Engineering a Platform for Mission Planning of Autonomous and Resilient Quadrotors

Davide Di Ruscio[1], Ivano Malavolta[1], and Patrizio Pelliccione[1,2]

[1]University of L'Aquila, Information Engineering, Computer Science and Mathematics Department, L'Aquila, Italy
[2]Chalmers University of Technology | University of Gothenburg, Department of Computer Science and Engineering, Sweden

{davide.diruscio,ivano.malavolta,patrizio.pelliccione}@univaq.it

Abstract. Quadrotors and UAVs in general are becoming as attractive instruments to safely and efficiently perform environmental monitoring missions. In professional use, quadrotors are manually controlled by expert operators via a remote controller. In research, several projects provide various degrees of automation for the execution of the mission; however, those projects are based on the use of programming languages which are too distant from the background of the stakeholders operating in the field (e.g., fire fighters, policemen, etc.).
In this paper we propose FLYAQ, a platform enabling to (i) graphically define monitoring missions via a web interface, (ii) decompose the mission according to the number and nature of available quadrotors, and (iii) generate the implementation code orchestrating all the quadrotors of the swarm to fulfil the common goal of the mission. The FLYAQ platform enables operators to focus on the mission itself, rather than on technical concerns arising from the use of quadrotors.
A reconfiguration engine is specifically designed to make the swarm resilient to faults and external events that may compromise the mission. Moreover, under some limitations explained in the paper, the reconfiguration engine permits to change the mission at run-time. The FLYAQ platform will be distributed as an open-source product.

1 Introduction

There are several papers that propose approaches to program quadrotors and to make them autonomous in the management of environmental missions [1, 2]. The academic leader in this field of research is Penn, University of Pennsylvania, Philadelphia - United States, along with other research groups all over the world. In general, quadrotors are programmed with a very low level language or provide very basic primitives [3–5]; this issue introduces an "error-prone" process even for experienced users and asks the programmer to deeply know the dynamics and the technical characteristics of the used quadrotors. It also makes the specification of missions unaffordable for a non-technical user, who has typically a good experience in the domain of environmental monitoring missions, but a very poor (if any) experience in software programming. This technological barrier hampers the technology transfer.

By considering such observations, in this paper we propose FLYAQ[1], an open source platform supporting the specification and the execution of environmental monitoring missions. In particular, the platform offers a graphical language to specify missions in the ground station at a high-level of abstraction. Integration with Open Street Map[2] allows on-site operators to view the various geographical points to be monitored. The mission will be then automatically decomposed by the platform in order to configure the involved quadrotors in order to accomplish the mission. A combination of model transformations, code generation, and formal reasoning will be used to automatically transform the mission specified in the FLYAQ specification language in low-level instructions provided by the platform of the quadrotor. The specification language of FLYAQ allows end users (with limited IT knowledge, but domain experts in environmental monitoring missions) to plan missions straightforwardly. Furthermore, since the platform has a direct integration with the map, it will be able to automatically compute the traits of flight.

The FLYAQ platform is especially designed to maintain a desired degree of resilience of the system, i.e., to ensure that the system will persistently deliver its services in a trustworthy way even when facing changes and unforeseen failures. A reconfiguration component has been conceived to this purpose and will exploit the information produced during design (such as characteristics of the used quadrotors, wind conditions, obstacles in the map, etc.) and gathered at run-time to guide run-time adaptation. Adaptations might be triggered by:

- *Faults on a quadrotor* - in this case the framework has to manage two main problems: (i) managing the quadrotor(s) with the fault, and (ii) reconfiguring the rest of the swarm so to accomplish the mission.
- *Unexpected context element* - quadrotors will be instrumented with sensors able to sense the environment with the aim at discovering obstacles which have not been considered at mission design-time. To avoid obstacles, a quadrotor uses some heuristics and exploits also the information coming from other quadrotors (e.g., if they have overcome or not encountered the obstacle).
- *Mission change* - the framework will support also changes on the mission made at run-time, for instance as required to accomplish a mission when a fault occurs on a quadrotor.

Roadmap of the paper. The remainder of the paper is structured as follows: the peculiarities of environmental monitoring missions and the related problems are discussed in Section 2. The proposed FLYAQ platform, which is able to deal with the described problems, is detailed in Section 3. Three main components of the platform, namely *mission design and code synthesis*, *reconfiguration engine*, and the *programming framework* are particularly detailed in Section 4, Section 5, and Section 6, respectively. Other representative attempts related to the work proposed in this paper are described in Section 7. Section 8 concludes the paper and outlines some perspective work.

[1] `http://www.flyaq.it`

[2] Open Street Map: `http://www.openstreetmap.org/`

2 Environmental monitoring missions

The application domain of the proposed platform is that of environmental monitoring missions. Examples of environmental monitoring missions include: continuous data acquisition during emergency situations (e.g., fires or earthquakes), longterm observations of how specific parameters of the environment change (e.g., air quality measurement), etc. Those kinds of mission are typically affected by a number of problems, in particular:

- **Costs**: this kind of missions typically requires high costs for personnel which have to be carried on the site to be monitored, and to the communication overhead required for synchronization purposes of the teams;
- **Safety**: Usually on-site personnel are exposed to significant risks (e.g., in case of fire, earthquake, and flood);
- **Timing**: Monitoring activities are very time consuming. The staff assigned to the task of monitoring is subject to grueling shifts. Moreover, the activities are stopped during the night slowing the execution of the mission (e.g., search for missing persons after natural disasters).

Figure 1 shows an example of mission that has been defined by using the FLYAQ platform. As can be seen in the figure, the language used to define the mission is graphical and allows the specification of the mission to be realized, together with several other aspects, such as obstacles (rectangles and circles), no fly zones, and emergency places (star symbol on the map) where quadrotors can safely land in case of problems. The mission represented in figure is an area to be monitored, where the swarm of quadrotors has to take photos according to a grid composed of squares of size 12x12 meters. The figure also shows the place that has been identified as home, i.e. where the swarm of quadrotors has to come back at the end of the mission.

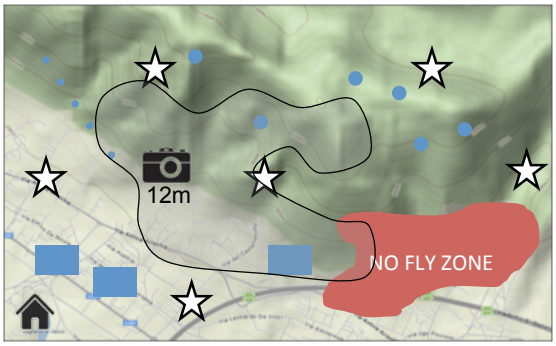

Fig. 1. Example of mission

Starting from this very high level description of the mission and from a description of the configuration of the swarm, FLYAQ autonomously computes the role of each single quadrotor to accomplish the mission. In this example we assume to have a swarm

composed of five quadrotors, each equipped with a camera. FLYAQ computes the entire organization of the mission including reconfigurations in response to possible faults. A detailed description of the reconfiguration is provided in Section 5. Moreover, FLYAQ automatically resolves the overlap between the monitoring area and no fly zones by reducing the monitoring area; this is because no fly zone areas are inviolable. Starting from the modeled mission, the FLYAQ platform generates the configurations for the five quadrotors. Once configured, these quadrotors perform the mission by flying from the home to the border of the monitoring area. Then, each quadrotor starts monitoring the assigned area: different monitoring areas are assigned to each quadrotor so to cover the entire area.

3 The FLYAQ platform

Figure 2 shows the operational setup of FLYAQ. The central box represents the FLYAQ platform, which supports all the activities ranging from mission modeling to simulation, to code generation, to mission monitoring and reconfiguration. The FLYAQ platform can be deployed on either a laptop or a singe-board device, e.g., Raspberry[3].

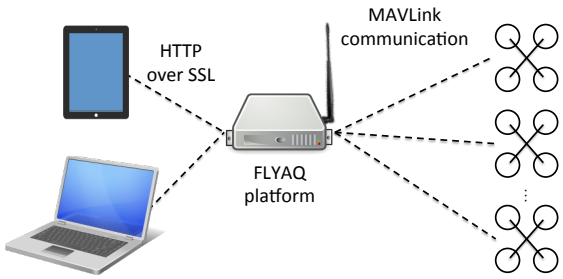

Fig. 2. FLYAQ operational setup

On-site operators design the mission, store, and monitor the status of ongoing missions via a standard web browser connected with the platform through a secure HTTP connection. This design decision enables operators to (re-)use any kind of device, such as tablets, laptops, etc., which are capable to run standard web browsers. Quadrotors are instructed and controlled by the FLYAQ platform via MAVLink communication[4]. MAVLink is a lightweight communication protocol for micro air vehicles. MAVLink enables quadrotors to be controlled from a long distance, even if the quadrotors are out of wifi range. In that case MAVLink can be used with radio modems to retain control up to eight miles.

By referring to Figure 3, the FLYAQ platform consists of eight main components. In the following we will give an overview of each of them.

[3] `http://www.raspberrypi.org/`
[4] `http://qgroundcontrol.org/mavlink/start`

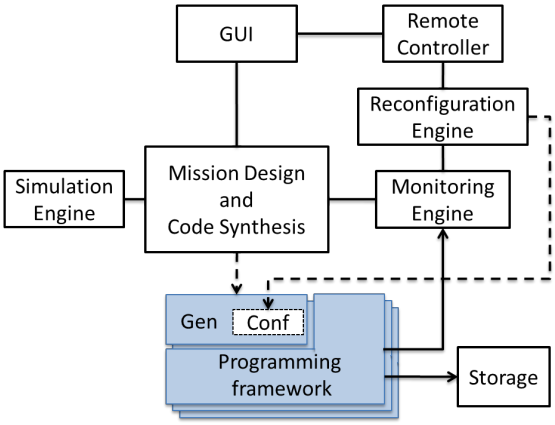

Fig. 3. FLYAQ Platform

Graphical User Interface: This component runs in the web browser used by on-site operators. It supports the interaction between operators and the FLYAQ platform. Since the system is conceived to support operators who are not required to be IT experts (e.g., firemen, woodmen, and rescue operators), the provided interface is simple to use and provides constructs which are specific to the monitoring mission domain. In particular, by means of this component the user can both specify monitor missions and control ongoing ones. The interaction with the other components of the platform is performed in a transparent way for the user, even though advanced users can have access to additional control panels, which permit to interact directly with the other parts of the platform.

Mission Design and Code Synthesis: This component plays a key role in the whole platform since it support all the modeling and code generation activities required for mission planning and for generating the final implementation code of the mission. A detailed description of this component is given in Section 4.

Simulation Engine: Before configuring the swarm of quadrotors involved in the modeled mission, this component permits to simulate it and to discover in advance possible problems. For instance, the engine is able to alert the user if the modeled mission cannot be executed since the number of available quadrotors is not enough to cover the interested area according to the user requirements (e.g., mission duration, types of used quadrotors).

Storage: This component is the central storage system used by the programming framework to store and maintain the current status of the mission being executed.

Monitoring Engine: During the execution of a modeled mission, the involved quadrotors are continuously in contact with the ground station, and more precisely with the monitoring engine component. All the relevant data regarding the quadrotor (e.g., altitude, position, battery charge level, and other data retrieved by the on-board sensors) are sent to the ground station and shown to the user. In critical cases (e.g., the battery drains too fast or the weather conditions become prohibitive) the monitoring engine can

autonomously ask the reconfiguration engine to reconfigure the involved quadrotors to fly back to the ground station (home), or ask the user to take their control.

Remote Controller: Even if we aim at managing quadrotor swarms, which once configured are able to perform missions in an autonomous way, in some cases it is necessary to provide users with the means to manually control some of the involved quadrotors (e.g., emergency situations occur and the quadrotors are not able to deal with them autonomously). In this respect, by means of facilities provided by the GUI, users can have access to the remote controller component, which permits to control remotely the considered quadrotor. This can be done once the current quadrotor configuration is changed by means of the reconfiguration engine component. In particular, such a component disables the mission which is running and enables the user to take the full control of the quadrotor.

Reconfiguration Engine: The reconfiguration engine is the component that manages the transition from one mode to another for each quadrotor and manages possible faults. This component is detailed in Section 5.

Programming framework: it consists of shared resources, such as a common functionality implemented in shared libraries, and run-time support. Generated code builds on the shared resources and functionalities provided by this component. The generated code contains the instructions to orchestrate each quadrotor in order to accomplish the mission. Those instructions are parametric with respect to a set of predefined parameters, which represent the variable part of the mission (*Conf* in the figure). Examples of these parameters include: the size of specific sub-areas to be monitored, the geographical position of a point of interest, the maximum altitude that can be reached by a quadrotor, etc. This component is detailed in Section 6.

4 Mission Design and Code Synthesis

As shown in Figure 4, this component supports all the modeling and code generation activities required for mission planning and for generating the final implementation code of the involved quadrotors. This component is the heart of the FLYAQ platform. Fundamentally, it defines and manages all the languages involved in the platform, together with the synthesis mechanisms needed to generate the final implementation code of the quadrotors. The involved languages span from the high-level design of the mission to the low-level implementation code realizing the final behavior of the quadrotors operating on the field. The main goal of the Mission Design and Code Synthesis component is to raise the level of abstraction for modeling the mission of a swarm of quadrotors, and to provide two synthesis steps towards the final ROS-compliant[5] implementation code. ROS (Robot Operating System) is an infrastructure for distributed inter-process/inter-machine communication and configuration. It provides hardware abstraction, device drivers, libraries and tools to create robot applications.

In order to raise the level of abstraction, we make use of Model-Driven Engineering (MDE) [6], which refers to the systematic use of *models* as first class entities throughout the software engineering life cycle. In MDE, domain-specific modeling languages

[5] `http://www.ros.org`

(DSMLs) are used to describe the system; they are defined using *metamodels*, which define the language main concepts, the relations among concepts within the domain, and their semantics. DSMLs are used to build a model of the system according to the semantics and constraints defined in their metamodel (in this case the model is said to "conform to" the DSML metamodel). Typically MDE approaches make use of a set of *transformation engines and generators* that produce various types of artifacts, such as target models, documentation, or code. Many are the model-to-model transformation approaches, as surveyed in [7].

Coming back to the mission design and code synthesis component, it relies on two modeling languages[6], as shown in Figure 4:

- *Monitoring Mission Language (MML)*: the language to design the mission from a high-level point of view. Concerns covered by this language include: coverage of specific areas, no-flight zones observance, intermediate goals achievement, coming back to the home and so on.
- *Quadrotor Behaviour Language (QBL)*: the language to describe the behaviour of each quadrotor in specific modes, and in coordination with the other quadrotors of the swarm. Concerns covered by this language include: reaching waypoints at given geographical positions, performing specific actions (e.g., taking a photo from a camera, landing, take off, etc.) during the mission, switching among specific behavioural modes, and so on.

The QBL models synthesis and ROS-compliant code synthesis steps lead from the mission-level language (i.e., MML) to the final ROS-compliant code realizing the mission. In the following sections we describe the involved languages and their corresponding synthesis step.

4.1 Monitoring Mission Language

Monitoring missions are graphically defined in the ground station by means of a dedicated MML modeling editor. The typical user of our Monitoring Mission Language is the operator who has to define the mission, (s)he is a non-technical stakeholder and has a very little knowledge of both the internal functioning of the FLYAQ platform, and of the technical specifications of the involved quadrotors. In order to achieve *versatility* and strong *adherence to the environmental monitoring missions* domain, we provide dedicated extension mechanisms for MML. Those mechanisms allow domain experts to specialize MML with additional constructs that are specifically tailored to the considered domain. For example, if operators are interested to monitoring solar panel installations in a rural environment, MML might be extended with the concept of solar panel groups, thermal image acquisition tasks, and solar panel damage discovery and notification tasks, etc. As shown in Figure 4, MML is composed of three layers: *Mission*, *Context*, and *Map*.

[6] A detailed description of all the elements constituting the modeling languages is out of the scope of this paper. Whereas, the focus of this work is on the engineering aspects of the FLYAQ platform.

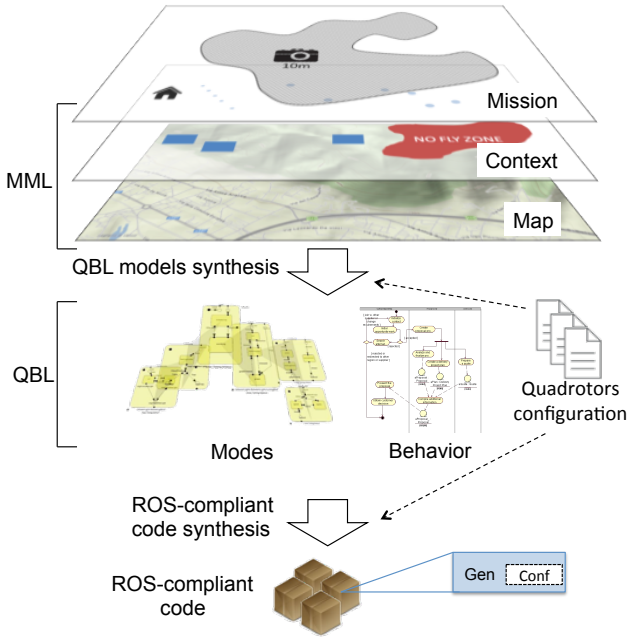

Fig. 4. Mission Design and Code Synthesis

- The **Mission** layer allows operators to design the mission from a high-level point of view with a reference to geographical points. Basically, an MML model is a combination of *tasks* and *events*. An *task* is a predefined sub-mission which can be executed during the mission, it can be performed until completion or it can be interrupted by an event. Examples of tasks provided by MML include: covering a specific area by acquiring images on a grid of x meters resolution, looking for some visual marker within a specific area, keeping a certain position on a given geographical point, coming back to a geographical point defined as the home of the mission, etc. Examples of *events* include: reaching a certain geographical point, acquiring a value from a sensor that is greater than a given threshold, finding an unforeseen obstacle, going out of energy, and so on.
- The **Context** layer models context information of the mission. By referring to [8], this modeling layer concerns spatial context and situational context. Indeed it represents those portions of geographical areas that have some relevant property, and those elements which can influence the execution of the mission, but that are not part of the mission itself. For example, in FLYAQ context models contain information about obstacles, no fly zones, emergency places where to land in case of emergency, wind and weather conditions, and so on.
- The **Map** layer represents the geographical area in which the swarm of quadrotors will operate. FLYAQ is integrated with Open Street Map[7], the well-known open source project providing a central geographic data source, with its map rendering

[7] `http://www.openstreetmap.org/`

and aesthetics. The *Map* layer is necessary to visualize the geographical points that have to be monitored during the missions being specified with respect to the geographical information of the area to be monitored.

These layers can be either (i) superimposed to provide a unique view of the mission or (ii) considered in isolation in order to provide a more focussed view of specific aspects of the mission. The user can interact with both the *Context* and *Mission* layers. In this way the user can define the mission but can also refine the context, for instance by adding new obstacles. The *Map* layer is a static visualization of geospatial data that can be used as reference for the mission. These layers permit to hide low-level details about the mission, which may be very hard to use by non-expert users.

By referring to Figure 4, an MML model is the input of the *QBL models synthesis* that produces models specified with the Quadrotor Behaviour Language (QBL). This synthesis step takes as input also the configuration of the swarm of quadrotors, i.e., the characteristics and number of the quadrotors available to the end user. This information is important to organize the mission and to assign a specific task to each quadrotor, according to its sensors, specific energy consumption parameters, and so on.

4.2 Quadrotor Behaviour Language

QBL enables the specification of two different diagrams, mode diagrams and behaviour diagrams representing the behaviour of each quadrotor in specific modes. These diagrams are inspired to UML activity diagrams and enable the specification of the internal behaviour of each single quadrotor, as well as synchronization among the different quadrotors. Examples of QBL actions include: land, take off, hover, head to, goto, read from a sensor, send feedback to the ground station, send/receive a notification to/from other quadrotors, etc.

Each quadrotor has associated different modes to specify different behavioural modalities, each devoted to describe the behaviour in a specific situation. Some of these modes, as well as the associated behaviour, can be defined once and for all, since they are independent from the specific mission. We are referring to modes like the “Comes back home” mode, which leads the quadrotor to a location that has been designed as home, and “Emergency” mode, which guides the quadrotor to an emergency place and performs the landing procedure. Other modes can be produced to realize missions in which a quadrotor is supposed to play different roles, such as reaching a specific area, and then starting with monitoring this area.

By referring to Figure 4, a QBL model is the input of the *ROS-compliant code synthesis* that produces the final ROS-compliant code that will command each quadrotor of the swarm. The synthesis step consists of model-to-code transformations implementing mission-independent mappings between QBL constructs and their corresponding code templates. This synthesis step takes as input also a set of models representing the flight dynamics of each quadrotor involved in the modeled mission. Those models describe the main aspects of a quadrotor from its flight dynamics point of view, such as body frame dimensions and shape, supported roll, pitch and yaw angles (and rates), its minimum and maximum speed, acceleration, flight altitude, battery duration, and so on. This information is needed to generate ROS-compliant code which is accurate with respect to the capabilities of the quadrotors it will run on.

4.3 ROS-compliant Code

As explained in Section 3, the generated ROS-compliant code is composed of two main parts: generated code and configuration parameters (configuration parameters permit to reconfigure the quadrotor, as done by the reconfiguration engine described in Section 5). The implementation code generated by the synthesis step described in Section 4.2 is based on the programming framework shown in Figure 3. For what concerns the hardware and low-level platforms, there are different solutions, both software and hardware, to support the design and development of autonomous quadrotors (e.g., Ascending Technologies[8], Arducopter[9], and A.R. Drone Parrot[10]). At this stage of the project, we are using Arducopter since it is an open-source platform based on Arduino, which provides a reasonable degree of flexibility both at software and hardware level. The programming language used by Arducopter provides the constructs for giving flight commands, and there are no monitoring specific constructs. In this respect, we aim at relying on the Arducopter platform and extend it with mechanisms to autonomously manage missions. In this respect, the manufactures of the quadrotors hardware might be interested in adopting the software produced in the context of the FLYAQ project.

5 Reconfiguration Engine: maintaining resilience

At design-time resilience is ensured thanks to static checks that are made once the mission has been designed. These checks permit to assess the feasibility of the mission according to the available swarm, weather conditions, quadrotors equipments and status, and so on. Moreover, the simulation engine plays an important role here, since it helps on-site operators to understand if the designed mission realizes their needs and if the mission can be accomplished successfully. In the remainder of this section we focus on the reconfiguration actions that are taken to maintain a desired degree of resilience when facing changes and unforeseen failures at run-time.

The reconfiguration engine is the component that manages the transition from one mode to another for each quadrotor in order to maintain a desired degree of resilience. Moreover, it can interact and change parameters of the variable part of the code that defines the behaviour of each quadrotor, i.e., the *Conf* part in the ROS code shown in Figure 4. The reconfigurations that FLYAQ is able to manage can be triggered according to different events detailed in the remainder of this section, i.e., *fault on a quadrotor*, *unexpected context element*, and *mission change*.

- *Fault on a quadrotor*. In case of a fault on a quadrotor, the platform has to resolve two main problems: (i) managing the quadrotor with the fault and (ii) reconfiguring the other quadrotors so to accomplish the mission.
  For what concerns (i), the quadrotor with the fault can be managed in two different ways. In case of manageable faults, the preferred solution of FLYAQ is to enforce the quadrotor to come back home. However, this solution requires to have a quadrotor able to come back home, i.e., it did not undergo a disruptive fault. Otherwise,

[8] `http://www.asctec.de/uav-applications/research/products/`
[9] `http://www.arducopter.co.uk/`
[10] `http://ardrone2.parrot.com/`

the second solution is the emergency plan: FLYAQ leads the quadrotor to the closest emergency place, if any available and feasible, or identifies the best place where to safely land. It is important to note that in FLYAQ we are considering only those faults that can be managed. In other words, we are dealing exclusively with software issues and the best we can do with hardware is to select products that are reliable.

For what concerns (ii), the reconfiguration really depends on the mission that have been defined: it can cause the redefinition of the areas borders within the mission (from the point of view of the single quadrotors), it can cause an additional monitoring point for a quadrotor, and so on. The reconfiguration takes also into account the status of each quadrotor of the swarm, such as its available battery life.

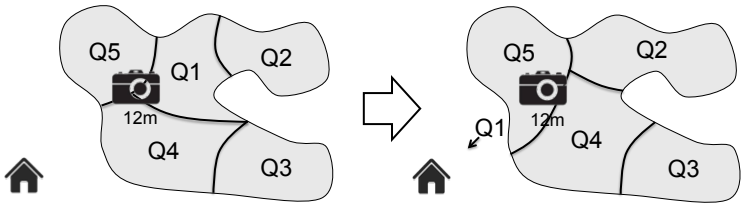

Fig. 5. Reconfiguration example

As an example, by referring to the situation presented in Section 2, let us assume that during the mission a failure happens on one of the engines of a quadrotor. Hopefully, the swarm of quadrotors was composed of drones that are resilient to damage to one engine, like the AscTec Firefly, a product of Ascending Technologies[11]. Specifically, these quadrotors are equipped with six motors and are able to fly even with five operating motors. In order to accomplish the mission, the FLYAQ platform has to reconfigure the swarm. As shown in Figure 5, FLYAQ contacts the quadrotor that has problems, let us call this quadrotor $Q1$, and triggers a mode change that leads $Q1$ to behave according to mode *"Comes back home"*. Moreover, the FLYAQ platform has to reconfigure the other drones of the swarm. In this case, the reconfiguration is realized by recomputing and reassigning adapted monitoring areas to the remaining four quadrotors. This is technically realized thanks to the fact that the generated code is parametric with respect to some parameters, as explained in Section 3. It is important to note that the reconfiguration does not involve the MML language, since no mission changes can be done.

– *Unexpected Context Element* - as seen in Section 4, the MML language has a layer that is devoted to specify the context in which the swarm of quadrotors will operate. One of the purposes of the context layer is the specification of obstacles. However, it can happen that the context is not completely accurate, e.g., some obstacles can be missing or not exactly represented. Moreover, a quadrotor can always encounter a bird that obviously cannot be represented as part of the context. To deal with this

[11] `http://www.asctec.de/uav-applications/research/products/asctec-firefly/`

kind of situations, quadrotors are instrumented with sensors able to sense the environment with the aim of discovering unexpected obstacles. Once an obstacle is sensed, a quadrotor switches to a predefined mode called *"Obstacles avoidance"* suitably defined to avoid obstacles. This mode is defined once forever and stops the quadrotor in the current position, tries to understand the dimensions of the obstacle, and if the dimensions are clear, the quadrotor autonomously defines a path that permits to go around the obstacle and then to maintain the mission. In the case of large obstacles, such as buildings, the quadrotor is not able to sense easily the dimensions of the obstacle and then contacts the FLYAQ platform to solve this problem. The platform checks if the other quadrotors have found a way to overcome the obstacle and in case the quadrotor that encountered problems can follow the path of one of the other quadrotors that have overcome or not encountered the obstacle. Otherwise, according to the battery life that is remaining on the quadrotor and according to the length of the remaining part of the mission, the FLYAQ platform can decide to call back home the quadrotor or even the entire swarm if it is the case.
- *Mission change* - FLYAQ supports also run-time changes to the mission. However, the changes that are admitted are those that require to just interact with the $Conf$ part of the code (see Section 3). This means that the code that defines the behaviour of a quadrotor cannot be re-generated at run-time; the reconfiguration engine only permits to change parameters, such as mission area borders, GPS coordinates of points to be monitored, etc.

6 Programming framework

This component is in charge of managing and orchestrating the swarm of quadrotors according to the ROS-compliant code generated from the QBL model. This component is in turn composed of two subcomponents:

- *Mission Instructor*: it is automatically generated by the ROS-compliant code synthesis step. This component uses the publish/subscribe communication paradigm of ROS to feed instructions to be delivered to the quadrotors forming the swarm.
- *Flight Instructor*: it is a component that is part of the programming framework. This component subscribes to the topics of interest, receives the messages published by the mission instructor, and delivers the corresponding commands to the proper drones. The delivery of those commands is realized by means of drone-specific controllers via MAVLink communication.

It is important to remark that the flight instructor component is mission- and drone-independent and it is developed once for all. Contrariwise the mission instructor is mission-dependent and is automatically generated every time a new mission is modeled and must be executed. Also, even if mission independent, each drone controller is tied to a specific type of drone, e.g., AR.Drone, Arducopter, AscTec Firefly, etc.

7 Related work

Currently, many research groups are working on technologies and solutions related to both quadrotors flight control and the management of autonomous quadrotors. A very

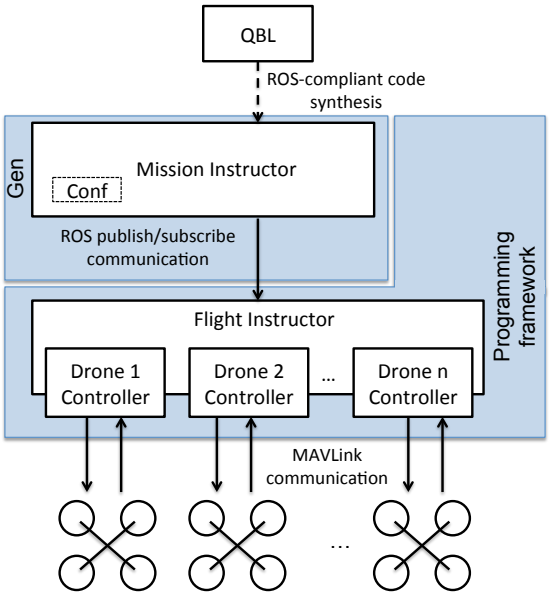

Fig. 6. Programming framework

detailed and complete survey on the advances in guidance, navigation, and control of unmanned rotorcrafts systems in general[12] is provided in [1]. Many *algorithms* have been proposed for automatic trajectory generation and control, with a strong focus on either trajectory optimization [9], feasibility [10], or safe obstacle and trajectories intersection avoidance [11]. Also, many research works have been proposed to provide navigation and control systems which are resilient to the presence of *unpredicted obstacles*. These approaches are based on real-time information which may come from cameras [12], other sensors mounted on the quadrotors (e.g., proximity sensors, LIDARs, GPS navigation sensors, UV sensors) [13] or a combination of cameras and sensors [14].

For what concerns the activity of *mission planning and definition*, many approaches focus on the definition of (either GPS-based or vision-based) waypoints and trajectories in the real world that must be navigated by the quadrotor in the field [15, 16], but to our knowledge *instruments which allow operators to define a complete mission of swarms of quadrotors are still missing*.

Differently from the approaches outlined above, our focus is on (i) the definition of the various tasks of a monitoring mission at a high-level of abstraction and (ii) on the automatic generation of the implementation code running on each quadrotor in the field. Therefore, the contribution of our approach is a platform which allows operators to straightforwardly define monitoring missions of swarms of quadrotors by masking all the complexity of the low-level and flight dynamics-related information of the quadrotors.

[12] According to [1], unmanned rotorcraft systems range from full-scale helicopters, to medium-scale helicopters, down to micro-flying robots.

Under this perspective, it is fundamental to remark that our approach is totally different from those approaches proposing autonomous waypoint navigation of UAVs. Indeed, the FLYAQ platform is able to automatically compute, plan, and assign all the waypoints that must be visited by each quadrotor of the swarm to accomplish the mission. Therefore, FLYAQ opens for future optimization and planning algorithms that do not demand to manually specify each single waypoint of the mission (that actually may be hundreds in complex missions).

8 Conclusions and Future work

In this paper we presented the FLYAQ platform for supporting environmental monitoring missions of swarms of autonomous quadrotors. The platform exposes a domain-specific modeling language (i.e., MML) that enables operators to plan missions by focussing on mission-specific tasks only. FLYAQ masks the complexity of the technical details of both the ground station and all the involved quadrotors; indeed, the quadrotors will operate autonomously in the field because the implementation code realizing their behaviour is automatically generated by FLYAQ from the mission model created by the operator. Moreover, FLYAQ also provides facilities for simulating, monitoring and (optionally) controlling the behaviour of the quadrotors.

Currently, the platform architecture and its modeling languages have been defined, and we are developing a first prototype of the graphical editor for the MML language. As future work, we are completing the specification of the Monitoring Mission Language and we are investigating on its expressivity and flexibility in covering the various types of tasks that may be demanded of a language like that. Also, we are working on the other auxiliary languages in order to provide a complete tool for simulating and verifying the generated behaviour of the quadrotors with a focus on properties such as safety, liveness and on the feasibility of the various flight trajectories that shall be travelled during the mission. Moreover, we are investigating on how developing a family of adaptation/fault tolerance architectural patterns to deal with the specific situations; also, we will extend the reconfiguration engine in order to being able to deal with concurrent and multiple faults or mission changes. Finally, we are planning to collaborate with other research groups for supporting fundamental capabilities such as resilience to unexpected obstacles. The platform will be distributed as an open-source product.

Acknowledgments

We would like to thank Roberto Antonini and Marco Gaspardone for their suggestions and useful discussions about the architecture of the FLYAQ platform. This work is partly supported by the Startup Grant of Working Capital 2012 (`http://www.workingcapital.telecomitalia.it/`), Art. 10 Nationally funded by MIUR, and Ricostruire project (RIDITT - Rete Italiana per la Diffusione dell'Innovazione e il Trasferimento Tecnologico alle imprese).

References

1. Kendoul, F.: Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. J. Field Robot. **29**(2) (March 2012) 315–378
2. Nathan, P., Almurib, H., Kumar, T.: A review of autonomous multi-agent quad-rotor control techniques and applications. In: Mechatronics (ICOM), 2011 4th International Conference On. (2011) 1–7
3. Martinelli, A.: Vision and imu data fusion: Closed-form solutions for attitude, speed, absolute scale, and bias determination. Robotics, IEEE Transactions on **28**(1) (2012) 44–60
4. Likhachev, M., Ferguson, D.: Planning long dynamically feasible maneuvers for autonomous vehicles. Int. J. Rob. Res. **28**(8) (August 2009) 933–945
5. Achtelik, M., Weiss, S., Chli, M., Siegwart, R.: Path planning for motion dependent state estimation on micro aerial vehicles. In: Proc. of the IEEE International Conference on Robotics and Automation (ICRA). (May 2013)
6. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer **39**(2) (2006) 25–31
7. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. **45** (July 2006) 621–645
8. Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D.: A survey of context modelling and reasoning techniques. Pervasive and Mobile Computing **6**(2) (2010) 161 – 180
9. Hehn, M., DAndrea, R.: Quadrocopter trajectory generation and control. In: IFAC world congress. (2011) 1485–1491
10. Augugliaro, F., Schoellig, A.P., D'Andrea, R.: Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In: IEEE/RSJ International Conf. on Intelligent Robots and Systems (IROS). (2012) 1917 –1922
11. Leonard, J., Savvaris, A., Tsourdos, A.: Towards a fully autonomous swarm of unmanned aerial vehicles. In: Control (CONTROL), 2012 UKACC International Conf. on. (sept. 2012) 286 –291
12. Andert, F., Adolf, F., Goormann, L., Dittrich, J.: Mapping and path planning in complex environments: An obstacle avoidance approach for an unmanned helicopter. In: Intl. Conf. on Robotics and Automation (ICRA). (may 2011) 745 –750
13. Merz, T., Kendoul, F.: Beyond visual range obstacle avoidance and infrastructure inspection by an autonomous helicopter. In: Intl. Conf. on Intelligent Robots and Systems (IROS). (sept. 2011) 4953 –4960
14. Zhang, T., Li, W., Achtelik, M., Kühnlenz, K., Buss, M.: Multi-sensory motion estimation and control of a mini-quadrotor in an air-ground multi-robot system. In: Intl. Conf. on Robotics and biomimetics. ROBIO'09, IEEE Press (2009) 45–50
15. Bouabdallah, S., Siegwart, R.: Full control of a quadrotor. In: Intl. Conf. on Intelligent Robots and Systems. (2007) 153 –158
16. Kendoul, F., Zhenyu, Y., Nonami, K.: Embedded autopilot for accurate waypoint navigation and trajectory tracking: Application to miniature rotorcraft uavs. In: Intl. Conf. on Robotics and Automation. (may 2009) 2884 –2890