FLYAQ: Enabling Non-Expert Users to Specify and Generate Missions of Autonomous Multicopters

Darko Bozhinoski¹, Davide Di Ruscio², Ivano Malavolta¹, Patrizio Pelliccione²,³, Massimo Tivoli²
¹Gran Sasso Science Institute, L’Aquila (Italy)
²University of L’Aquila, Department of Information Engineering, Computer Science and Mathematics (Italy)
³Chalmers University of Technology | University of Gothenburg, Dep. of Comp. Science and Eng. (Sweden)
darko.bozhinoski@gssi.infn.it, davide.diruscio@univaq.it, ivano.malavolta@gssi.infn.it,
patrizio.pelliccione@gu.se, massimo.tivoli@univaq.it

Abstract—Multicopters are increasingly popular since they promise to simplify a myriad of everyday tasks. Currently, vendors provide low-level APIs and basic primitives to program multicopters, making mission development a task-specific and error-prone activity. As a consequence, current approaches are affordable only for users that have a strong technical expertise. Then, software engineering techniques are needed to support the definition, development, and realization of missions at the right level of abstraction and involving teams of autonomous multicopters that guarantee the safety today’s users expect. In this paper we describe a tool that enables end-users with no technical expertise, e.g., firefighters and rescue workers, to specify missions for a team of multicopters. The detailed flight plan that each multicopter must perform to accomplish the specified mission is automatically generated by preventing collisions between multicopters and obstacles, and ensuring the preservation of no-fly zones.

I. INTRODUCTION

The near future will be pervaded by multicopters performing a variety of tasks, like damage assessment after earthquakes, searching for survivors after airplane accidents and disasters, coastal surveillance, securing large public events, monitoring oil and gas pipelines, observing traffic flows, monitoring pollution emission, and protection of water resources [14]. However, at the state of the art and practice on-site operators must deeply know all the types of used multicopters in terms of e.g., flight dynamics and hardware capabilities in order to correctly operate with them. On-site operators have to simultaneously control a large number of multicopters during the mission execution. Moreover, professional use of multicopters often is realized by allocating two operators for each multicopter, where the first operator controls the movements, while the second one controls the instrumentation, like photo camera, its gimball, and other sensors.

Nowadays, vendors provide low-level APIs and basic primitives to program multicopters, thus making mission development an (error-prone) activity that can be performed by a restricted number of highly skilled professional stakeholders. Tasks are very specific and this limits the possibilities for their reuse across missions and organizations. Consequently, current approaches are affordable only for users that have a strong expertise in the dynamics and technical characteristics of the used multicopters. Then, software engineering approaches and methodologies are needed to support the definition, the development, and the realization of missions involving teams of autonomous multicopters while guaranteeing safety.

This paper describes a tool for the definition of missions of teams of multicopters and the generation of the detailed flight plan of each multicopter in the team. Specifically, starting from a high-level description of the mission, FLYAQ automatically generates a detailed flight plan for a team of autonomous multicopters that will satisfy the specified mission while preventing collisions with other multicopters and obstacles, and respecting no-fly zones. End-users of FLYAQ have expertise neither in ICT nor in multicopters dynamics, e.g., firefighters and rescue workers. The webpage of the tool is available at http://www.flyaq.it.

II. OVERVIEW OF THE TOOL

FLYAQ ensures a strong adherence with the application domain by providing an extensible domain specific language, named Monitoring Mission Language (MML), which permits to graphically define the missions. Extension mechanisms of the language allow domain experts to specialize MML with additional tasks that are specifically tailored to the considered domain. For example, if operators are interested on monitoring solar panel installations in a rural environment, the language might be extended with tasks representing the concept of, e.g., solar panel groups, thermal image acquisition, solar panel damage discovery and notification.

As shown in Figure 1, MML is composed of three layers: i) to specify the mission by means of the modeling constructs
provided by the language, ii) to specify the context in which the swarm of multicopters has to operate, such as no-fly zones, obstacles, etc., and iii) a map representing the geographical zone where the mission will be executed.

Once the mission has been specified, waypoints and trajectories are automatically calculated. They are represented in an intermediate language named QBL. Examples of QBL actions include: land, take off, hover, head to, goto, read from a sensor, send feedback to the ground station, and send/receive a notification to/from other multicopters. QBL has been defined through an iterative process involving experts of the application domain considered by FLYAQ. QBL models are interpreted at mission run-time by a set of software controllers, each of them commanding a single multicopter according to the various movements and actions contained into the QBL model. Each controller is dedicated to a specific type of multicopter so that it is able to account for the specific flight dynamics and other characteristics of the managed multicopter.

As shown in Figure 2, the QBL model automatically generated out of the MML one is organized into $n$ parts, each of them describing the detailed flight plan of a specific multicopter. The flight plan of a multicopter can be abstracted as a finite state transition system where each transition corresponds to a QBL operation. Both cyclic and alternative behaviours can be performed. Multicopters can exchange synchronization and communication messages (see dashed arrows in the Figure 2). FLYAQ assumes that at the beginning of the mission each multicopter will be positioned at its home location and that at the end of the mission it will land to a (possibly different) location. Then, the finite state system is structured in three parts:

- **Mission entering** consisting of the operations required to start the mission, e.g., take-off;
- **Mission tasks execution** consisting of the operations required to accomplish each mission-specific task, e.g., searching for an object in an area, taking a picture in the waypoints of an area, detecting the level of carbon dioxide in a specific point;
- **Mission exiting** consisting of the operations required to conclude the mission, e.g., going back to home, landing.

The main difficulty of the approach is that MML is extensible and hence it is not possible to define once all the translation rules needed to translate MML tasks into QBL operations. Therefore, the automated generation is based on three main concepts: (i) typology and characteristics of the zone that is affected by the mission, (ii) strategies to be applied to calculate the concrete movements that drones have to perform, and finally (iii) actions to be performed while visiting the identified waypoints. These concepts pose constraints and permit to use consolidated and optimised algorithms (e.g., shortest-path calculation between two points, path planning, etc.) that will be exploited by the generation process to, e.g., determine how a drone should visit a fly-zone according to specific path planning policies. Then, a FLYAQ tool extender can define concrete tasks for the considered domain by associating these tasks to the general concepts defined in the tool. In this way the generation of drone specification is fully automated.

The translation from MML to QBL relies on three auxiliary functions that implement suitable operations to (i) distribute the geographical area of each task into a set of (sub-)areas, each of them assigned to a specific drone (function Divide); (ii) let a drone approach the mission by reaching the starting point in the zone assigned to it (function Appr), and (iii) cover the assigned zone according to the specified strategy and by performing the specified actions for each way point (function Cover). The generation of auxiliary functions builds on state-of-the-art and well-established algorithms for solving problems like polygon partitioning, path finding, and graph traversals [13]. The modularity of the current implementation of FLYAQ allows a straightforward inclusion of alternative algorithms and/or future advances of existing ones without affecting the generation process.

The generation from MML to QBL has been implemented by following the model-driven engineering paradigm. More precisely, the outputs of the three auxiliary functions applied to the source MML model are represented as three corresponding models. Such models are taken as input by the model transformation MM2QBL, which is able to generate QBL models out of MML ones (see Listing 1). Such transformation is developed by means of the Atlas Transformation Language (ATL) [8], which is a hybrid language containing declarative and imperative constructs. The fragment of the MM2QBL transformation consists of a header section (line 2), transformation rules (lines 14-36), and a number of helpers, which are used to navigate models and to define complex calculations on them (lines 4-12).

According to the header section the MM2QBL transformation takes as input four input models to generate a QBL model out of them. Helpers and rules are the constructs used to specify the transformation behaviour. Each rule defines the elements to be generated by means of target patterns (e.g., lines 32-35) that specify the instances of the target metamodel to be generated (i.e., the DBS metaclass of the QBL metamodel) and a set of bindings. A binding refers to a feature of the type, i.e., an attribute or a reference, and specifies an expression whose value initializes the feature.

To implement the generation of auxiliary functions, corresponding helpers have been defined. For instance, the divide helper in lines 4-9 is able to read the source Divide model and retrieves as output the sub-zones representing a spatial partition of the task space.

The MM2QBL transformation has been designed so to have three main rules to manage the generation of target model frag-
ments related to mission entering, tasks execution, and exiting (see lines 20–22). Additional rules are specified for generating specific elements of the target QBL models. For instance, the generation of target TakeOff elements is performed by the MissionEntering_TAKEOFF rule, which is called by the MissionEntering rule (see line 34).

Listing 1. Fragment of the MM2QBL transformation

```plaintext
module mm2qbl
1 create OUT : QBL from IN : MML, IN_APPR : APPR, IN_COVER : COVER, IN_DIVIDE : DIVIDE;
2... helper def : divide(task : MML!Task, positions : Sequence(MML!Coordinate), _context : MML!Context) : DIVIDE!
3... Output =
4 DIVIDE!Mapping.allInstances()
5 select(m | m.input.task.name = task.name and
6 thisModule.sameCoordinates(m.input.positions,
7 positions) and
8 thisModule.sameContext(m.input._context,_context)
9 ).first().output;
10... helper def : appr(...) : Sequence(APPR!Coordinate) = ...;
11 helper def : cover(...) : Sequence(COVER!Output) = ...;
12... rule DroneTasks
13 from
14 s: MML!DroneTasks
to
15 d : QBL!Drone (name <- s.drone.name)
16 do |
17 thisModule.MissionEntering(d);
18 thisModule.MissionTasks(d);
19 thisModule.MissionExiting(d);
20 }
... rule MissionEntering(d : QBL!Drone) 
21 using 
22 approachingPoints : Sequence(APPR!Coordinate) = ...
23 lastApproachingPoint : APPR!Coordinate = approachingPoints ->last();
24 t:QBL!DBS 
25 drone <- d,
26 transitionFunctions <- thisModule.MissionEntering_TAKEOFF
27 (t,si,lastApproachingPoint.altitude) ...
28 }
...```

Due to space limitations, we do not provide the reader with the full implementation of the MML2QBL transformation; the interested reader can download it from http://www.flyaq.it.

III. INTENDED USERS OF THE TOOL AND SCENARIOS

The intended users of FLYAQ are public or private entities that have to perform missions via multicopters. We also expect that multicopters manufacturers will be interested in participating in the industrial exploitation of the idea with the aim of including in the produced multicopters the software developed by FLYAQ. In fact, the software layer produced by the project will allow to specify monitoring missions at a high level of abstraction, that is, making possible the definition for users non-expert in ICT, and that are experts in a specific mission. Additional users of FLYAQ are model-driven engineering experts that have to perform the extension of the FLYAQ tool to support the specification and execution of new tasks and types of specific missions according to the knowledge extracted from domain experts.

Within the panorama of missions [14], typical scenarios concern: (i) Disaster Prevention and Management, like damage assessment after earthquakes, searching for survivors after airplane accidents and disasters; (ii) Homeland Security, such as coastal surveillance, securing large public events; (iii) Protection of Critical Infrastructure, such as monitoring oil and gas pipelines, protecting maritime transportation from piracy, observing traffic flows; (iv) Communications, like broadband communication, telecommunication relays; (v) Environmental Protection, such as pollution emission, protection of water resources.

Figure 3 shows a screenshot of the FLYAQ editor while the on-site operator is specifying by means of MML a mission to monitor a large public event in a small city for security reasons (tasks of the mission have been graphically manipulated so to improve the readability of the figure). The editor consists of four main components: an interactive map for drawing and editing missions, a dedicated palette with all the available types of tasks according to MML and its extensions (A in Figure 3), a palette for managing the available drones to be used in the mission (B in Figure 3), and a panel for managing the execution order of the specified tasks (C in Figure 3). For more details about MML interested readers can refer to [4].

A FLYAQ mission essentially results in a set of geographical areas, movement strategies that drones involved in the mission should perform on selected areas, such as coverage, search for an object, etc. and actions to be performed while traversing the interested waypoints, e.g., taking a picture or performing a video. The specified mission is composed of two tasks to be performed in parallel:

- **Photo Grid Task (PGT)** - this task is performed above a square (see the rectangle in Figure 3 within the circle PGT) to monitor it. The photo grid task identifies a virtual grid within the area, each cell of the grid having a size of 10 meters. The drones executing the task will fly over each cell of the grid at an altitude of 25 meters, and then will take a picture of the area directly below them.

- **Road Task (RT)** - this task refers to a polyline corresponding to the streets to be monitored (see the polyline in Figure 3 identified by the circle RT). Drones are required to fly along the polyline at an altitude of 25 meters, and take a picture every 200 meters along the polyline.

The mission will be realized by three drones that will be positioned in a large parking lot close to the city center (see the
FLY AQ allows the user to define also contextual information about the mission. In this example, the context specification comprises missions involving different tasks, various numbers of tasks, and various context descriptions.

**Mission tasks execution**: This phase concerns the execution of each task in which one task, i.e., task \( t_1 \), involves both \( d_1 \) and \( d_2 \), as shown in Figure 4: \( d_1 \) takes off from home and reaches the altitude of the mission’s starting point. The obstacle is within the area involved in the PGT task and it avoids collisions between \( d_3 \) and \( d_4 \) during the execution of \( t_1 \). According to the specified strategy and grid dimension, \( d_4 \) covers its discretized sub-area by performing a specific visit plan (see the sequence of arrows shown in the right-hand side of the figure).

First, \( d_1 \) reaches the starting point of the mission by means of a sequence of GoTo operations. The mission entering path is calculated in order to avoid both collisions with other drones in the mission and traversing no-fly zones. In fact, as shown in the right-hand side of Figure 5, \( d_1 \) approaches the mission by traversing the \( c_1 \) and \( c_2 \) points, hence reaching \( p_1 \), which is a vertex of the area specified for the execution of task PGT. Traversing these points means that \( d_1 \) avoids specified no-fly zones and the path specified for the execution of task RT by drone \( d_3 \), hence avoiding also collisions with \( d_3 \). Accordingly, for \( d_2 \) and \( d_3 \) different mission entering paths are calculated.

Second, \( d_1 \) covers its sub-area by means of a sequence of GoTo operations and DoPhoto actions; the sequence of waypoints is: \( q_1, \ldots, q_{16} \) (see the mission task execution part of Figure 4). Notice that \( q_1 \) coincides with \( p_1 \), then the first GoTo, i.e., GoTo\((q_1)\), will have no effect since the \( d_1 \) will be already in \( p_1 \). This is a side effect of having the code generated, however this has no effect on the mission execution and this GoTo operation can be removed by minimizing the final drone behaviour specification.

**Mission exiting**: This phase concerns exiting the mission hence leading \( d_1 \) to come back to home and land (see transitions from \( s_2 \) to \( s'_1 \) in Figure 4).

### IV. PRELIMINARY ASSESSMENT OF THE TOOL

As a first step towards a realistic assessment of the feasibility of our automatic generation method, we executed generated QBL models by using a Software-In-The-Loop (SITL) simulation platform. The main characteristic of SITL simulations is that the used software stack is exactly the same as the one used in real flights; the only difference with respect to real flights is that the key low level hardware drivers (e.g., GPS sensors, accelerometers, etc.) are simulated via a dedicated software. The main component of our SITL simulation stack is MAVProxy\(^1\), that is a developer-oriented, minimalist and extendable ground control station for any unmanned autonomous vehicle. Moreover, we performed real tests involving the Parrot AR.Drone2.0 multicopter\(^2\); a video of a demo might be found at: http://cs.gssi.infn.it/files/flyaq.mp4. We considered missions involving different tasks, various numbers of tasks, and various context descriptions.

### V. RELATED WORK

A comprehensive survey of approaches for cooperative teams of UAV operating as distributed processing systems can be found in [3]. The work in [12] introduces CSL, which is a high-level feedback control language for mobile sensor networks. The style of the language is similar to that of Petri nets (missions are made of tasks with tokens and transitions). The run-time architecture of the proposed approach allows engineers to update a modelled mission at run-time by means

---

\(^{1}\)http://ridge.github.io/MAVProxy

\(^{2}\)http://ardrone2.parrot.com/
of a patching system for the mission specification. Differently from our approach, the CSL language does not support any kind of check on the feasibility and safety of the modelled mission; also, trajectory plan in 3D is not supported. Many algorithms have been proposed for automatic trajectory generation and control, with a strong focus on either trajectory optimization [7], feasibility [1], or safe obstacle and trajectories intersection avoidance [11]. The interested reader can refer to [6], which proposes an overview of existing motion planning algorithms specific for UAV guidance.

From a slightly different perspective, the work in [10] proposes a new paradigm called cyber-physical computing cloud (CPCC). It allows any customer to assign, check, and distribute sensing services on virtual vehicles. Essentially, this approach ports the principle of Platform-as-a-Service (PaaS) to the distributed robotics domain. According to this principle, the system can perform multi-customer information acquisition missions on swarms of UAV operated and maintained by a third party, similarly to how traditional web-based PaaS systems work. Differently from our approach, in [10] free-space environment is assumed and collisions are not taken into consideration. Moreover, location movements related to the tasks are manually given by the customers of the PaaS system, then tasks are assigned to physical vehicles by using a binding algorithm based on Voronoi cells. 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 multicopter in the field [2], [9].

Differently from these approaches, our main objective is to provide an extensible software tool that makes the specification and generation of missions possible for people that are neither expert on ICT nor in robotics. In other words we address the problem from the software engineering perspective, as our tool (i) focuses on the definition of the various tasks of a monitoring mission at an higher level of abstraction, i.e., tasks and tasks dependencies; (ii) allows engineers to automatically generate detailed flight plans from a user friendly, domain-specific, and graphical description of a mission; (iii) generates flight plans that avoid obstacles, collisions and no-fly zones; (iv) does not demand to manually specify each single waypoint of the mission (that actually may be hundreds in complex missions), rather it is able to automatically compute, plan, and assign all the waypoints that must be visited by each multicopter of the swarm to accomplish the mission; and (v) is independent from the used task allocation, geometric and path finding algorithms, thus enabling for the use of state-of-the-art and well-established algorithms depending either on the next advances of those algorithms and on the traits of the missions to be performed in the future.

VI. POTENTIAL IMPACT

The major benefits related to the adoption of the FLYAQ tool by organizations that need to carry out dangerous and difficult missions are:

Low cost - the use of FLYAQ and its multicopters permits to reduce the costs due to the employment of personnel on the site to be monitored, and to the adoption of complex communication means for synchronizing the teams.

Enhanced safety - the employment of autonomous multicopters in dangerous missions permit to avoid to expose the staff on-site to significant risks in case of particular situations due to fire, earthquake, flood, etc.

Improved timing - usually monitoring activities are very time consuming, the staff assigned to monitoring is subjected to gruelling shifts, and often the missions are stopped during the night. The autonomous multicopters instructed by means of the FLYAQ environment help to reduce such difficulties.

Graphical language - missions are graphically defined in the ground station. The graphical language is integrated with the Open Street Map well-known open source product in order to visualize the geographical points that have to be monitored during the missions being specified. Having a graphical language for defining monitoring missions, permit to hide the detailed flight plan, which may be very hard to use and understand by non-expert users.

Versatility - the graphical language used for defining monitoring missions is very simple, and provides the concepts which are familiar to the domain experts. Thus there is no need for expensive training sessions or special maintenance activities.

Tool support - research and development groups who are willing to work on self multicopters and instruct them at a high level of abstraction, will have access to the source code of the FLYAQ tool, extend, reuse individual components, and then carry out their research without having to develop an entirely new framework from scratch.

VII. CONCLUSIONS AND FUTURE WORK

This paper describes the FLYAQ tool to generate from a user friendly, domain-specific, and graphical description of a mission, the detailed flight plan that each multicopter in a team has to perform. The generation approach avoids (i) collisions among multicopters and between multicopters and obstacles, (ii) violations of no-fly zones, and (iii) unexpected behaviours that may come from the collaboration of independent multicopters. FLYAQ has been conceived to allow non-experienced users to easily specify missions for a team of multicopters. As future work we are planning to enhance the current implementation of FLYAQ by integrating methodologies to control the mission execution at run-time. The idea is to exploit the synthesized flight plan at run-time so to force the drones to exhibit only desired behaviours [5]. Another interesting future work concerns the ability of accounting for time and resources consumption that are extremely important in this domain. This will enable, e.g., the possibility of statically checking mission end-to-end timelines, or the realization of resource-aware missions. We will investigate also the possibility of making communication between drones more flexible, so to enable also emerging behaviours; this will open challenging and futuristic scenarios where intelligent swarms of drones will decide at run-time operations to be performed in order to satisfy the goal of a (possibly evolving) mission. Finally, the current benefits and limitations of the FLYAQ languages and their supporting tools will be assessed by means of studies involving the various stakeholders of FLYAQ. This will permit also to identify potential directions for improvements.
REFERENCES


