AEROSTACK: An Architecture and Open-Source Software Framework for Aerial Robotics

Jose Luis Sanchez-Lopez[1], Ramón A. Suárez Fernández[1], Hriday Bavle[1], Carlos Sampedro[1], Martin Molina[2], Jesus Pestana[1], and Pascual Campoy[1]

***Abstract*— To simplify the usage of the Unmanned Aerial Systems (UAS), extending their use to a great number of applications, fully autonomous operation is needed. There are many open-source architecture frameworks for UAS that claim the autonomous operation of UAS, but they still have two main open issues: (1) level of autonomy, being in most of the cases limited and (2) versatility, being most of them designed specifically for some applications or aerial platforms.**

As a response to these needs and issues, this paper presents Aerostack, a system architecture and open-source multi-purpose software framework for autonomous multi-UAS operation. To provide higher degrees of autonomy, Aerostack's system architecture integrates state of the art concepts of intelligent, cognitive and social robotics, based on five layers: reactive, executive, deliberative, reflective, and social. To be a highly versatile practical solution, Aerostack's open-source software framework includes the main components to execute the architecture for fully autonomous missions of swarms of UAS; a collection of ready-to-use and flight proven modular components that can be reused by the users and developers; and compatibility with five well known aerial platforms, as well as a high number of sensors.

Aerostack has been validated during three years by its successful use on many research projects, international competitions and exhibitions. To corroborate this fact, this paper also presents Aerostack carrying out a fictional fully autonomous indoors search and rescue mission.

I. INTRODUCTION

Unmanned Aerial Systems (UAS), popularly known as drones, were historically used for missions that were too dull, dirty or dangerous for manned aircrafts [5]. Nevertheless, during the last years, a large number of new civilian and commercial applications have pushed the research and development of the UAS.

Within the group of the rotary-wing UAS, the multirotors characterized by their multiple fixed-pitch independent propellers have attracted the interest of the UAS community thanks to their simple, cheap and low maintenance design as well as to their high maneuverability that allows a four degrees of freedom movement with hovering and Vertical Take-Off and Landing (VTOL) capabilities.

*The authors would like to thank the Consejo Superior de Investigaciones Cientificas (CSIC) of Spain for the JAE-Predoctoral scholarships of two of the authors and their research stays, and the Spanish Ministry of Science MICYT DPI2014-60139-R for project funding, as well as the Spanish Ministry for Education, Culture and Sports for funding the international research stay of one of the authors".

[1]Computer Vision Group, Centre for Automation and Robotics, CSIC-UPM (Spain). {jl.sanchez, pascual.campoy}@upm.es. www.vision4uav.eu

[2]Department of Artificial Intelligence, Technical University of Madrid (UPM) (Spain)

Fig. 1: Parrot ARDrone 2 carrying out a fully autonomous emulated indoors search and rescue mission during an exhibition in the European Night of the Researchers (described in section IV) in front of an audience of 400 people using Aerostack. No Motion Capture System is used.

Many research groups and companies are focused on designing and building the hardware components that can be used on UAS. This paper is focused on the software and algorithms, assuming to have complete and well designed hardware for the UAS for any desired application.

To simplify the use of the UAS, reducing its cost of operation, as well as increasing its safety, a high level of automation is desired. Multiple challenges limit the current level of autonomy of the UAS for high-demanding applications such as localization and mapping on unstructured and changing environments; precise control of the aircraft with collision avoidance; trajectory and mission planning with a high level of cognition and intelligence; human-robot, environment-robot and robot-robot interaction; safety and fault tolerance among others. A large amount of research groups are working to solve these problems, although as separate and individual challenges. Only a limited number of them aim to develop a completely integrated solution for full autonomy, combining in a single architecture a number of highly interrelated and specialized components. Such an integration of components creates new challenges (e.g., efficient multi-tasking execution, adaptability to be used in different problems, scalability, etc.).

The software running on the computers onboard UAS can be classified in three main software components:

- Firmware: the time-critical management of the hard-

ware.
- Middleware: the time-critical control of the system.
- Operative System: the computer-intensive non time-critical control and management of the system.

Multiple commercial and open-projects exist that aim to develop complete software architectures (see [14] for a complete survey). To the best of the author's knowledge, the most well-known open-source commercial projects are the "PX4 Flight Stack[1]", and the "APM Flight Stack[2]". Both of them have the main software components of middleware and operative system and they are compatible with many UAS commercial hardware. Despite being able to achieve a good level of autonomy of the UAS, they are not designed to achieve a very high level of autonomy, being the operative system software components of the architecture very limited.

The activity of several research groups has produced some open-source architecture frameworks for UAS, being the most relevant ones, up to the author's knowledge:

- The "asctec_mav_framework"[3], developed by ASL - ETHZ, has a special focus on autonomous navigation of Ascending Technologies Aircrafts[4] and it is not compatible with any other aircraft platforms.
- The "hector_quadrotor" [13] framework, developed by HECTOR - TU Darmstadt, focused on heterogeneous cooperation for search and rescue (SAR) tasks.
- The "telekyb" [10] 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 change the architecture design for new capabilities.
- The "Paparazzi" [3] project, developed and used by ENAC and MAVLAV - 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.
- The "Twirre" [29] 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, more complex hybrid architectures able to provide more degree of autonomy and (2) versatility, 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 author's recent progress and main results in this line of research. In this paper, Aerostack[5] (a software framework for AErial RObotics) is described. The authors have recently named Aerostack to a framework that integrates and consolidates results of more than three years of research work on both research of components for aerial systems and their integration in efficient architectures [23] and [25]. As a result, we have created Aerostack as a mature, robust, reliable, documented, tested and validated software framework.

Compared to other architectures and frameworks, the main contribution of Aerostack is twofold: (1) a complete multi-layered architectural organization to support fully autonomous flights and (2) a versatile software framework for multiple uses.

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, Aerostack 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. Aerostack 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.

In the following sections, the paper presents and discusses both contributions in more detail. The remainder of the paper is organized as follows: section II presents the key aspects of Aerostack architecture. Section III describes the open-source software implementation of Aerostack architecture. In section IV some tests demonstrate the power and performance of Aerostack. Finally, section V concludes the paper and points out some lines of future work.

II. AEROSTACK SYSTEM ARCHITECTURE

Aerostack's architecture follows the hybrid reactive/deliberative paradigm, i.e., an architecture that integrates both a deliberative and reactive approaches [1] and [15]. The presented design includes five layers: reactive, executive, deliberative, reflective and social (Fig. 2).

The first three layers correspond to the popular hybrid design known as the three layer architecture [9] and [20]: (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

[1] http://px4.io/
[2] http://ardupilot.com/
[3] http://wiki.ros.org/asctec_mav_framework
[4] http://www.asctec.de/en/

[5] Aerostack webpage: http://www.aerostack.org/ and Aerostack Github repository: https://github.com/Vision4UAV/Aerostack

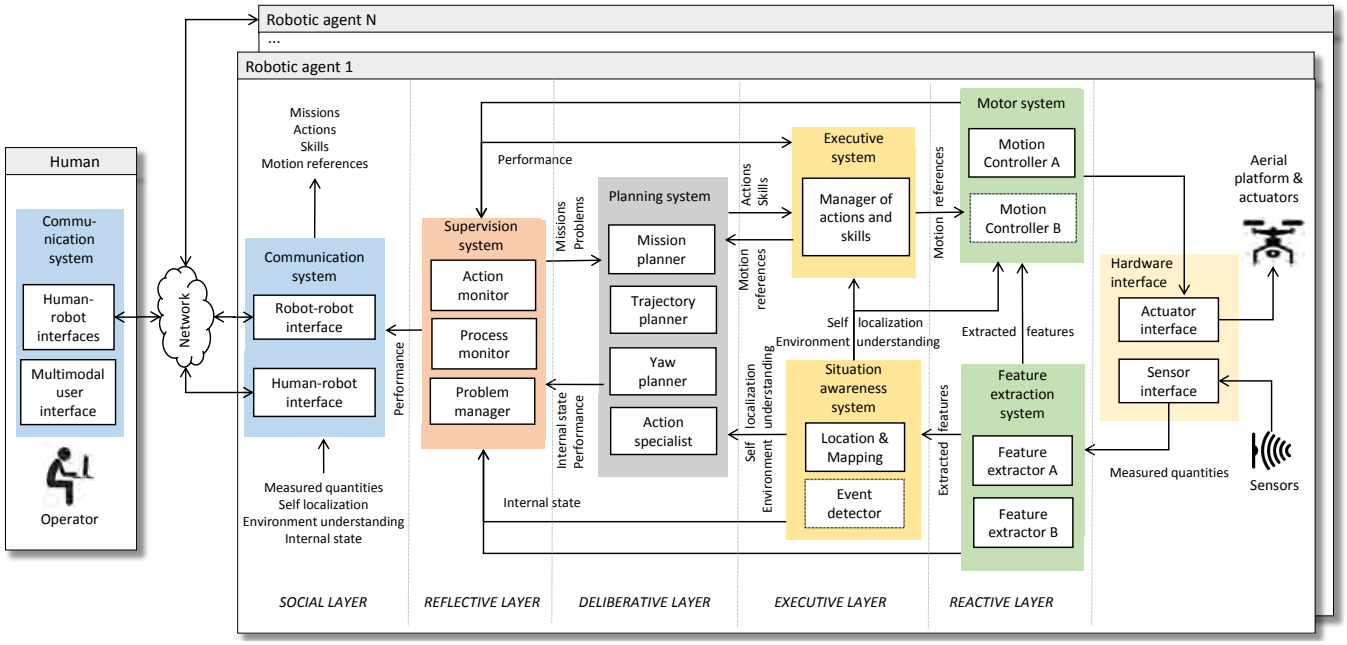

Fig. 2: Main components of the multi-layered architecture of Aerostack. The architecture is formed by N heterogeneous robotics agents and the human operators. Every robotic agent shares the same layered architecture, although it can have different component implementations as well as different hardware. 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.

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 vision and pattern recognition algorithms (signal processing, recognition of objects and basic relationships). Examples of feature extractors are: read bumper, extract color, compute centroid of an image, recognize visual marker, detect power line tower, etc. A feature extractor can integrate a set of perception procedures using different combination methods (e.g., fusion, sequence, etc.).

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 [27], [6], [2], [26] to simulate certain self-awareness able to supervise the other layers. The reflective layer helps to see if the robot is actually 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., [8]). In this level is important to establish an adequate communication with human operators and other robots.

The architecture is also consistent with the usual components related to guidance, navigation and control of unmanned rotorcraft systems [12]. 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.

A. Ontology for Aerial Robotics

In order to facilitate the semantic interoperability of the different components, we use an ontology (see table I) for aerial robotics that 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 concepts are classified according to the input/output categories that Fig. 2 shows. The current

formalization of this ontology is based on common data representations. A complete formal specification of this ontology using an appropriate language (e.g., OWL) is a pending task to be done in the future.

For example, the notion of skill is useful as an intuitive concept to help operators express more easily what complex abilities should be active for a particular robot. A skill is automatically translated to a set of running processes. Skills differ from actions in that an action (e.g., take-off, go to a point) finishes by itself when it reaches the goal, and a skill (e.g., interpret visual commands, avoid obstacles), once it is activated, it is permanently active without any limit of time, until it is deactivated by an external influence (e.g., the mission planner, the operator, etc.).

Concept	Description
Measured quantity	Values corresponding to direct measurements recorded by sensors. Aerostack uses platform- and sensor-independent parameters whose values are obtained with the corresponding interfaces
Extracted feature	Single features extracted from images and 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.
Environment understanding	This includes the characteristics of the environment and its elements as they are believed by the drone. For example: walls, pole obstacles, other robots, etc.
Self localization	Drone localization in the environment with its kinematic values (e.g., speeds) as they are believed by the drone. For example: pose, speeds, distance to obstacles, etc.
Internal state	States of the drone (e.g., landed/flying/hovering, emergency, armed, autonomous/manual, etc.), states of its components and state of communications (e.g., online/offline), perception states (a tracked object is inside the bounding box), etc.
Mission	Complex goal to be done by the drone (e.g. search an object in a field, deliver a parcel, etc.). It can be specified by human operators with a set of tasks and subtasks that describe the different parts of the mission to be done.
Action	Elementary task that the drone is able to perform, for example: take-off, land, go to point, rotate yaw, flip, wait, etc. We assume that actions are mutually exclusive, i.e., only one action can be performed at any given moment.
Skill	A skill expresses a particular drone's ability, for example: avoid obstacles, limit extreme movements, interpret visual markers, etc. Skills can be active or inactive in a particular drone.
Motion reference	Motion values to be considered as goals by controllers or planners. Examples of references are: position, speed, or yaw.
Problem	A problem corresponds to an undesired situation related to: unachievable goal due to unexpected changes in the environment, a software error or a hardware fault.
Performance	The operational performance of the computational processes includes the execution state of processes and unexpected execution errors.

TABLE I: Example concepts of the ontology for aerial robotics to facilitate the semantic interoperability of the different components of Aerostack's architecture.

B. Autonomy and self adaptation in complex environments

As was mentioned before, Aerostack includes components for a reactive behavior that provides certain level of autonomy (some previously published papers describe details of such components [23]). In addition, Aerostack also includes components to increase the degree of autonomy to operate in complex and dynamic environments. This section presents in more detail such components.

In general, fully autonomous robots are able to accomplish their assigned mission without human intervention while adapting to operational and environmental condition [11]. Different degrees of autonomy can be identified [4], [12] that require to simulate cognitive tasks. This includes dimensions such as the following [11]:

- Human independence. The robot can be operated with simple commands from general human operators and does not require highly specialized operators and technical jargon.
- Dynamic environment. The robot can operate in dynamic and complex environments with unexpected situations where it is required abilities such as: self adaptation, threat avoidance, self-diagnosis, fault tolerance, etc.
- Complex missions. The robot can perform complex missions (such as search and rescue missions) where situation awereness and complex planning is required.

In particular, Aerostack provides the following components related to these levels of autonomy:

- The *planning system* automatically generates the goals in order to accomplish a particular mission. Aerostack includes a task-oriented mission planner and other more specific planners (trajectory planner and yaw planner). This planning system helps to increase autonomy in terms of human independence, complex missions and self adaptation in dynamic environments.
- The *executive system* includes a behavior manager that accepts directives from the deliberative layer and sequences them to be performed by the reactive layer. The behavior manager translates requested actions and behaviors expressed as symbolic descriptions (e.g., take-off, move to a point, etc.) into specific orders for the motion controllers and the activation of certain running processes. The executive layer helps to increase autonomy in terms of human independence and complex missions.
- The *supervision system* is a key functional package that ensures a correct behavior of the robot. The supervision system helps to provide self adaptation and fault-tolerance. This typically consists of three steps: failure detection, notification, and recovery. The supervision system includes the following processes: the action monitor and the process monitor for failure detection and notification, and the problem manager for fault recovery. The supervision system helps to increase autonomy in dynamic environments providing self adaptation and fault tolerance.

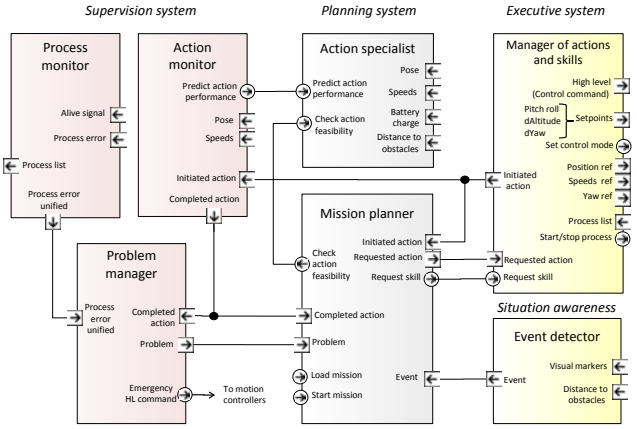

Fig. 3: Detail of processes used in Aerostack related to the supervision system. This diagram shows some components of the reflective, deliberative, and executive layers and their interconnectivity.

Using Aerostack, the operator can specify in advance a mission using a xml-based language and the mission planner interprets such a specification to generate step by step actions to be done. Event handlers can be defined using condition-action rules to react in the presence of certain exceptional situations. Exceptions can change the execution of a mission by requesting additional actions or skills and by using particular commands (abort mission, abort task, etc.). Aerostack's mission planner follows the approach called reactive planning that differs from classical planning in the way that it computes just one next action in every instant, based on the internal state (about goals, tasks, etc.) and the external world state [7]. This simplification is useful to cope with highly dynamic and unpredictable environments.

Within the supervision system (see Fig. 3), the action monitor supervises the execution of a requested action and informs when the action has been completed or failed. This is important to simulate self-awareness of the degree of completion of goals to be able to notify it to the adequate destination (to the operator, to the mission planner, etc.). The process monitor supervises the execution of different processes and is responsible for acquiring and informing about the errors produced by such processes. The process monitor verifies that each process is alive using a watchdog technique and to get the states of processes. The process monitor collects errors produced by processes in order to notify in a uniform way these errors to other components.

Aerostack also includes a problem manager that collects the errors and unexpected events, notifies them to the appropriate processes and initiates the corresponding recovery actions (if possible). In order to manage faults, we assume the following taxonomy of abnormal situations:

- Unachievable goal. This happens when a requested action (from the mission planner) cannot be performed under a correct operation. The main reasons for this are unexpected changes in the environment (e.g., strong wind, too obscure or too much light, an unexpected obstacle, poor quality of wifi connection, etc.) or wrong planning assumptions (e.g., the point to reach is too close to static known obstacles). The problem manager detects such problems with the action monitor and the situational awareness system (flight monitor), and notifies these problems to the mission planner and the operator to initiate recovery actions.
- Software error. This corresponds to errors such as: invalid input data, a process is down unexpectedly, safeguard software error (internally, the execution of a process detects an error caused by wrong programming assumptions or programming mistakes), a process is taking more time than expected (an infinite loop due to programming errors). The problem manager detects such errors with the performance monitor. In this case, the main recovery action is to notify this error to the operator with an appropriate error message to help the developers correct the problem.
- Hardware malfunction. This situation happens when hardware components (camera, gps sensor, propeller, etc) are broken or are working partially. These type of faults can be classified into the following categories [18]: (1) actuator faults, i.e., partial or total loss of actuators control action, (2) sensor faults, i.e., incorrect readings from sensors and (3) component faults, i.e., faults in the component of the robot.

In general, for hardware malfunctions, different recovery actions can be performed such as switching hardware components (using an alternative hardware component), dynamically readjust controllers when some of the physical actuators (e.g., a propeller) are broken, managed by fault-tolerant control techniques [21] and emergency land.

III. AEROSTACK SOFTWARE FRAMEWORK

The presented Aerostack's architecture has been implemented as an open-source software framework, which allows users and developers to have a flight-proven multi-aerial collection of ready-to-use components. Thanks to this open-source software framework, the development process of new systems is sped up, allowing developers to test their algorithms even in early stages of the project.

The main features of Aerostack's software framework (detailed in the following sections) are the synthesized in the following:

- Fully Autonomous Operation. A fully autonomous aerial system can be set up based on Aerostack.
- Multirobot swarming possibilities. Capability of realizing multi-aerial-robot missions.
- Modularity. Its two dimensional modularity arranges the components by their functionality as well as by their level of dependency.
- Scalability. It implements separately common processes and optional processes. This allows to modify or develop new optional processes without the need of changing any other process.
- Versatility. Developers are able to build new optional processes easily by using the available common pro-

cesses and libraries in addition to the well-defined internal messages and interfaces.
- Distributed processing. It allows the execution of the components both in one single computer or distributed in many computers.
- Compatibility with various multi-rotor platforms and sensors through the usage of a well specified interface.
- Flight proven and ready-to-use components with the capability of running simulations on big parts of the developed architectures.

A. Multi-process modular organization

One of the main characteristics of Aerostack is modularity. This allows to create independent modules with specific functionalities which can be exploited once connected to the rest of the architecture. This modularity allows the individual testing of modules, easing the project progress. Also, understanding the modules as input-output systems permits to test in simulation the compatibility of their interfaces with the full system instance at hand.

A basic module of Aerostack's architecture is the process. We distinguish between the intuitive meaning of a process (what it represents) and its computational support (how it is implemented). A process acts through time to change certain parameters of objects (e.g., certain physical quantities), consuming resources (e.g., time or space) to convert inputs into outputs. It is important to note that a process has a function, i.e., a purpose or practical use for which the process is designed or exists (the intention or the objective of the process). The idea of process is an appropriate concept to describe the components of the architecture of Aerosack. It helps to divide the whole problem of automated support for UAS into partial functional roles. Each process in Aerostack is named as an active processor with its functional role, 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 idea of a process is also similar to the concept of an atomic functional block used in SysML with input/output ports. Processes are grouped in systems. A system is a complex module that includes a set of interconnected processes that provides a common functionality. The idea of system is also similar to the concept of functional block (composite block) used in SysML, with input/output ports.

Aerostack is implemented as a multi-process framework supported by ROS (Robot Operating System) [19]. A process in Aerostack corresponds to the concept of a ROS node. Each process that operates in a particular UAS is programmed as a subclass of a common class called "DroneProcess" which provides default routines. When Aerostack is running for a particular UAS, each process is an executable instance of a program running in a computer (each process can also call subprocesses, child computational processes).

Aerostack uses asynchronous processing techniques (e.g., multitasking) that allow deliberative functions to execute independently of reactive behaviors. Planning algorithms can be computationally expensive, so they must be decoupled from real-time execution and avoid slow down the reaction time. In general, the inter-process communication (IPC) in multitasking operating systems, where different processes run concurrently, admits different communication mechanisms such as pipes, message queues, semaphores, sockets, shared memory, etc. Aerostack uses communication methods provided by ROS, mainly: (1) topics (processes can communicate with each other by publishing messages to topics) and (2) services (a communication mechanism between processes based on a request-reply scheme). Taking advantage of ROS features, Aerostack can perform a distributed processing, running its components both in one single computer or distributed in many computers and being only limited by the hardware.

B. Division in groups of software packages

Fig. 4 shows that the Aerostack framework is divided into seven groups of software packages. This modular division has been done following practical reasons according to different possible uses of the framework:

- The group of packages *Aerostack library* includes software modules that implement specific algorithms (e.g., computer vision algorithms, SLAM algorithms). This corresponds to ROS-independent programs that can be reused directly by developers to build new software for robotic platforms.
- The group of packages called *Aerostack ROS library* includes software modules that implement specific algorithms. This corresponds to ROS-dependent programs that can be reused directly.
- *Aerostack core* gathers the necessary common software to extend the functionalities of any process to a "DroneProcess", as well as the definition of Aerostack custom messages. It also includes Aerostack-dependent helper packages. It is ROS-dependent.
- *Aerostack common processes* corresponds to the necessary common software to articulate a complete multi-layered architecture. This includes the supervision system, the executive system and the communication system (with the human machine interface HMI).
- *Aerostack optional processes* includes alternative processes related to the following functional modules of the multi-layered architecture: the planning system, the situation awareness system, the feature extraction system and the motor system. For example, Aerostack provides optional modules for planning (e.g., trajectory planners), feature extraction (e.g., Aruco visual markers) and motion controllers (e.g., trajectory controllers or visual servoing controllers). This software implements the interfaces between the general algorithms of the library, ROS library groups of packages and Aerostack modules, as well as extending "DroneProcess" capabilities.

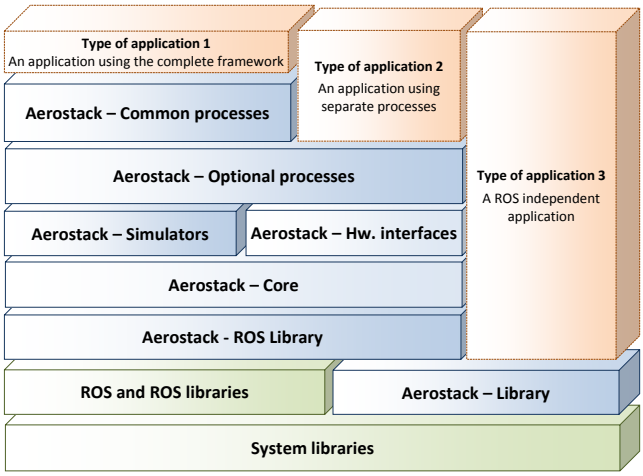

Fig. 4: Software packages corresponding to the Aerostack framework. The seven groups of software packages can be observed: library, ROS library, core, common processes, optional processes, simulators, and hardware interfaces.

- *Aerostack hardware interfaces* includes the corresponding software interfaces for different aerial platforms, sensors and actuators. For example, Aerostack includes interfaces for AscTec Autopilot, Parrot AR Drone, Parrot Bebop, Mikrokopter Flight Controller as well as MavLink. This also includes interfaces for sensors such as UEye cameras and Optical Flow Sensors (Px4Flow), Hokuyo LIDARs and standard GPS, among others.
- *Aerostack simulators* includes simulation modules that are useful to test the correct behavior of the full system or specific modules before a field test. They include physics engines to ensure realism on the simulations.

C. System Requirements

This section briefly enumerates the system requirements to execute Aerostack. It requires a computer with Ubuntu Linux 14.04 or above with ROS Indigo or Jade installed. The preferred programming languages are C++ (some packages make use of C++11 standard) and Python.

Aerostack has been run in a wide variety of computers, ranging from the micro-computer ODROID-XU3 (a 2.0 GHz ARM quad core processor, 2 GB of memory), the micro-computer AscTec Mastermind (Intel Core i7 processor, 4 Gb of memory), and average laptops (Intel Core i5 processor, 4 Gb of memory).

D. Use Cases

This section describes different examples of use cases that show de high level of versatility of Aerostack as well as its scalability.

Users can take advantage of Aerostack to operate UAS platforms in the following way:

- Perform tele-operated flights controlled by user commands using the Human Machine Interface. Examples of user commands are: take-off, move forward, move backward, turn, move up, move down, land, etc. The user operates with the keyboard, the joystick, and the mouse of the ground station computer to control the flight of the UAS.
- Tracking a selected object by visual contact. The user can select the object to track by selecting camera images using the Human Machine Interface.
- Following a specific mission plan, defined by the user as a set of tasks or a set of waypoints.
- Flying within a specific spatial area, with certain limits defined by the user.
- Use a specific physical configuration that modifies significantly the size or weight of the UAS. Our flight controllers can be adapted by the user to the new configuration (in a XML file).

Developers who are familiar with software programming can use Aerostack to operate new UAS with the following characteristics:

- A UAS with new specific types of sensors. The developer can reuse Aerostack but the developer needs to program and integrate new software modules to process the information from the new sensors.
- A UAS with a different physical platform. The developer can reuse Aerostack for different UAS platforms, but the developer needs to program the interfaces between Aerostack and the actuators of the new physical platform.
- A UAS with one or several software components that substitute an existing software component (for example, a new localization and mapping) but have the same inputs/outputs as the previous one. The developer can reuse Aerostack, but the developer might need to substitute the core algorithm used by the previous component within the package of optional processes.
- A UAS with one or several software components that substitute an existing software component (for example, a new localization and mapping) that have a different inputs/outputs map than the previous one. The developer can reuse Aerostack, but the developer might need to program and integrate a new software component within the package of optional processes. Additionally, the developer might need to extend the functionalities of the connected components to be consistent with the new inputs/outputs map.
- A UAS with new functionalities that needs additional software components. The developer can reuse Aerostack but the developer needs to program and integrate the new software modules. Additionally, the developer might need to extend the functionalities of the connected components to extend the functionalities of Aerostack.

It is important to emphasize that any of the available components of Aerostack can be replaced by a similar counterpart at the simple cost of developing an Aerostack interface (e.g. the trajectory controllers given by the previously mentioned “PX4 Flight Stack” could be used withing Aerostack architecture).

IV. AEROSTACK EVALUATION

Aerostack is fully operative, validated since its birth in February 2013, by simulations and real flight tests with multiple UAS (more than 5) flying simultaneously, and with five different aerial platforms equipped with diverse sensors. In the following sections, we describe additional evidences related to the Aerostack evaluation and its experimental and practical use.

A. Reported uses of Aerostack

Aerostack has been used from its first pre-release, in the development of many different projects, including two UAS competitions: IMAV 2013 [16], and IARC 2014 [24], both with successful results. In addition, Aerostack has been employed in shows and exhibitions for both general and specialized public, highlighting the European Night of the Researchers[6] (see Fig. 1) which was attended by more than 400 people.

Simultaneously, Aerostack has been used as the main framework for other research activities: In [17], using Aerostack, the authors demonstrated that a fully autonomous UAS was able to follow any object using computer vision. In [22], the authors expanded Aerostack capabilities to demonstrate the benefit of using a coordinator to accomplish high-level missions requested by the user with a fully autonomous swarm of UAS. In [28] Aerostack was used for research and development of Natural User Interfaces for Human-Drone Interaction using hand gestures, speech, body movements and visual cues.

Finally, Aerostack is also being used in other ongoing projects like the participation on the IMAV 2016 Indoors Competition.

B. Example of Real Operation

To demonstrate the operation of Aerostack, apart from the cited works in section IV-A, a fully autonomous mission has been executed using Aerostack.

A search and rescue mission is emulated. An autonomous swarm of UAS departs from the rescue equipment base. The rescue equipment operators have previously defined the regions where they wanted the drones to search for the victim. The drones autonomously navigate to these areas avoiding collisions with other obstacles. Once the victim is detected by a drone, it starts to track it, reacting to natural commands from the victim. Once the victim gives the "I am Ok" command, the drones autonomously return to rescue equipment base. The rescue equipment operators are supposed to be non-qualified personnel (in the use of the drones, for example health-care workers), therefore a simple GCS is needed with Natural User Interfaces.

This proposed search and rescue mission is simplified to be able to make use of the available components of Aerostack. The simplified mission (see Fig. 5) is described as follows: a single autonomous UAS takes-off from a predefined point known as "Home". After the take-off, the drone navigates along pre-defined waypoints through a structured indoors environment with pole obstacles identified by visual markers. Once the drone arrives to the last target point it hovers until it receives a command from the ground control station (GCS) to start the visual following task. During this task, the drone visually tracks a person. If the person goes out of the field of vision of the drone, the drone hovers and informs the GCS of the loss. If the person comes again inside the field of vision of the drone, it automatically restarts the visual tracking. Additionally, during this task, the drone is able to receive visual commands from the person through visual markers. The drone reacts to these visual markers commands by doing flips. In the last place, the person shows the drone a visual marker that orders it to go back to the "Home" point. Then, the drone navigates through the obstacles until it arrives to the "Home" point and finally lands. All the tasks done by the drone have voice feedback. The GCS counts with a Graphical User Interface to supervise and monitor the drone. Due to a limitation on the flight area size, a single drone is used instead of a swarm. Additionally, for simplicity, a Parrot ARDrone 2 is used as the aerial vehicle, which has no extra payload capabilities. Note that no Motion Capture System is employed.

As it can be pointed out, there is a simplification on the complexity of the environment and the tasks, but not in the requirements of the mission and in the level of autonomy required by the drone, which allows to demonstrate the capabilities of Aerostack.

In the following link `https://youtu.be/8WtVDaJADsA` the reader can find a video with the full mission. This mission is similar to the one executed in exhibitions like the European Night of the Researchers (Fig. 1) or the Spanish national congress Civil Drone 2016 with a total audience of more than six hundred people, without fails or crashes.

C. Evaluation Metrics

This section provides a number of quantitative evidences (about modular organization, processes, etc.) based on the 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 44 software modules. 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 (see Fig. 4). 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

[6]Webpage: `http://www.madrimasd.org/lanochedelosinvestigadores/actividad/vuelo-de-drones-y-m%C3%A1s-robots-asombrosos?lan=en`

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 III-C, the available components have run in very different computers. In the example presented on section IV-B, Aerostack was running on a single computer with an Intel i7-2620M processor and 8 Gb of memory. One single agent was operated with 39 processes executed simultaneously and 158 ROS message topics were published. Even with this huge 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. For example, the usability has been demostrated with the fact that more than 30 known users and developers (different from the Aerostack developers) have succesfully used the software framework in different practical applications. 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.

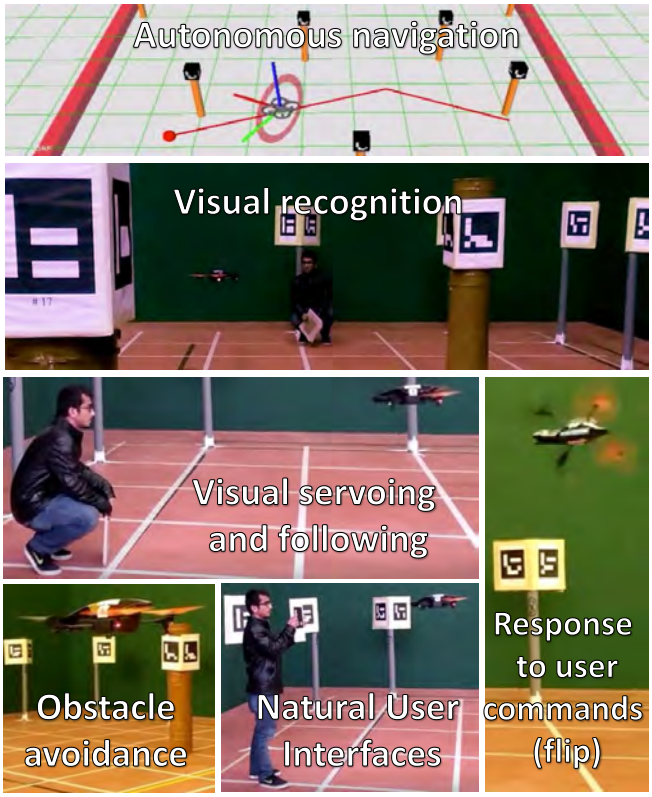

Fig. 5: Snapshots of the fictional indoors search and rescue autonomous mission carried out by a Parrot ARDrone 2 using Aerostack. The UAS is able to navigate avoiding obstacles while planning collision-free trajectories, and to visually recognize and track a person. In addition, the subject is able to interact with the drone using a Natural User Interface, in this case, visual markers, and the drone responds by executing a flip and returning to the "Home" point.

The experience with Aerostack has also proved its scalability. In the last three years, since the original implementation, 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.

V. Conclusions and Future Work

It has been demonstrated that 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 presented Aerostack[7], 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 is based on the author's previous work [23], being the contributions of this paper twofold. On the one hand, the evolution of the system architecture from its control-oriented definition towards a description based on the state of the art of intelligent, cognitive and social robotics, based on five layers: reactive, executive, deliberative, reflective, and social. This formalization confers Aerostack's architecture more autonomous capabilities and a higher level of versatility. On the second hand, a more mature version of the open-source software framework of Aerostack is presented. This software framework includes the main components to execute the architecture for fully autonomous missions of swarms of UAS. 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 UAS ready-to-use

[7]Aerostack webpage: `http://www.aerostack.org/` and Aerostack Github repository: `https://github.com/Vision4UAV/Aerostack`

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. Last but not least, 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 three years 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, three clear lines of future work exist: On the one hand, Aerostack architecture can be improved by incorporating deeper details in the intelligent, cognitive or social layers of the architecture. Secondly, the main components of Aerostack's software framework can be improved, making it more robust and efficient. Finally, 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.

REFERENCES

[1] R. C Arkin, E. M Riseman, and A. R Hanson. Aura: An architecture for vision-based robot navigation. In *proceedings of the DARPA Image Understanding Workshop*, pages 417–431, 1987.

[2] R. J. Brachman. Systems that know what they're doing. *IEEE Intelligent Systems*, 17(6):67–71, Nov 2002.

[3] Pascal Brisset, Antoine Drouin, Michel Gorraz, Pierre-Selim Huard, and Jeremy Tyler. The paparazzi solution. In *MAV 2006, 2nd US-European Competition and Workshop on Micro Air Vehicles*, 2006.

[4] B. T Clough. Metrics, schmetrics! how the heck do you determine a uav's autonomy anyway. Technical report, DTIC Document, 2002.

[5] C. Cuerno Rejado, L. Garcia Hernandez, A. Sanchez Carmona, A. Carrio Fernandez, J. L. Sanchez-Lopez, and P. Campoy Cervera. Evolution of the unmanned aerial vehicles until present. *DYNA DYNA-ACELERADO*.

[6] D. N Davis. Computational architectures for intelligence and motivation. In *Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on*, pages 520–525. IEEE, 2002.

[7] J. Downs and H. Reichgelt. *European Workshop on Planning: EWSP '91, Sankt Augustin, FRG, March 18–19, 1991 Proceedings*, chapter Integrating classical and reactive planning within an architecture for autonomous agents, pages 13–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[8] B. R. Duffy, M. Dragone, and G. MP OHare. Social robot architecture: A framework for explicit social interaction. In *Android Science: Towards Social Mechanisms, CogSci 2005 Workshop, Stresa, Italy*, 2005.

[9] E. Gat. On three-layer architectures. In David Kortenkamp, R. Peter Bonnasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots*. AAAI Press, 1998.

[10] V. Grabe, M. Riedel, H.H. Bulthoff, P.R. Giordano, and A. Franchi. The telekyb framework for a modular and extendible ros-based quadrotor control. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 19–25, Sept 2013.

[11] H.M. Huang. Autonomy levels for unmanned systems (alfus) framework volume i: Terminology version 2.0. *National Institute of Standards and Technology (NIST). Special Publication 1011-I-2.0*.

[12] Farid Kendoul. A survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 29(2):315–378, 2012.

[13] Stefan Kohlbrecher, Johannes Meyer, Thorsten Graber, K Petersen, Oskar von Stryk, and U Klingauf. Robocuprescue 2014-robot league team hector darmstadt (germany). *RoboCupRescue 2014*, 2014.

[14] Hyon Lim, Jaemann Park, Daewon Lee, and H.J. Kim. Build your own quadrotor: Open-source projects on unmanned aerial vehicles. *Robotics Automation Magazine, IEEE*, 19(3):33–45, Sept 2012.

[15] Robin Murphy. *Introduction to AI robotics*. MIT press, 2000.

[16] J. Pestana, J. L. Sanchez-Lopez, P. de la Puente, A. Carrio, and P. Campoy. A vision-based quadrotor multi-robot solution for the indoor autonomy challenge of the 2013 international micro air vehicle competition. *Journal of Intelligent & Robotic Systems*, pages 1–20, 2015.

[17] J. Pestana, J.L. Sanchez-Lopez, S. Saripalli, and P. Campoy. Computer vision based general object following for gps-denied multirotor unmanned vehicles. In *American Control Conference (ACC), 2014*, pages 1886–1891, June 2014.

[18] X. Qi, D. Theilliol, J. Qi, Y. Zhang, J. Han, D. Song, L. Wang, and Y. Xia. Fault diagnosis and fault tolerant control methods for manned and unmanned helicopters: A literature review. In *Control and Fault-Tolerant Systems (SysTol), 2013 Conference on*, pages 132–139. IEEE, 2013.

[19] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[20] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[21] I. Sadeghzadeh and YM Zhang. A review on fault-tolerant control for unmanned aerial vehicles (uavs). *Infotech@ Aerospace, St. Louis, MO*, 2011.

[22] C. Sampedro, H. Bavle, J.L. Sanchez-Lopez, R. Suarez-Fernandez, A. Rodriguez, M. Molina, and P. Campoy. A flexible and dynamic mission planning architecture for uav swarm coordination. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, page 0, June 2016.

[23] J. L. Sanchez-Lopez, J. Pestana, P. Puente, and P. Campoy. A reliable open-source system architecture for the fast designing and prototyping of autonomous multi-uav systems: Simulation and experimentation. *Journal of Intelligent & Robotic Systems*, pages 1–19, 2015.

[24] J.L. Sanchez-Lopez, J. Pestana, J.-F. Collumeau, R. Suarez-Fernandez, P. Campoy, and M. Molina. A vision based aerial robot solution for the mission 7 of the international aerial robotics competition. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 1391–1400, June 2015.

[25] J.L. Sanchez-Lopez, J. Pestana, P. de la Puente, R. Suarez-Fernandez, and P. Campoy. A system for the design and development of vision-based multi-robot quadrotor swarms. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 640–648, May 2014.

[26] P. Singh and M. Minsky. An architecture for cognitive diversity. *Visions of mind: architectures for cognition and affect*, 312:166, 2005.

[27] A. Sloman. What sort of architecture is required for a human-like agent? In M. Wooldridge and A. Rao, editors, *Foundations of Rational Agency*. Kluwer Academic Publishers, 1999.

[28] R. Suarez-Fernandez, J.L. Sanchez-Lopez, C. Sampedro, H. Bavle, M. Molina, and P. Campoy. Natural user interfaces for human-drone multi-modal interaction. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, page 0, June 2016.

[29] J Van de Loosdrecht, K Dijkstra, JH Postma, W Keuning, and D Bruin. Twirre: Architecture for autonomous mini-uavs 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.