [23]: (1) *reactive layer* with low-level control with sensor-action loops; (2) *executive layer* (or sequencing layer) that accepts symbolic actions from the deliberative layer and generates detailed behavior sequences for the reactive layer; this layer also integrates the sensor information into an internal state representation; and (3) the *deliberative layer* generates global solutions to complex tasks using planning (e.g., planning optimal trajectories). The reactive layer functions in the present while the deliberative layer uses information from the past and projection to the future.

The reactive layer is a sensor-action loop that includes feature extractors (in the feature extraction system) and motion controllers (in the motor system). Feature extractors may read simple states of sensors or may implement complex computer vision and pattern recognition algorithms (signal processing, recognition of objects and basic relationships). Motion controllers typically implement combinations of Proportional-Integral-Derivative (PID) controllers (e.g., cascade controllers). For example, these type of controllers can accept orders about a desired value for a variable (position, speed, altitude, and yaw) in form of single commands or simultaneous commands that are translated into low level commands to be sent to actuators.

To increase the degree of autonomy of robots, Aerostack includes a *reflective layer* based on cognitive architectures [2, 4, 30, 31] to simulate certain self-awareness able to supervise the other layers. The reflective layer helps to see if the robot is actually

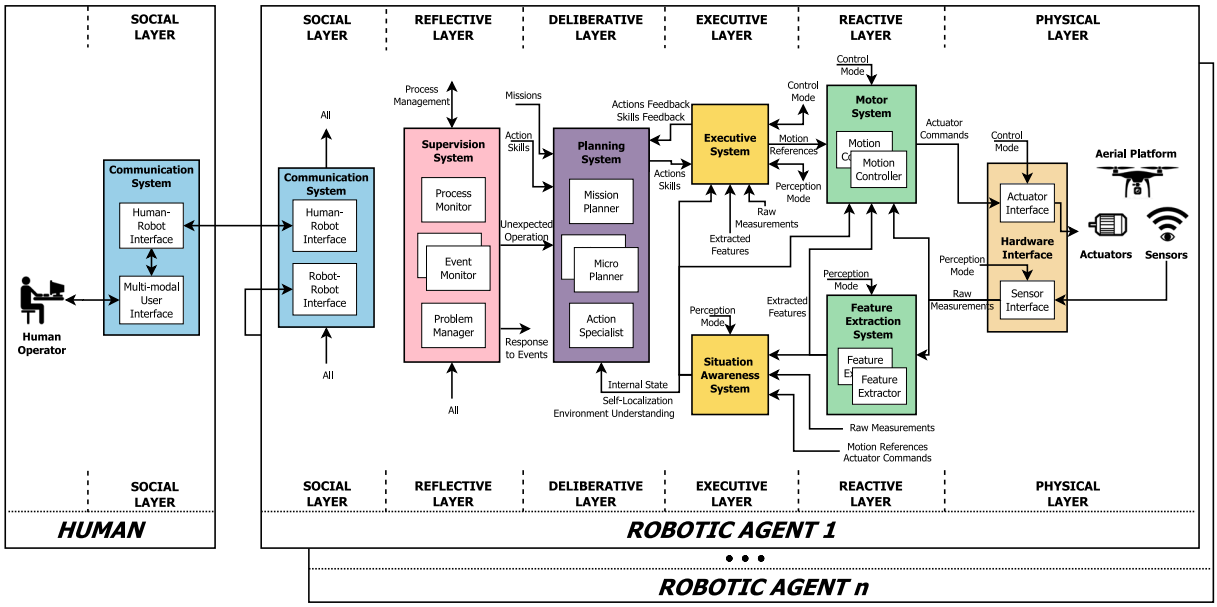



**Fig. 1** Main components of the multi-layered architecture of Aerostack. The architecture is formed by $n$ heterogeneous robotics agents and the human operators. Every robotic agent shares the same layered architecture, although it can have different component implementations as well as different hardware. The architecture includes five layers: the social layer allows the robotic agents to communicate with the rest of agents. The reflective layer supervises the other layers to see if the robot is making progress to its goals and to react in the presence of problems. The deliberative layer generates global solutions to complex tasks using planning. The executive layer takes actions from the deliberative layer and generates detailed behavior sequences for the reactive layer; it additionally integrates the sensor information into an internal state representation. Finally, the reactive layer counts with low-level control with sensor-action loops

making progress to its goal and to react in the presence of problems (unexpected obstacles, faults, etc.) with recovery actions.

Aerostack includes also a *social layer* with communication abilities, as it is proposed in multiagent systems and other architectures with social coordination (e.g., [5]). In this level is important to establish an adequate communication with human operators and other robots.

Aerostack has another functional organization in seven subsystems. Their functional specification, together with a description of their inputs and outputs is done in an abstract level (thanks to a complete ontology), avoiding to describe algorithmic or implementation details. The seven proposed subsystems, deeply analyzed in Section 3, are the following: Feature Extraction System, Motor System, Situation Awareness System, Executive System, Planning System, Supervision System, and Communication System.

The architecture is also consistent with the usual components related to guidance, navigation and control of unmanned rotorcraft systems [9]. In particular, the Navigation System (NS) corresponds to our feature extraction system and situation awareness system, the Guidance System (GS) corresponds to our executive system, planning system and supervision system and, finally, the Flight Control System (FCS) corresponds to our motor system.

## 2.2 Features of Aerostack Framework

Concerning the behavior of UAS, the proposed multi-layered organization has the following characteristics:

– *A cognitive model to support autonomous behavior*. The multi-layered model is a solution that identifies and organizes the required cognitive processes to support autonomous behavior, integrating processes for perception, reactive control, deliberative reasoning, supervision, etc. The proposed model includes ideas from the state of the art in artificial intelligence for autonomous robotics such as hybrid paradigms

(reactive/deliberative), reflective behavior and social coordination.

– *Separate representations to facilitate communication*. The organization is separated in layers which is useful to have representations at different levels. This is important, for example, to offer a more natural interaction with operators using their language (e.g., with concepts such as mission goals, tasks, skills, etc.) instead of the low level technical jargon used for certain components (e.g., set points for controllers).
– *A functional division for efficient execution*. The architecture is divided into functional blocks, i.e., blocks that play a functional role. This facilitates the implementation of the architecture as a set of processes using a distributed network platform with different computers and a multitasking operating system. This is important to provide the required efficiency for real flights with real time constraints.

Considering a robotic engineering viewpoint, where productivity is important for rapid development and easy maintenance of UAS, we have proposed a multi-layered organization paying special attention to the following features and advantages:

– *Precise meaning of terms*. We have formulated the components of the architecture using common and generally accepted precise meanings from the state of the art of autonomous robotics. The use of standard meanings facilitates acceptance by general developers, understandability and, consequently, makes it easier the work for maintenance. Besides the names for architectural components, an ontology has been defined for aerial robotics, to be used as common terminology for component interoperability.
– *Uniform hierarchical organization*. We have organized the architecture according to homogeneous blocks at three abstract levels: layer, system and process. The homogeneity and hierarchical organization also facilitates understanding the complexity of the model.
– *Implementation independent*. The description is formulated defining the role of components, but independently from specific implementations. Thus, the model is not committed with specific hardware and software, which facilitates reusability for different platforms.

### 2.3 The Aerostack Library of Aerial Robotic Components

Aerostack provides a library of reusable software components for aerial robotics. We use the name of process to call each elemental component of the library. The notion of process is appropriate since it helps to divide the whole problem of automated support for UASs into partial functional roles. Each process has a function, i.e., a purpose or practical use for which the process is designed, and it is named as an agent according to its main function, for example: mission planner (main function: planning a mission) or obstacle recognizer (main function: recognize obstacles).

The computational support of a process is designed as an atomic executable unit (a data processor) that receives input data and, as a result of a certain information processing, generates output data. The library of processes is implemented using ROS (Robot Operating System) [22] and each process in Aerostack is implemented as a ROS node. Aerostack runs concurrently processes in a multitasking operating system using the inter-process communication methods provided by ROS: (1) a publish-subscribe mechanism using messages and topics, and (2) a request-reply scheme (services). This multitasking support is important, for example, to execute independently processes at lower level layers (executive and reactive, with frequencies between 10 Hz and 1000 Hz) and higher level layers (reflective and deliberative, with frequencies between 0.1 Hz and 10 Hz). Planning algorithms can be computationally more expensive, so they must be decoupled for real-time execution and avoid slow down the reaction time.

The library of components provides modularity and it is open, so new processes can be included easily in the future, without changing the core of the system and the rest of processes. To create a new process it is necessary to program the corresponding algorithms and data structures in a class (e.g., in C++ language) and apply certain Aerostack conventions to be part of the library. For example, it must be created as a ROS node and it must be subclass of a specific class (called DroneProcess) that provides a common functionality (supervision, execution control and standard error messages)

Processes can also be used by developers outside a multi-layered architecture to reuse certain algorithms

(e.g., computer vision algorithms) for a new UAS. In this case, they can be used as single classes without the dependence on the multi-layered architecture.

### 2.4 The Component Assembly Mechanism

To develop a software architecture for a particular UAS, the developer follows a compositional modeling approach using the processes from the library as model fragments or building blocks. The developer can select the appropriate processes from the Aerostack library and compose the global architecture by assembling and adapting the selected components configuring their inputs, outputs and local parameters (if they have parameters). The developer selects the components according to the required features for the UAS. For example, if the developer wants to build a UAS with the feature self-localization by visual markers she or he must select four processes: visual markers localizer, obstacle detector visual marks, obstacle distance calculator and self localizer. The relation between features and processes is documented in the Aerostack library to facilitate this selection to the developer.

Processes are grouped in systems. A system is a complex module that includes a set of interconnected processes that provides a common functionality. In general, the idea of a process in Aerostack is similar to the concept of an atomic functional block used in the SysML (System Modeling Language, a general purpose modeling language used in systems engineering) with input/output ports. The idea of system similar to the concept of functional block (composite block) used in SysML, with input/output ports.

Aerostack uses an ontology to represent input/output data of processes and facilitates the semantic interoperability of the different components. The ontology is organized according to the multi-layered architecture. This ontology has been defined specifically for Aerostack following common terminology found in the research literature about robotics and aerial systems. The ontology defines the formal and explicit specification of shared concepts. The current formalization of this ontology is based on common data representations (using ROS messages). A complete formal specification of this ontology using an appropriate language (e.g., OWL) is a pending task to be done in the future.

The terminology presented below shows categories of data in aerial robotics that provide abstraction. The level of abstraction has been carefully selected in order to be usable in the domain of robotics, but they are generic to be reusable, i.e., they do not define specific data representations used in actual implementations. We use these concepts in the rest of the paper to define the types of inputs and outputs of the processes and systems to describe the architecture in a general way. The ontology includes the following concepts:

- *Raw measurements*: Values corresponding to direct measurements recorded by sensors. Aerostack uses sensor-independent parameters whose values are obtained with the corresponding interfaces (e.g. all the cameras, despite of being from different manufacturers, use the same measurement data type for representing the acquired images, allowing therefore the interoperability between them).
- *Extracted features*: Single features extracted from measurements of physical quantities. In general, the extracted features can include a partial interpretation of characteristics of the environment such as lines, intersections, visual markers, approximate pose, etc.
- *Self-localization*: Robot localization in the environment together with its kinematic values (e.g., velocities) as they are believed by the robot. For example: pose, velocities, accelerations, forces, torques, etc. This data encodes the Situation Awareness of the Robot.
- *Environment understanding*: Characteristics of the environment and its elements as they are believed by the robot. For example: walls, pole obstacles, other robots, distance to obstacles, etc. This data encodes the Situation Awareness of the Environment.
- *Internal state*: This encodes the Self-awareness of the Robot, including the self characteristic of the robot. For example: pose of the sensors in the robot, battery level, etc.
- *Perception mode*: Representation of the perception set-up, including the enabled sensors, the feature extractors and the situation awareness components, together with their relationship.
- *Robot commands*: Motion values that are accepted by the actuators. Examples are: voltage, or Pulse Width Modulation (PWM).

- *Motion references*: Motion values to be considered as goals by controllers or planners. Examples of references are: position, velocity, or yaw.
- *Motion mode*: Representation of the configuration of the motor system, including the enabled controllers and their relationships.
- *Actions*: An action is used to express an elementary goal that the aerial robot is able to achieve by itself, using its own actuators. Actions might have a finite duration or infinite duration until they are disabled. An illustrative set of actions are: take off, go to a point, move forwards, pick up item, drop item, rotate yaw, and land.

  We distinguish between two categories of actions: Executive actions are the ones accepted by the Executive System and work in present time. Deliberative actions involve planning, and therefore work in future time. These deliberative actions are only accepted by the Planning System.

  Actions might include, not only the symbolic name that expresses the action to be done, but also the complement of the action (called action complement). Examples of actions and their action complements are: go to point $P$, being the point $P$ the action complement of the action go to point; pick up item $A$, begin the item $A$ the goal of the action pick up item.

  Additionally, actions return a feedback (action feedback), that is a performance value about the incremental progress of an action. Similarly, once an action is finished (because it is completed, or because it is unfeasible), a performance value is returned (Action Result).
- *Skills*: To represent a particular robot's ability the concept of skill is used. An illustrative set of skills is the following: reactively avoid obstacles, self locate using visual markers, and recognize items. Skills can be active or inactive in a particular robot. In general, skills have influence in the behavior of actions. Thus, skills can be understood as global modifiers for sets of actions. Skills have infinite duration until they are disabled. We use here the concept of skill defined in [13], in contrast to other meanings defined for skills in the literature of robotics.

  Skills might include, not only the skill by itself, but also complements that add some extra information. For example, self locate using visual markers with camera $C$, being $C$ the complement that defines which camera measurements must be used to detect visual markers and perform the self localization.

  Similarly to actions, skills return a feedback (skill feedback), that is a performance value of the enabled skill.
- *Planning references*: They indicate an specific goal given to the planning components with the objective to create a plan formed by motion references. An example of a planning reference is a point P that is required to be achieved and is given to a trajectory planner.
- *Mission*: Complex goal to be executed by the robot (e.g. search an object in a field, deliver a parcel, etc.). It can be specified by human operators with a set of tasks that describe the different parts of the mission to be done.
- *Task*: It encodes the same concept than the mission with the only difference of the size of the mission. A task is a complex goal with smaller complexity than a mission. The exact division between Mission and Task depends on the final user or developer decision.
- *Society knowledge*: This concept includes all the shared information between robotic agents. This shared knowledge might include raw sensor measurements, extracted features, situation awareness information, control and action goals, or in general anything. Depending on the nature of the shared information, it might be included in any component of the architecture.
- *Unexpected operation*: To permit the Supervision System to react to environment or operation changes, several variables require to be monitored. In the case that one of the monitored variables acquires a certain value, a flag is enabled and this monitored event message is sent.
- *Process problem*: This includes all the possible problems related to the processes execution that are monitored by the Process Monitor of the Supervision System.
- *Process management*: To group the common processes messages related to their performance, state, problems, and in general, their management. For example a sensor might report a problem if the sensor stopped working (but the software process is still running); or it might report a

performance issue if the measurement rate has decreased because the hardware is too hot and needs to cool down; or it might report an error if the execution failed and the process stopped.

## 3 Detail of Aerostack Subsystems

This section describes in detail the main subsystems of the Aerostack architecture. For each subsystem, a general description is provided.

### 3.1 The Feature Extraction System

The goal of the Feature Extraction System is to transform the raw measurements provided by the sensors into a simpler and more usable information (see Fig. 2). These extracted features simplify the raw measurements in a way that the components that receive them are able to use them more efficiently.

The inputs of this system are the raw measurements given by the sensors; and the perception mode, given by the Executive System, that encodes the configuration of the complete perception system (including the sensors, the Feature Extraction System, and the Situation Awareness System). Its only output are the extracted features.

Similarly to the other systems, the specification of the Feature Extraction System imposes no prior restriction on the type or quantity of its inputs and outputs and the restrictions will only come with its particular implementation (and its related systems). Thanks to this on purpose open definition, any kind of feature extractor, using any kind of sensor information might be used in Aerostack architecture.

To ensure the correct operation of the Feature Extraction System, the Executive System (described in Section 3.4), enables, disables and connects, by means of the perception mode input, the available components in a way that they will never cause a fault on the system. It is therefore a requirement that the available components count with a proper start up and a shutdown routine.

Aerostack counts on with a large number of components included in the Feature Extraction System, ranging from:

- Signal filters that extract a particular frequency component of a measurement given by a sensor, for example low-pass, high-pass, band-pass, etc.
- Computer vision based detectors and trackers, for example a FAST keypoints extractor, or ArUco visual marker detector [6].
- Point cloud based detectors.
- Etc.

### 3.2 The Situation Awareness System

The Situation Awareness System, represented in Fig. 3, has the goal to interpret the information provided by the sensors and the Feature Extraction System to create a useful understanding of the current situation to be used in the decision making and control processes. The Situation Awareness System is highly dependent on the mission, the environment, and the sensors, architecture and robot setup.

The inputs of this system are the raw measurements given by the sensors, the processed information given by the Feature Extraction System, and the motion references and actuator commands from the

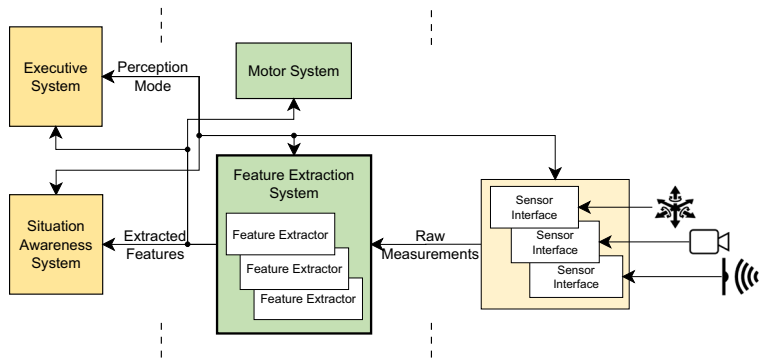



**Fig. 2** General description of the Feature Extraction System, and its relationship with the rest of the components of Aerostack

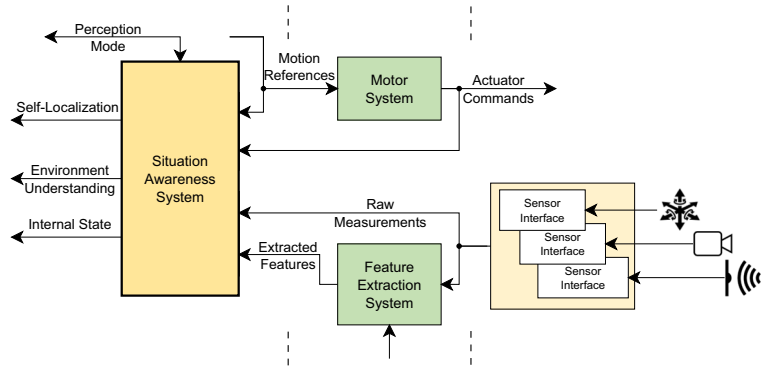



**Fig. 3** General description of the Situation Awareness System, and its relationship with the rest of the components of Aerostack

Motor System. It is worth to note that this specification of the Situation Awareness System impose no prior restriction on the number, type, or nature of the used sensors, or the exploited Feature Extractors. The sensors can as well be located on board the robotic agent or on ground. The restrictions will come only with the particular implementation of the Situation Awareness System.

In general and for completion, as the other subsystems, the Situation Awareness System could also include as inputs, some additional information coming from the Communication System (Section 3.7) and given by other agents of the multi-robot system (if any), like their estimated state, their estimated state of the map, or other information given by their sensors or Feature Extraction Systems. Including this information provided by other agents of the system allows the creation of a multi-robot Situation Awareness System.

The perception mode input, given by the Executive System, encodes the configuration of the complete perception system (including the sensors, the Feature Extraction System, and the Situation Awareness System).

The outputs of the Situation Awareness System, that define its functionality, are the following:

- *Self-localization*: estimation of the full situation state of the robot, as for example its pose, its velocity and its acceleration.
- *Internal state*: information related to the internal variables of the aerial robot like battery level, or kind, number, pose or range of the sensors equipped on the robot, etc.
- *Environment understanding*: estimate of the map of the environment in a format that is usable for the rest of the components of Aerostack.

It is important to note that there is no restriction about how the Situation Awareness System represents internally the map or environment. This internal model needs to be converted to the appropriate representation, to make it usable for the rest of the components of Aerostack, conferring more flexibility internally but ensuring its performance within the rest of the components.

Different examples of the Situation Awareness System can be found on [19, 21, 25, 29].

### 3.3 The Motor System

The Motor System, represented in Fig. 4, has the responsibility to generate the actuator commands for the hardware elements of the robot, ensuring that the commanded motion references are followed, knowing the motion feedback.

The inputs of the Motor System are grouped in:

- *Motion references*, that are motion values to be considered as goals, and are given by the Executive System.
- *Motion feedback*, including the raw measurements given by the sensors; the estimated state of the robot and the environment given by the Situation Awareness System; and the extracted features given by the Feature Extraction System.
- *Motion mode*, that encodes the configuration of the Motor System and modifies its behavior, and that is given by the Executive System.

The only output of the Motor System are the actuator commands, used by the hardware of the robot.

Similarly to the other systems, the specification of the Motor System imposes no prior restriction on the type or quantity of its inputs and outputs and the

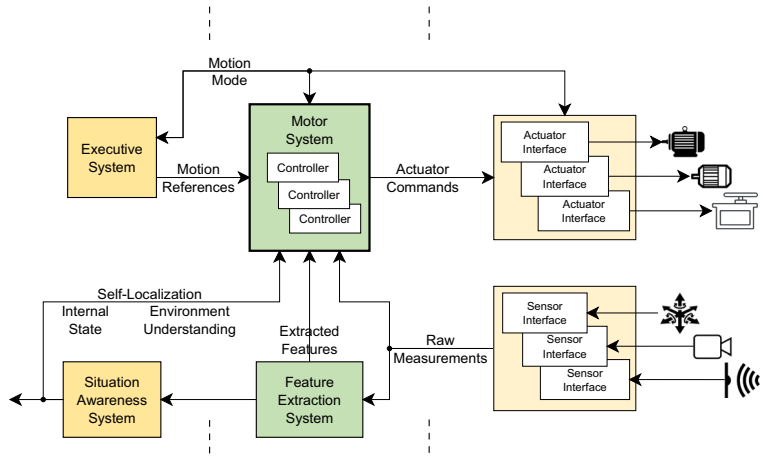



**Fig. 4** General description of the Motor System, and its relationship with the rest of the components of Aerostack

restrictions will only come with the particular implementation of the Motor System (and its related systems). Thanks to this on purpose open definition, any kind of controller, using any kind of motion feedback might be used in Aerostack architecture.

The Motor System comprises, therefore, a set of controllers that can be used independently or in cascade to allow the robot to follow a given motion reference. To ensure the correct operation of the Motor System, the Executive System (described in Section 3.4), enables, disables and connects, by means of the motion mode input, the available controllers in a way that they will never cause a fault on the system. It is therefore a requirement that the controllers included in the Motor System count with a proper start up and a shutdown routine.

The different controllers of the current version of Aerostack are the following:

- Low-level embedded controllers (provided by the hardware autopilot of the aerial platform): motor speed controller, angular velocity control, horizontal attitude control.
- Navigation controllers (deeply described in [15, 16, 18]): linear velocity controllers, position controllers, heading controller, point to look control component, path following component.
- Visual servoing controller (deeply described in [17, 20]).

It is worth to highlight that nowadays, some commercial multirotors, specially the most advanced ones, include embedded in their hardware many controllers that can be used with different control modes, having internally its own Motor System (and optionally its own Situational Awareness System). Nevertheless, as this is not a general fact, and it was even less common in the moment that Aerostack was firstly designed, Aerostack provides its own controllers. However, the definition of the Motor System of Aerostack does not enforce to use Aerostack controllers, being possible to use these embedded controllers.

### 3.4 The Executive System

The main goal of the Executive System (represented in Fig. 5) is to accept directives from the deliberative layer (or from a human operator) and to sequence them to be performed by the reactive layer. To be able to properly sequence these directives, the Executive System uses the feedback given by the perception components.

Two different kinds of directives are accepted from the deliberative layer, actions and skills (described in Section 2).

An important property of the Executive System is that it creates a clear separation between two representation levels: (1) a symbolic level, where goals are described with linguistic symbols, which is very useful as an operator language for the specification of missions, and (2) a controller or perception level, where goals are described with quantitative values that are used as reference values for controllers, or commands of the components of the perception system.

The Executive System has a symbolic representation of the dynamic state of the aerial robot. For example, the state of the robot might be landed, taking off, hovering, or landing. These states can be divided

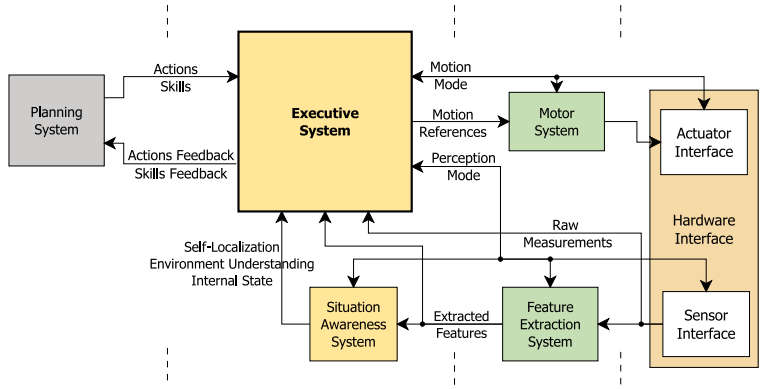



**Fig. 5** General description of the Executive System, and its relationship with the rest of the components of Aerostack

in (1) states with a finite duration (for example, the taking off state automatically ends when the aerial robot has reached a specific altitude), and (2) states with undetermined duration (for example, the hover state only ends when the Executive System disables it).

These states and their transitions can be described using a finite state machine. The transitions between states describe the feasible actions that the Executive System can accept. The skills are modifiers of these states and transitions. For example, the action take-off, that represents a transition from the landed state to the taking-off state, only exists if a skill for measuring the flying altitude is enabled.

Actions are therefore augmented with skills. Some skills are required for the execution of a particular action, but some other skills are optional and only chosen by the operator. The notion of skill is useful as an intuitive concept to express more easily what complex abilities should be active, without considering low-level technical details. Internally, a skill is automatically supported by a set of running processes. Thus, the activation of skills is associated to the increase of resource consumption (memory space, processing time, battery charge) so it is important to deactivate unnecessary skills when it is possible.

The Executive System has therefore a complete knowledge of all the possible actions and skills, together with their possible effects. The Executive System has the responsibility of ensuring that the transitions between states is feasible. For example, a take off action is only allowed from a landed state, and never from the hovering state.

Another additional task of the Executive System is to prepare all the components (motion and perception components) involved on a specific requested action or skill, enabling them in the proper instant of time in a way that they do not collide with other components, monitoring their state, and in case of being unavailable, generating a response to the requested action or skill.

It is important to note that the Executive System is only checking the feasibility of an action or skill in the present time, unlike the Planning System, that is checking the feasibility in the future time. For example, since a take off action requires to start the propellers of the aerial robot, if the propellers cannot be started, the Executive System will notice that the action take-off is not feasible, nevertheless, the Executive System will never check for example if the battery level would allow to complete an specific mission before taking off.

The Executive System is able to enable, disable and reconfigure all the motion components (Motor System and Actuator Interfaces) by means of the motion mode command; and to enable, disable and reconfigure all the perception components (Situation Awareness System, Feature Extraction System, and Sensor Interfaces) by means of the perception mode.

Different architectures for the Executive System have been tested in Aerostack. The tested approach range from a centralized architecture where a single component performs the complete functionality of the Executive System (an example of this centralized version is shown in [13] where the Executive System is implemented as a main process called Manager of Actions and Skills), to a distributed layered architecture (presented in Fig. 6), where there is a coordinator component (the Behavior Manager) that interacts in two different levels of specialized components (the first level distinguish between actions and skills, and the second level distinguish between motion components and perception components). A further analysis

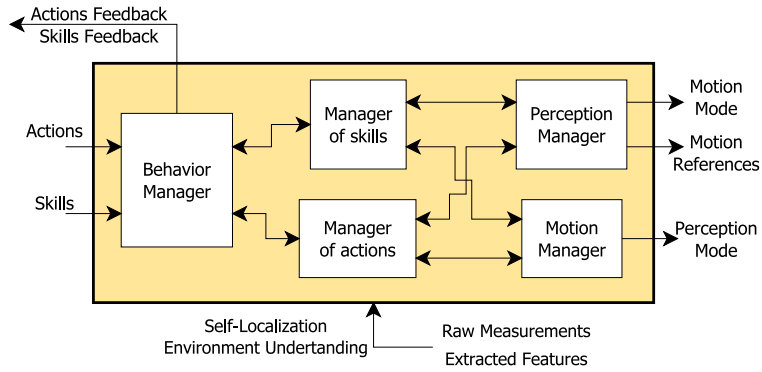



**Fig. 6** Detail of the Executive System

of the proposed architectures of the Executive System is out of the scope of this paper, being only worth to mention that distributed layered architectures perform better than centralized ones, being their complexity and the effort to add new states and actions smaller.

### 3.5 The Planning System

The Planning System, represented in Fig. 7, generates goals to accomplish a particular complex mission, task or deliberative action. These generated goals are represented as executive actions and skills that are forwarded to the Executive System. Additionally, it reacts to changes in the operation provided by the lower-level layers and to unexpected operation events given by the Supervision System, generating new goals that modify the previously produced ones. Unlike the Executive System, the Planning System works in future time, taking into account the current state of the robot, predicting the consequences of the planned actions in the future.

The Planning System has to be able to generate goals fast enough to make possible an efficient reaction to changes in the mission, in the environment, or in the state of the robot. This is specially critical in aerial robots, since their unstable nature disqualifies them to passively wait a slow response of the Planning System.

The Planning System can be divided in three components:

- *Mission planner*. The mission planner (Section 3.5.1) is a deliberative component that receives as input a mission, a task or a deliberative action to be performed and generates as output a sequence of executive actions to be executed together with their corresponding required skills. The mission planner generates such actions considering the dynamic changes in the environment.
- *Action specialist*. The action specialist helps the mission planner during the deliberation to anticipate if a tentative actions is feasible, according to

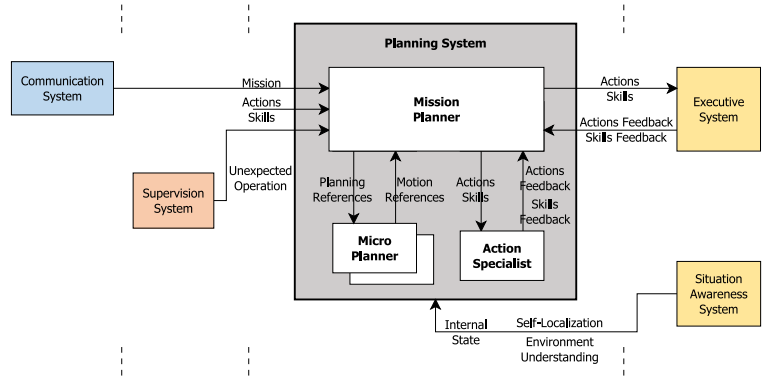



**Fig. 7** General description of the Planning System, and its relationship with the rest of the components of Aerostack

the current situation. For this purpose, the action specialist has a knowledge of all possible actions and their effects. For example, the action specialist can verify in advance that a certain spatial point is too far to be reached, considering the current charge of battery. The action specialist is also able to predict physical magnitudes of certain actions such as required time, distance to cover, amount of battery to consume, required free space, etc. It is important to know, that this estimation is approximate, i.e. it is done using inexact models and help to find more efficiently the solution, anticipating certain clear solutions. This means that, when the actions are executed, the robot can behave in a different way due to specific changes in the environment. It is important to note that, unlike the Executive System, the action specialist is checking the feasibility of the actions, not only in the present time, but also in the future time.

– *Micro planners* (optional, chosen by the mission planner). The micro planners generate motion references that can be followed by the robot, given the current (self-localization, and internal state) and desired (planning references) state of the robot, and the environment map (environment understanding). An example of a micro planner is the trajectory planner (Section 3.5.2), that generates trajectory references.

### *3.5.1 Mission Planner*

The mission planner receives as input a mission, a task or a deliberative action to be performed and generates as output a sequence of executive actions to be performed together with their corresponding skills which the Executive System is able to manage. To do this, it is able to request motion references to several micro planners by sending them planning references. For example, the deliberative action "go to point ($P$) with active obstacle avoidance", being the point ($P$) the action complement that describes the coordinates of the the destination point, and it has to be transformed in the following executive action, "follow path ($T$) in path following mode", being the path ($T$) the action complement formulated as a sequence of waypoints. To do so, it is used a path planner (see Section 3.5.2) that generates a collision-free path ($T$) to reach the point ($P$).

The mission planner is able to react to changes in the operation provided by the lower-level layers. These changes in operation might include: changes in the environment (e.g. a new obstacle appeared) or in the knowledge of the environment (e.g. a new obstacle is mapped), changes in the internal state of the robot (e.g. low battery warning), changes in planned events (e.g. a planned action has finished, a planned trajectory is not collision-free anymore).

The mission planner is able to react to unexpected operations given by the Supervision System (Section 3.6). These unexpected operations might be any kind of difficulties, including action problems (e.g. an action cannot be executed) or processes problems (e.g. a process has unexpectedly finished).

Finally, the mission planner has the responsibility to send to the Executive System, if feasible, the planned executive actions and skills.

Aerostack proposes, but not limits, to use task-based mission planners, that decompose a complete mission in a task tree. Tasks are defined as a basic component to structure a mission with a modular organization. At the same time, every task might be decomposed in another simpler task tree. An important requirement is that every task tree always ends with actions (either executive or deliberative).

The first mission planner used in Aerostack (presented in [19, 21, 25, 28, 29]) proposes to use a basic task-based mission planner, whose missions have a predefined sequential behavior. Missions are defined as a task tree by the operator using a particular language using XML syntax to be readable by both humans and machines.

The mission definition includes the possibility to use while-loop and conditional-clauses allowing to change the mission flow and reacting therefore to some monitored events. An example of a while-loop is the following: while battery level is greater than a specific value, keep doing the current task. An example of a conditional-clause is: if a specific visual marker is detected, perform a particular task.

A second mission planner available in Aerostack, presented in [13], is an extension of a task based approach (with task trees) together with reactive planning (using event handlers formulated with rules) to facilitate a more flexible specification of plans to be adaptive to a dynamic environment. This representation is supported by a formal language, the TML language

(Task-based Mission specification Language), that incorporates deliberative terms about actions and skills that were not considered in previous planners of Aerostack. This language formalization creates a new simpler but complete grammar and vocabulary that is verified after the mission definition by the operator, allowing to correct errors, giving robustness to the mission planner.

Another mission planner available in Aerostack (see [24]) proposes a task-based mission planner able to work in dynamic environments.

The operator only has to define the high level mission without specifying the complete mission tree, but only high level tasks. These high level tasks (called behaviors) are internally represented as a task tree that might be called by the operator and have been included in the mission planner. Examples of these behaviors are: explore an area (that requires to sample the area to explore and concatenate several navigation to a point $P$ actions); or find an object (that requires to explore an area until the object is found, including specific stops to look for the object with the onboard camera).

Together with the behaviors, a set of restrictions and conditions that have be satisfied for every task, have been included in the mission planner. Examples of the restrictions and conditions are: before moving to point $P$, a take-off is needed; or, when trying to navigate to a point $P_A$, if it is occupied, navigate to a point $P_{A'}$ in the neighborhood of point $P_A$.

The complete mission tree can be generated from a very simple set of tasks and behaviors. The missions and tasks are therefore automatically nested, creating complex missions without the operator intervention. Additionally, it allows to interact with dynamic and changing environments.

### *3.5.2 Trajectory Planner*

The trajectory planner generates deliberative motion references for the aerial robot. These motion references might include pose references, velocity references and acceleration references (among others) with or without time constraints. The generated motion references have to be compatible with the input of the used controllers available in the Motor System.

For the sake of simplicity, only pose references without time constraints are considered as the output of the trajectory planner since the existing path following controller incorporates an acceleration and velocity along path planner.

In common under-actuated multirotor aerial robots, only four degrees of freedom are controllable, normally its heading (yaw angle) and its position in world coordinates. Additionally, the commanded heading and position can be decoupled, so the pose path planner might be divided in:

– *Position path planner*. It generates collision-free paths that can be followed by the aerial robot, given the current and desired position of the aerial robot, and the shape of the obstacles (obstacles includes static environment obstacles and moving obstacles). The available position path planner is implemented based on PRM, Potential Field Map and A$^\star$ (see [19, 21, 25, 29] for more details about the trajectory planner).
– *Heading path planner*. It generates heading motion references that must be followed by the aerial robot. For example, the yaw value can be specified as a point to look or as a yaw to look. This can be used with different objectives, like maximizing the performance of the on-board sensors (for example a camera), or to achieve a particular goal (for example to watch over some target).

Both proposed path planners, position and heading, might be used independently, depending on the mission planner requirements. Common under-actuated multirotor aerial robots are normally symmetric in their horizontal dimensions, and therefore, for common navigation tasks, to changing the heading angle is not needed to reach a particular position point, and consequently, the heading planning might be decoupled from the position planning.

It is important to note, for the sake of generality, that aerial robots with more than four degrees of freedom (e.g. multirotors with tilting propellers or multirotors with manipulators) might be used, with more complex path or trajectory planners. The proposed architecture allows to use any kind of aerial robots, despite the simplification done in this section.

## 3.6 The Supervision System

The goal of the Supervision System (represented in Fig. 8) is to ensure the correct autonomous behavior of the whole aerial robot. It evaluates if the robot is actually making progress to its goals and to react

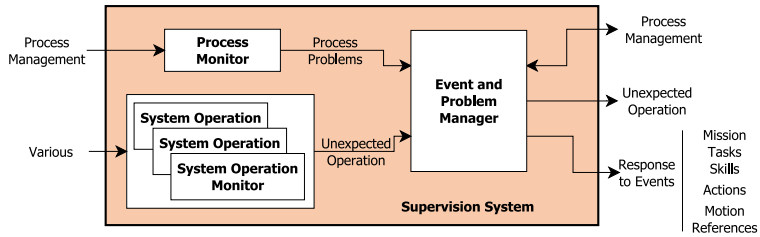



**Fig. 8** General description of the Supervision System. Since it is supervising all the other layers, it has a tight relationship with all the components of Aerostack

in the presence of problems or unexpected situations (e.g. faults, lost communications, etc.) with recovery actions. The Supervision System helps to provide therefore a fault-tolerance execution. In general, handling fault-tolerance typically consists of three steps: failure detection, notification, and recovery.

The general structure of the Supervision System includes (1) a set of specialists in kinds of events and problems, and (2) an event and problem manager.

Each specialist, called an system operation monitor, is specialized in detecting a particular class of events or problems, receiving as input the necessary information about situation and actions, and generating as output a category of event or problem. A specific important system operation monitor is the process monitor, that supervises the correct operation of the processes.

In addition to the system operation monitors, the Supervision System includes an event and problem manager, that is able to generate a response to events, malfunctions or unexpected situations, depending on the level of emergency and the nature of the event or problem received.

### *3.6.1 System Operation Monitor*

The system operation monitors are specialized in detecting a particular class of events or problems, receiving as input the necessary information about situation and actions and generating as output a category of event or problem.

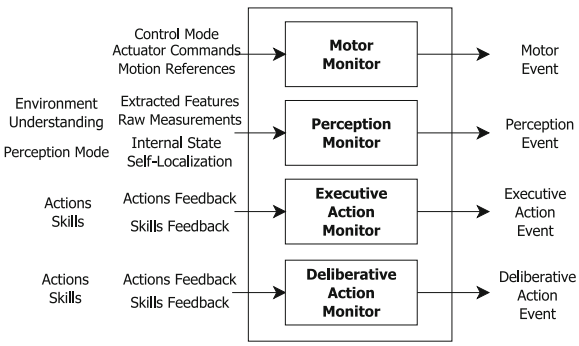



**Fig. 9** Example of system operation monitors

Examples of the system operation monitor (see Fig. 9) are the following:

- *Motor monitor*. It detects events and problems related to the Motor System and the Actuator Interfaces. An example would be a controller giving an incorrect actuator command, such as an $NaN$ value or an infinite value.
- *Perception monitor*. It detects events and problems related to the Situation Awareness System, the Feature Extraction System, and the Sensor Interfaces. Example of these events are incorrect measurements given by the sensors, like $NaN$ values; or large covariances values in the estimated self-localization that require to run re-localization algorithms.
- *Executive action monitor*. It monitors the executive actions and skills feedback from the Executive System. It supervises the execution of a requested action and informs when the action has been completed or when it has failed. For example, if the requested action is to move to a certain point, the action monitor verifies periodically the distance between the robot and the desired point and, when the distance is less than a threshold (established by a configuration parameter), the action monitor notifies that the requested action has been completed.
- *Deliberative action monitor*. It monitors the deliberative layer, in the same way than the executive action monitor.

### *3.6.2 Process Monitor*

The process monitor has the mission of supervising the correct operation of the processes. The process monitor is capable of monitoring all different

processes hosted in different computers and is responsible for acquiring and informing about the different errors related to processes, including when a process stops its execution unexpectedly.

To check if a process is alive, the process monitor uses a watchdog technique. This technique consists of periodically sending signals to a supervisor (in this case, the process monitor) to ensure that the supervised component is still alive. If the supervised component stops sending signals for a certain amount of time, the supervised component is considered to be offline. By using a watchdog in Aerostack, every process sends an alive signal to the process monitor, so that the monitor can track if a process has interrupted its correct execution.

In addition, the process monitor gets the execution state (e.g. paused, running, etc.) of the processes. Each process automatically notifies its execution state to the process monitor when the process sends the alive signal.

### *3.6.3 Event and Problem Manager*

Recovery actions can be performed as a response to malfunctions or unexpected situations, detected by the problem monitors. In order to recover problems and depending on the level of emergency and the nature of the problem, the problem manager acts on a different level of the Aerostack. A priority scheme is needed, since the presence of a failure can propagate several detected errors. For example, if the altitude sensor fails, it can generate a fault detection in the sensor interface but also in other processes that use the altitude measurement (for example the Situation Awareness System or the Motor System).

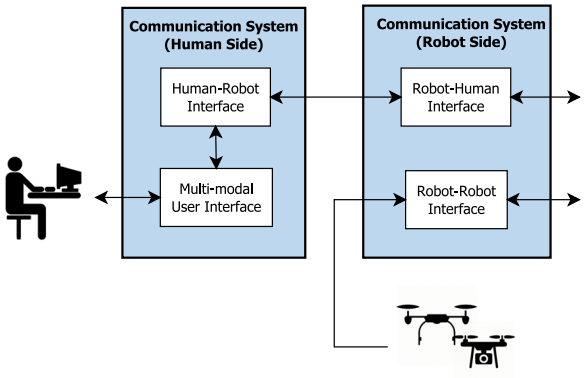



**Fig. 10** General description of the Communication System, and its relationship with the rest of the components of Aerostack

### 3.7 The Communication System

The goal of the Communication System, represented in Fig. 10, is to allow a robotic agent to exchange information with the human operators and with the rest of the robots of the system. This system is able, if needed, to establish a bidirectional communication with all the components of the agent, to permit this information transfer.

The Communication System can be separated in two main parts, (1) the robot side, that is included in every robot agent architecture, and (2) the human side, that has one instance per human operator.

Both sides include interfaces (in Fig. 10, called $X - Y$ Interface, being $X, Y = \{Robot, Human\}$) that convert data between different formats and networks allowing the intercommunication between all the agents (human or robotic) of the system. In general, the implementation of the communication interfaces is highly dependent of the kind of network used. For example, internally, every robotic agent might use ROS as the middleware for interprocess communication, but, between agents, MavLink middleware might be used for inter-agent communication.

The human side of the Communication System also includes multimodal user interfaces with graphics, speech, visual images, hand gestures, among others, that can simplify the interaction between the human and the aerial robot. A review of the multi-modal user interfaces that are available in Aerostack can be found in [32].

## 4 Aerostack Evaluation

Aerostack is fully operative, validated since its first alpha version in February 2013, by simulations and real flight tests with multiple aerial robots flying simultaneously (up to five), and with five different aerial platforms equipped with diverse sensors. In the following sections, we describe additional evidences related to the evaluation of Aerostack, enumerating first a set of reported used (Section 4.1), then describing a full experiment (Section 4.2) and finishing with some general evaluation metrics (Section 4.3).

### 4.1 Reported uses of Aerostack

Aerostack has been used from its first pre-release in the development of many different projects.

In the one hand, it has been used in three different aerial robotics competitions:

- In IMAV 2013 [19, 21], where a swarm of up to five Parrot ARDrone 2.0 platforms were flying simultaneously executing a navigation indoors mission. The competition ended, being awarded with the first place in the category of indoor autonomy.
- In IARC 2014 [26], an AscTec Pelican equipped with an AscTec Mastermind computer and multiple cameras was performing a very complex LIDAR-denied indoors mission, obtaining two awards: the best system control and the best target detection.
- In IMAV 2016, where a custom custom PixHawk based quadrotor equipped with an Intel NUC was performing a fully autonomous indoor mission.

Simultaneously, Aerostack has been used as the main framework for other research activities:

- In [25, 29], Aerostack was first used to perform complex navigation missions of a fully autonomous swarm (up to five) of aerial robots.
- In [17, 20], using Aerostack, the authors demonstrated that a fully autonomous aerial robot was able to follow any object using computer vision algorithms.
- In [27], a fully autonomous search and rescue mission was carried out thanks to Aerostack.
- In [24], the authors expanded Aerostack capabilities to demonstrate the benefit of using a global coordinator to accomplish high-level missions requested by the user with a fully autonomous swarm (more than three) of aerial robots.
- In [32] Aerostack was used for research and development of Natural User Interfaces for Human-Drone Interaction using hand gestures, speech, body movements and visual cues.
- In [13] a task-based mission specification language was developed to enhance the previous Aerostack mission planner.

Finally, Aerostack has been employed in multiple shows and exhibitions for both general and specialized public, highlighting the 2016 European Night of the Researchers[2] where three Parrot ARDrone 2.0 did an autonomy demonstration which was attended by more than 400 people.

The excellent results achieved in the international competitions, the success in the diverse research activities, and the high level of satisfaction of the audience in the public demonstrations, evidences the good performance of Aerostack in terms of functionality, usability, and reliability, awakening the interest of the scientific community in Aerostack.

### 4.2 Experiment

This section describes an experiment as a representative example of system development and mission execution to demonstrate and illustrate the capabilities of Aerostack.

The experiment described here is based on a search and rescue mission. In this mission, an emergency situation is emulated inside a house and the first response to the emergency, until the human rescue team arrives to the accident area, is demonstrated (Fig. 12).

Two exploratory fully autonomous aerial robots navigate from the rescue equipment base to the inside of the house, entering through an open window or door, searching for a target (e.g. an injured human subject).

As soon as an subject is detected, a rescue aerial robot is called, arriving to the position where the subject has been found. This rescue aerial robot has the responsibility of guaranteeing the survival of the subject while the human rescue team arrives to the area. The rescue aerial robot might therefore, for example deliver a first aid package, and track and monitor the human subject, interacting with him or her by means of natural interfaces. Thanks to this aerial robot, the rescue equipment is able to monitor the state of the injured person, being able to generate an efficient response.

If the emergency situation has been stabilized and the subject does not need assistance anymore, or if the human rescue team has arrived to the area, the rescue aerial robot might be commanded (by the subject or the rescue team) to come back to the rescue equipment

**Table 1** Summary of the software components used for the specific system architecture corresponding to the experiment

| Layers | System | Processes |
|---|---|---|
| Physical layer | Hardware Interface | ARDrone drivers (13 proc.) |
| Reactive layer | Feature Extraction System | Front image rectifier |
| | | ArUco Eye |
| | | TLD tracker (2 proc.) |
| | Motor system | Trajectory controller |
| | | Visual servoing controller |
| | | Heading commander |
| Executive layer | Situation Awareness System | Odometry based self localizer |
| | | Visual marker based localizer |
| | | Tracker eye |
| | | Obstacle recognizer |
| | | Obstacle distance calculator |
| | Executive System | Manager of actions |
| Deliberative layer | Planning System | Mission planner |
| | | Trajectory planner |
| Reflective layer | Supervision System | Process monitor |
| Social layer | Communication System | Robot-robot interface |
| | | Alphanumeric user interface |
| | | Graphical user interface (GUI) |
| | | TLD GUI |
| | | ArUco GUI |
| | | Speech and sound interface (3 proc.) |

base. In the mean time, the search aerial robots keep searching for more targets. As soon as the house is completely explored, or the emergency situation has been stabilized, the search aerial robot finish their exploratory mission, and they autonomously return to the rescue equipment base.

#### *4.2.1 Hardware Configuration*

To execute Aerostack, we used three Unix based laptops: (1) computer A with Intel i7-4510U (2.00GHz, 4 cores) and 8 GB of memory (2) computer B with an Intel i7-3612QM (2.1GHz, 4 cores) and 8 GB of memory and (3) computer C with Intel i7-6700HQ (2.4 GHz, 4 cores) and 20GB of memory. The three of them had WiFi and ethernet connection. The laptops were connected in a LAN using their Ethernet interface and a switch.

For simplicity, we used AR Drones 2.0 as the aerial platforms, but other types of aerial platforms could also be used. Each aerial platform was individually connected to an associated laptop by means of the WiFi connection, creating as many additional WLANs as aerial platforms were used (three in this case).

#### *4.2.2 Specific System Architecture for the Experiment*

This section illustrates how we used the general Aerostack architecture and the library of processes for building the particular software system architecture for the concrete search and rescue problem corresponding to the experiment. Table 1 summarizes the components that we used for each aerial robot.

Figure 11 shows the organization of the processes of the Situation Awareness System. The figure shows a block diagram where each rectangle represents a process or a system (e.g., odometry based pose estimator, visual marker based pose estimator, etc.) and they are interconnected using flow ports (e.g., Aruco observations, estimated speeds, etc.).

In this case, the Situation Awareness System uses two processes for the self-localization and mapping that are (1) odometry based self localizer that uses the model of the quadrotor, the information given by

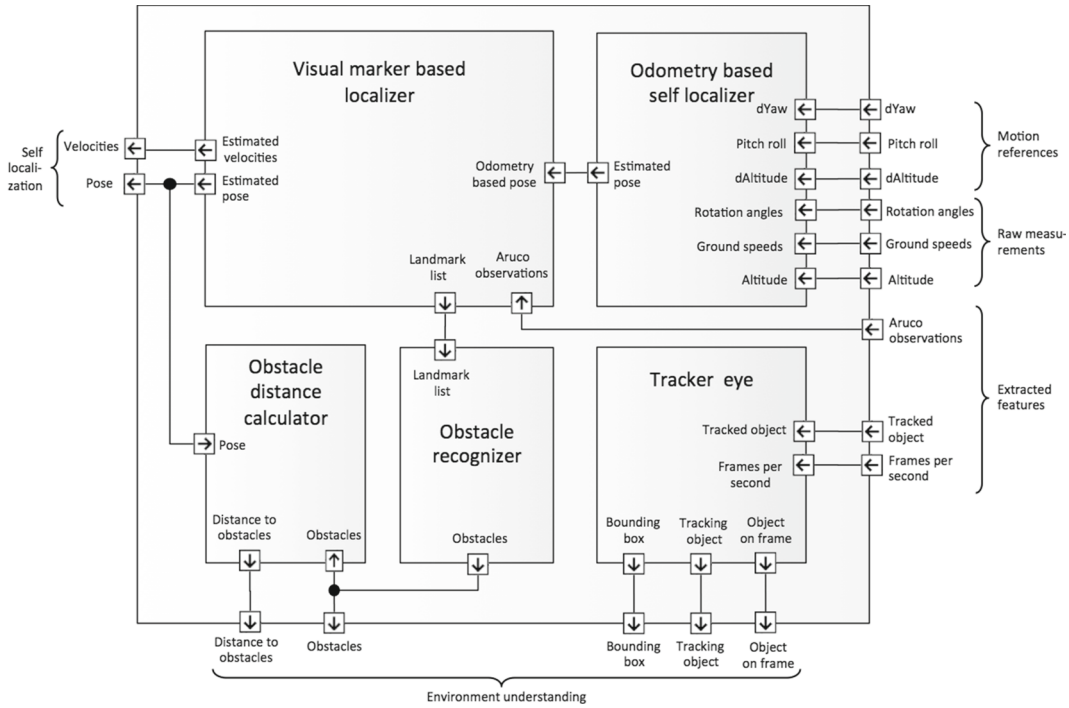



**Fig. 11** Block diagram of the Situation Awareness System used for the experiment

the IMU, the optical flow sensor and the altitude sensor to estimate the odometric pose of the aerial robot, and (2) a visual marker based localizer to estimate the pose and velocities of the aerial robot using the visual markers in the environment (e.g., Aruco visual markers) together with its odometric pose estimation with respect to the world reference frame, and the pose of the visual markers with respect to the world reference frame.

Three processes are used for the environment understanding that are (1) obstacle processor that converts the map of visual markers, that is only useful for the localization estimation processes into a usable map of geometric primitives (obstacles) based on a previous knowledge of the relationship between the visual markers and the geometric primitives; (2) obstacle distance calculator that extracts additional information of the map of geometric primitives like the distance between the current position of the aerial robot and the closest obstacle; and (3) tracker eye, that analyzes the information coming from the TLD tracker, estimating if the the object to track is present in the image.

The architecture implemented for this experiment includes the majority of the components defined in the general description along Section 3. However, the functionality of certain generic components has been distributed in this particular architecture for historical reasons to keep the compatibility with previous software versions. For example, the functionality of the action monitor is included in the manager of actions process. Additionally, the mission planner process incorporates some of the functionalities of the event detector. Furthermore, the trajectory controller process includes the management of all the controllers that this component has (velocity controller, position controller, trajectory controller), which corresponds to the Executive System in the general architecture. We are currently working on building new versions of such components to follow the general architecture.

This example corresponds to an application that uses the complete Aerostack framework, selecting the appropriate components. This means that it was not necessary to develop additional components (e.g., new type of sensors, new computer vision algorithms), and therefore, the effort carried out by the developers to perform this experiment was limited to:

– Software architecture configuration. Design the architecture of processes for each aerial robot.

Write the launch files that use the appropriate software components from Aerostack.

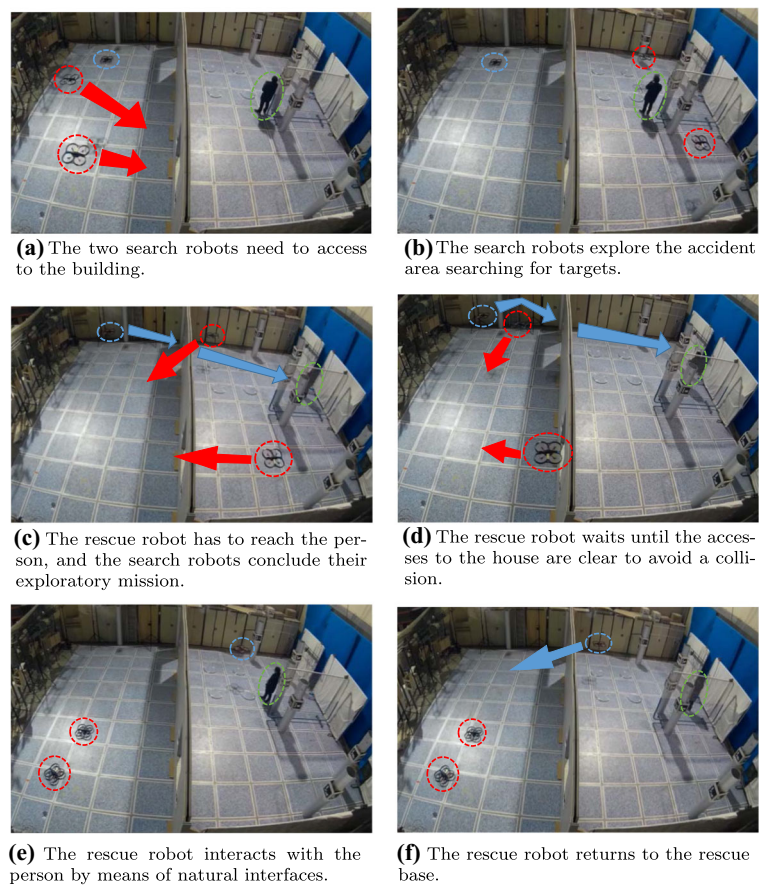
– Configuration of every component of every aerial robot. Write xml files that configure every component, including the environment map and the mission specification.

### *4.2.3 Mission Execution*

The global goal is to search for a subject in a spatial area (Fig. 12) and to naturally interact with him or her. The global search goal is distributed in different local sub-goals with two different aerial robots. Each robot covers a different local search area. A third aerial robot is used for the natural interaction with the subject. The environment is an indoors area, with simple shape (walls and poles) static elements, augmented with visual markers (ArUco markers). Both the obstacles and the searched subject are uniquely labeled thanks to the visual markers. We used ArUco markers to perform the localization and mapping task for simplicity although other more advanced methods could be also used.

In more detail, the mission is as follows:

– Two aerial robots take-off at the same time from one specific take-off point in the rescue equipment base.
– Each aerial robot covers a different search area defined as a set of waypoints as destination points.
– Unknown simple shape static obstacles are present and each aerial robot must detect them and avoid them. In addition, each aerial robot must avoid collisions with the other robots. The aerial robots must cross narrow areas and they must decide how to enter in the appropriate order to avoid collisions amongst themselves.
– When an aerial robot recognizes the presence of the subject, it notifies the third aerial robot with the location of the subject, and both initial aerial



**(a)** The two search robots need to access to the building.

**(b)** The search robots explore the accident area searching for targets.

**(c)** The rescue robot has to reach the person, and the search robots conclude their exploratory mission.

**(d)** The rescue robot waits until the accesses to the house are clear to avoid a collision.

**(e)** The rescue robot interacts with the person by means of natural interfaces.

**(f)** The rescue robot returns to the rescue base.

**Fig. 12** Different frames of the search and rescue mission

robots return to their respective take-off points and land.

– After receiving the notification about the location of the subject, the third aerial robot takes-off from its specific take-off point at the rescue equipment base and approaches the subject.
– The third aerial robot remains near the subject and naturally interacts with him or her until receiving a command from the subject to return to its take-off point and land.

Figure 12 shows different frames of the search and rescue mission. The rescue base (where the aerial robots take-off) is located in the left side of the images. The accident area is supposed to be the interior of a house, where the subject (highlighted in green) is supposed to be. A video with the complete execution of the mission can be watched in https://youtu.be/t2mJftbBHWc.

The trajectories followed by the aerial robots during the execution of the search and rescue mission, are shown in Fig. 13. The two search aerial robot followed the trajectories plotted in red and green, while the rescue aerial robot followed the blue trajectory.

Due to the complexity of the mission and the difficulty to understand the trajectories shown in Fig. 13, the experiment is divided in the different instant times displayed in Fig. 14. These plots also represent with dashed lines, the collision-free planned path provided by the Planning System. Similarly than before, the two search aerial robots followed the trajectories plotted in red and green, while the rescue aerial robot followed the blue trajectory.

The search and rescue mission is executed as follows:

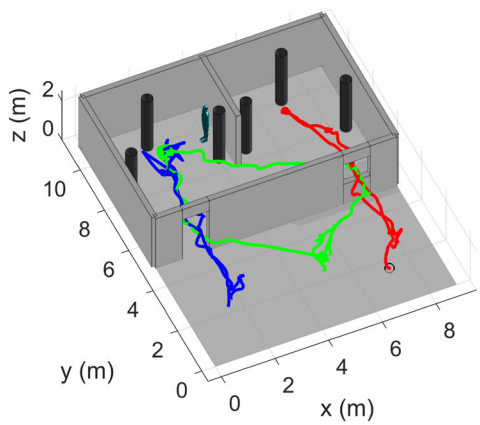
In Fig. 12a, the two search aerial robots (highlighted in red) take-off from the rescue base and start their exploratory mission, needing to access to the building (either through the window or the door). Figure 14a shows the trajectories followed by the two search aerial robots at this time instant.

Once inside the building, every search aerial robot explores the accident area, searching for targets (see Fig. 12b). Every aerial robot explores a different part of the accident area as in the trajectories in Fig. 14b.

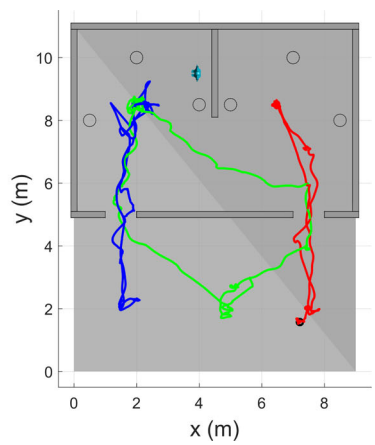
As soon as the subject is detected, the rescue aerial robot (highlighted in blue) takes off from the rescue base, with the objective to reach the subject (Fig. 12c). Figure 14d shows the trajectories of the search aerial robots when the subject is detected. In parallel, the search aerial robot conclude their mission and come back to the rescue base. Although it is not needed, the subject has a visual marker to be properly identified by the search robots.

As can be seen in Fig. 12d and the corresponding trajectories in Fig. 14e, the search robot is faster leaving the house than the rescue robot entering in it, so to avoid a collision, the search robot autonomously waits until the accesses to the house are clear.

In Figs. 12e and 14g, the rescue robot interacts with the subject by means of natural interfaces.

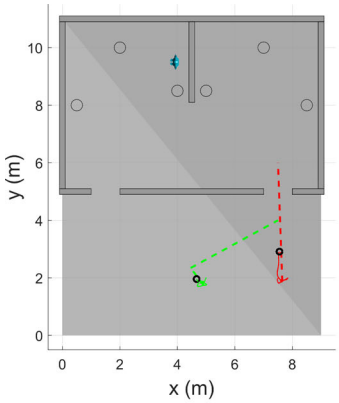



**(a)** 3D trajectory of the aerial robots in the search and rescue mission.

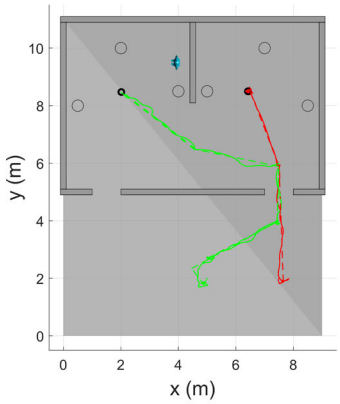



**(b)** 2D trajectory of the aerial robots in the search and rescue mission.

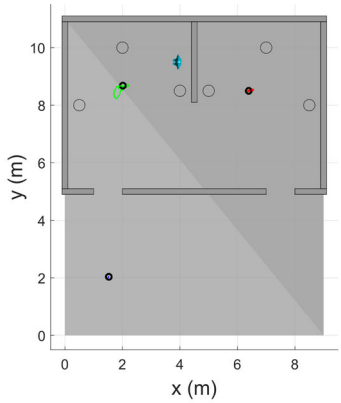
**Fig. 13** Trajectory followed by all the aerial robots in the entire search and rescue mission



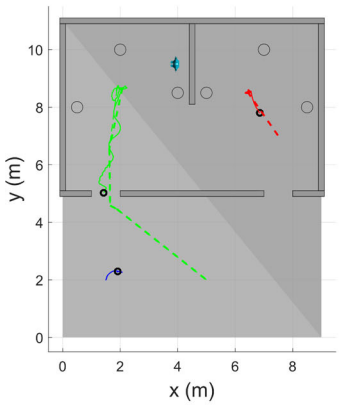

**(a)** 2D trajectories from time $t = 0$ s to $t = 15$ s.



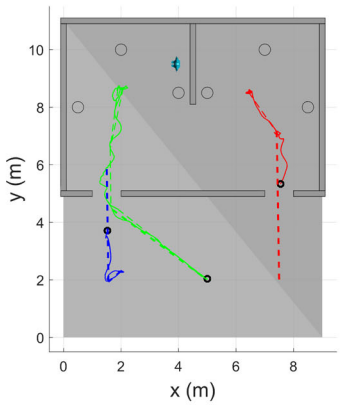

**(b)** 2D trajectories from time $t = 0$ s to $t = 55$ s.



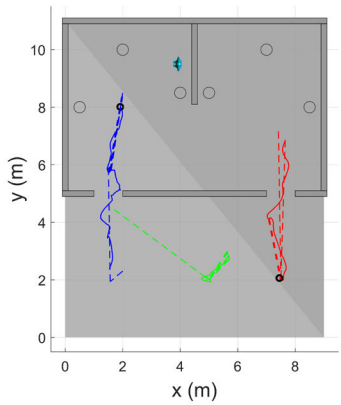

**(c)** 2D trajectories from time $t = 65$ s to $t = 75$ s.



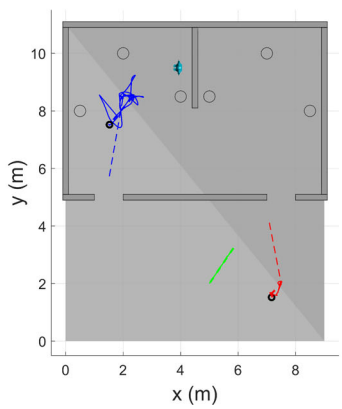

**(d)** 2D trajectories from time $t = 65$ s to $t = 85$ s.



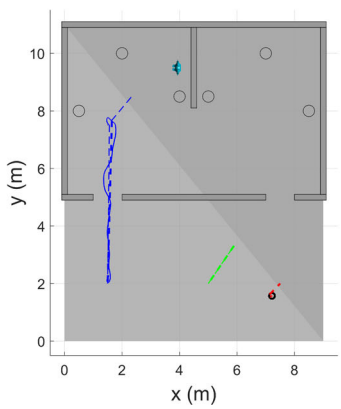

**(e)** 2D trajectories from time $t = 65$ s to $t = 98$ s.




**(f)** 2D trajectories from time $t = 95$ s to $t = 110$ s.




**(g)** 2D trajectories from time $t = 110$ s to $t = 160$ s.




**(h)** 2D trajectories from time $t = 160$ s to $t = 182.33$ s.

**Fig. 14** Different trajectories of the aerial robots in several time frames

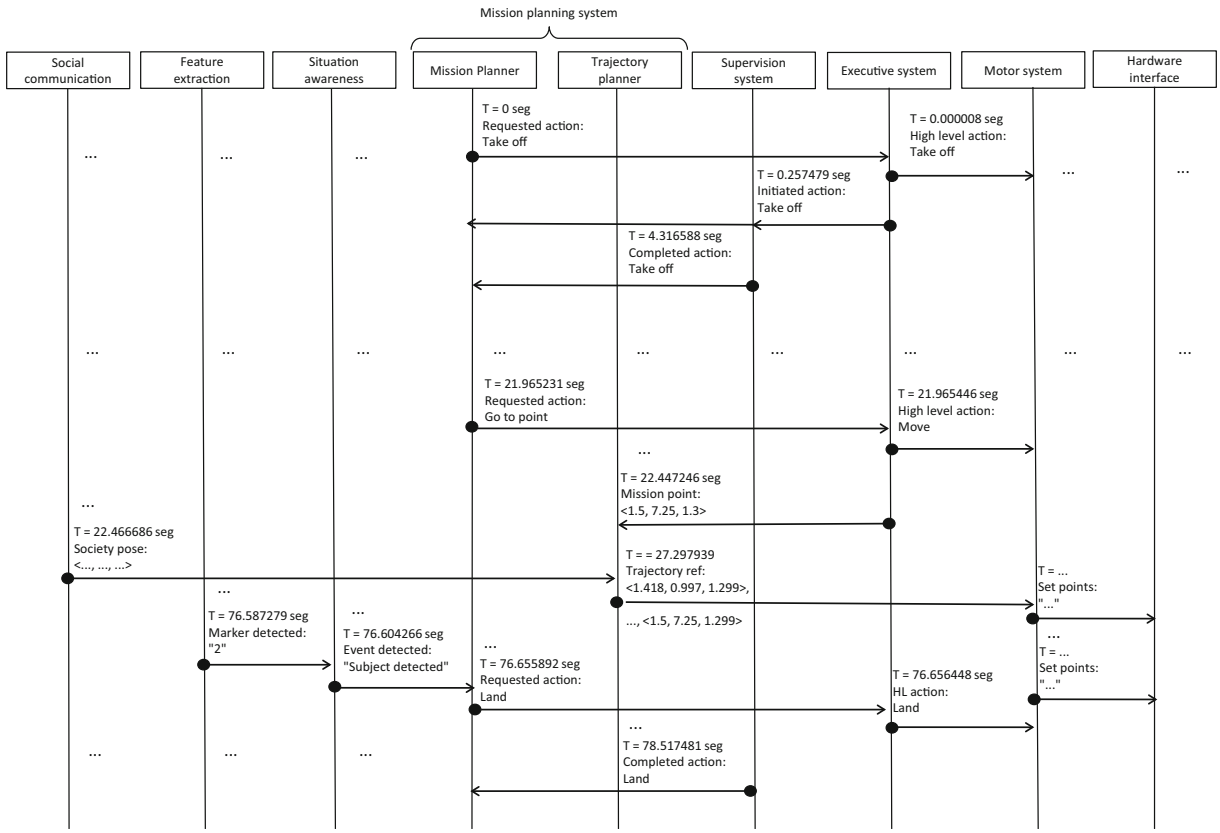



**Fig. 15** Example of inter-process communication among Aerostack systems. The figure shows a sample of low level ROS messages sent and received between systems corresponding to one of the complex experiments that we performed to evaluate Aerostack. The example shows the sequence of the messages using time stamps to illustrate the temporal delays

After the emergency situation is stabilized, the rescue aerial robot autonomously returns to the rescue team base (Figs. 12f and 14h).

Figure 15 shows the inter-process communication between some of the involved systems in the experiment. This example shows only a few messages for illustrative purposes. This figure shows, for example, the role that the supervision system plays to monitor the action execution. The example also illustrates how the event corresponding to the subject detection triggers the landing action by the mission planner. In the figure, the time stamps indicate (in seconds) the delay of the sequence of messages.

### 4.3 Evaluation Metrics

This section provides a number of quantitative evidences (about modular organization, processes, etc.) based on previous experiments that demonstrate some performance and quality features of the software framework.

Aerostack is a modular and specialized software with 89 software modules (defined as ROS packages), apart from multiple standard ROS packages. Aerostack organizes modules taking into account their functionality in different processes and sub-systems. For the previous example, it was necessary to use 22 software modules per robot agent. In addition, Aerostack also provides a modular organization of components (division in groups of software packages) according to the type of use and their level of dependency of ROS and other components of Aerostack. The previous flight experiment corresponds to a case of an application that uses the complete software framework.

As described, Aerostack uses asynchronous multitasking, where different processes run concurrently, with inter-process communication provided by ROS.

Depending on the application, one can execute from 10 to 50 processes simultaneously per robotic agent in a single computer or along multiple distributed computers. The computational needs are highly dependent on the specific implementation of the used modules, but, as stated in Section 4.1, the available components have run in very different computers. In the example presented on Section 4.2, one single agent was operated with 40 processes executed simultaneously and the ROS messages were published on about 180 different ROS topics. Even with this amount of processes and information exchanged, Aerostack worked fluidly and efficiently in real time.

The examples where Aerostack has been used also demonstrate other software quality features such as usability and scalability. Usability in Aerostack is provided by its modularity and a uniform documentation, both in source code and text documents. Aerostack counts also with manuals and tutorials that presents the main aspects of Aerostack with case of uses and examples ranging from basic users to developers.

The experience with Aerostack has also proved its scalability. In the last four years, since the original implementation in February 2013, Aerostack has grown gradually by including new components for more complex problems.

The first public release counted with 41 software modules, while the presented one has more than double this amount, 89 software modules. Currently, Aerostack is a live and evolving product supported by our academic team at the Technical University of Madrid that keeps updating the software framework and adding new components and functionalities.

## 5 Conclusions and Future Work

A fully autonomous operation of UAS is needed with the objective to simplify their use and to extend its utilization to a great number of applications. To solve this challenge, many open-source architecture frameworks for UAS have been developed, but they still present two main weakness: (1) in most of the cases, the acquired level of autonomy is limited, focusing on semi-autonomous missions. (2) versatility is typically restricted, being the available open-source architecture frameworks limited to some applications or aerial platforms.

To fill these gaps, this paper described Aerostack,[3] a system architecture and open-source multipurpose software framework for fully-autonomous single and multi-UAS.

Aerostack aims to help developers design their own implementation of their system architecture by having a reference model with a full specification of the required components. In addition, Aerostack provides a reusable open-source software framework formed by flight proven and ready to use executable software components and libraries which help developers to speed up the build process of their designed system.

Aerostack was firstly presented in [27], and is based on the authors' previous work [25, 29]. Compared to the initial publication of Aerostack, the main contribution of this paper is that it provides a deeper description of Aerostack, providing details of every subsystem of its architecture together with implementation examples which can be useful for researchers and developers for a more complete understanding of the framework. The paper also includes a more detailed discussion and justification of the framework organization and a more complete experiment to demonstrate the capabilities of Aerostack.

The formalization of the system architecture using the state of the art of intelligent, cognitive and social robotics knowledge, based on five layers: reactive, executive, deliberative, reflective, and social, confers Aerostack's architecture more autonomous capabilities and a higher level of versatility.

The open-source software framework of Aerostack includes the main components to execute the architecture for fully autonomous missions of swarms of aerial robots. This release includes as well a collection of components with two-dimensions modularity that can be used in specific environment conditions and mission requirements that allows the users and developers to have a fully autonomous swarm of aerial robots ready-to-use and flight-proven. The provided framework counts with the compatibility of five well known aerial platforms, as well as a high number of sensor interfaces. In addition, Aerostack's framework includes a documentation for basic users and developers, guiding them when using it; and also the support

of a multidisciplinary team of researchers at the Technical University of Madrid that is actively working with Aerostack, which ensures the continuous evolution and update of Aerostack.

Aerostack has been tested through four years (since February 2013) of successful use on research projects, international competitions and exhibitions. To confirm this, this paper presented Aerostack carrying out a fictional fully autonomous indoors search and rescue mission.

As Aerostack is alive because of its active use, two clear lines of future work exist: Firstly, the main components of Aerostack's software framework can be improved, making it more robust and efficient. Secondly, researchers can use Aerostack as is, limiting their contributions to new components with more functionalities than the existing ones, such as state estimators, controllers, planners, or computer vision algorithms among others.

**Acknowledgements** This research work has been partially supported by the Spanish Ministry of Economy and Competitiveness through the project VA4UAV (Visual autonomy for UAV in Dynamic Environments), reference DPI2014-60139-R. The authors would like to thank, as well, the Consejo Superior de Investigaciones Cientificas (CSIC) of Spain for the JAE-Predoctoral scholarships of one of the authors and his funded research stays.

The authors would like to thank other members from the Computer Vision and Aerial Robotics (CVAR) research group and the Department of Artificial Intelligence (UPM) for their help in software programming and the development of fight experiments: David Palacios, Adrian Diaz-Moreno, Guillermo de Fermín, Alberto Camporredondo and Carlos Valencia.

## References

1. Arkin, R.C., Riseman, E.M., Hanson, A.R.: Aura: An architecture for vision-based robot navigation. In: Proceedings of the DARPA Image Understanding Workshop (1987)
2. Brachman, R.J.: Systems that know what they're doing. IEEE Intell. Syst. **17**(6), 67–71 (2002). doi:10.1109/MIS.2002.1134363
3. Brisset, P., Drouin, A., Gorraz, M., Huard, P.S., Tyler, J.: The paparazzi solution. In: MAV 2006, 2nd US-European Competition and Workshop on Micro Air Vehicles (2006)
4. Davis, D.N.: Computational Architectures for Intelligence and Motivation. In: Proceedings of the 2002 IEEE International Symposium on Intelligent Control, 2002. IEEE (2002)
5. Duffy, B.R., Dragone, M., O'Hare, G.M.: Social robot architecture: a framework for explicit social interaction. In: Android Science: Towards Social Mechanisms, Cogsci 2005 Workshop, Stresa, Italy (2005)
6. Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., Marín-Jiménez, M.: Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recogn. **47**(6), 2280–2292 (2014). doi:10.1016/j.patcog.2014.01.005. http://www.sciencedirect.com/science/article/pii/S0031320314000235
7. Gat, E.: On three-layer architectures. In: Kortenkamp, D., Bonnasso, R.P., Murphy, R. (eds.) Artificial Intelligence and Mobile Robots, AAAI Press (1998)
8. Grabe, V., Riedel, M., Bulthoff, H., Giordano, P., Franchi, A.: The telekyb framework for a modular and extendible ros-based quadrotor control. In: 2013 European Conference on Mobile Robots (ECMR), 19–25 (2013). doi:10.1109/ECMR.2013.6698814
9. Kendoul, F.: A survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. J. Field Rob. **29**(2), 315–378 (2012)
10. Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., von Stryk, O., Klingauf, U.: Robocuprescue 2014-robot league team hector darmstadt (Germany). RoboCupRescue 2014 (2014)
11. Lim, H., Park, J., Lee, D., Kim, H.: Build your own quadrotor: Open-source projects on unmanned aerial vehicles. IEEE Robot. Autom. Mag. **19**(3), 33–45 (2012). doi:10.1109/MRA.2012.2205629
12. Van de Loosdrecht, J., Dijkstra, K., Postma, J., Keuning, W., Bruin, D.: Twirre: Architecture for autonomous miniuavs using interchangeable commodity components. In: IMAV 2014: International Micro Air Vehicle Conference and Competition 2014, Delft, The Netherlands, August 12–15, 2014, Delft University of Technology (2014)
13. Molina, M., Diaz-Moreno, A., Palacios, D., Suarez-Fernandez, R.A., Sanchez-Lopez, J.L., Sampedro, C., Bavle, H., Campoy, P.: Specifying complex missions for aerial robotics in dynamic environments. In: International Micro Air Vehicle Conference and Competition, IMAV 2016, Beijing, China (2016)
14. Murphy, R.: Introduction to AI robotics. MIT press (2000)
15. Pestana, J.: On-Board Control Algorithms for Quadrotors and Indoors Navigation Master's Thesis. Universidad Politécnica de Madrid, Spain (2012)
16. Pestana, J., Mellado-Bataller, I., Fu, C., Sanchez-Lopez, J.L., Mondragon, I.F., Campoy, P.: A General Purpose Configurable Navigation Controller for Micro Aerial Multirotor Vehicles. ICUAS (2013)
17. Pestana, J., Sanchez-Lopez, J., Campoy, P., Saripalli, S.: Vision based gps-denied object tracking and following for unmanned aerial vehicles. In: 2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), 1–6 (2013). doi:10.1109/SSRR.2013.6719359
18. Pestana, J., Mellado-Bataller, I., Sanchez-Lopez, J.L., Fu, C., Mondragón, I.F., Campoy, P.: A general purpose configurable controller for indoors and outdoors gps-denied navigation for multirotor unmanned aerial vehicles. J. Intell. Robot. Syst. **73**(1-4), 387–400 (2014)
19. Pestana, J., Sanchez-Lopez, J., de la Puente, P., Carrio, A., Campoy, P.: A vision-based quadrotor swarm for the participation in the 2013 international micro air vehi-

cle competition. In: 2014 International Conference on Unmanned Aircraft Systems (ICUAS), 617–622 (2014). doi:10.1109/ICUAS.2014.6842305
20. Pestana, J., Sanchez-Lopez, J., Saripalli, S., Campoy, P.: Computer vision based general object following for gps-denied multirotor unmanned vehicles. In: American Control Conference (ACC), 1886–1891 (2014) doi:10.1109/ACC.2014.6858831
21. Pestana, J., Sanchez-Lopez, J.L., de la Puente, P., Carrio, A., Campoy, P.: A vision-based quadrotor multi-robot solution for the indoor autonomy challenge of the 2013 international micro air vehicle competition. J. Intell. Robotic Syst. 1–20 (2015). doi:10.1007/s10846-015-0304-1
22. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software, vol. 3 (2009)
23. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Pearson Education (2003)
24. Sampedro, C., Bavle, H., Sanchez-Lopez, J., Suarez-Fernandez, R., Rodriguez, A., Molina, M., Campoy, P.: A flexible and dynamic mission planning architecture for uav swarm coordination. In: 2016 International Conference on Unmanned Aircraft Systems (ICUAS) (2016)
25. Sanchez-Lopez, J., Pestana, J., de la Puente, P., Suarez-Fernandez, R., Campoy, P.: A system for the design and development of vision-based multi-robot quadrotor swarms. In: 2014 International Conference on Unmanned Aircraft Systems (ICUAS) (2014). doi:10.1109/ICUAS.2014.6842308
26. Sanchez-Lopez, J., Pestana, J., Collumeau, J.F., Suarez-Fernandez, R., Campoy, P., Molina, M.: A vision based aerial robot solution for the mission 7 of the international aerial robotics competition. In: 2015 International Conference on Unmanned Aircraft Systems (ICUAS), 1391–1400 (2015). doi:10.1109/ICUAS.2015.7152435
27. Sanchez-Lopez, J., Suarez-Fernandez, R., Bavle, H., Sampedro, C., Molina, M., Pestana, J., Campoy, P.: Aerostack: An architecture and open-source software framework for aerial robotics. In: 2016 International Conference on Unmanned Aircraft Systems (ICUAS) (2016)
28. Sanchez-Lopez, J.L., Pestana, J., de la Puente, P., Carrio, A., Campoy, P.: Visual quadrotor swarm for the imav 2013 indoor competition. In: Armada, M.A., Sanfeliu, A., Ferre, M. (eds.) ROBOT2013: First Iberian Robotics Conference, Springer, Advances in Intelligent Systems and Computing, vol. 253 (2013). doi:10.1007/978-3-319-03653-3_5
29. Sanchez-Lopez, J.L., Pestana, J., Puente, P., Campoy, P.: A reliable open-source system architecture for the fast designing and prototyping of autonomous multi-uav systems: Simulation and experimentation. J. Intell. Robotic Syst. 1–19 (2015). doi:10.1007/s10846-015-0288-x
30. Singh, P., Minsky, M.: An architecture for cognitive diversity. Visions of mind: architectures for cognition and affect **312**, 166 (2005)
31. Sloman, A.: What sort of architecture is required for a human-like agent? In: Wooldridge, M., Rao, A. (eds.) Foundations of Rational Agency, Kluwer Academic Publishers (1999)
32. Suarez-Fernandez, R., Sanchez-Lopez, J., Sampedro, C., Bavle, H., Molina, M., Campoy, P.: Natural user interfaces for human-drone multi-modal interaction. In: 2016 International Conference on Unmanned Aircraft Systems (ICUAS) (2016)