System simulation of a fleet of drones to probe cumulus clouds

Rafael Bailon-Ruiz, Christophe Reymann, Simon Lacroix, Gautier Hattenberger, Hector Garcia de Marina, Fayçal Lamraoui

System simulation of a fleet of drones to probe cumulus clouds

Rafael Bailon-Ruiz[1], Christophe Reymann[1,2], Simon Lacroix[1],
Gautier Hattenberger[3], Hector Garcia de Marina[3], Fayçal Lamraoui[4]

***Abstract*— Simulation plays an essential role in the development of complex systems. This paper reports on the development of a simulation infrastructure for a fleet of UAVs conceived to probe clouds, using an adaptive sampling scheme that calls for cloud mapping and trajectory planning. The mission is presented, the global approach to solve it and the ensemble of required processes are sketched. An overall simulation architecture is then depicted, and the details of its development using the Robot Operating System (ROS) are presented.**

I. Introduction

Atmospheric scientists have been early users of UAVs, that bring forth several advantages over manned flight to probe atmospheric phenomena: low cost, ease of deployment, possibility to evolve in high turbulences, etc. UAVs are now used for missions like volcanic emissions analysis, polar research, or climatic and meteorological sciences, and the yearly conferences organized by the International Society for Atmospheric Research using Remotely piloted Aircraft – ISARRA[1] gather a growing audience. An in-depth overview of the various fixed-wing airframes, sensor suites and state estimation approaches that have been used so far in atmospheric science is provided in [1].

The objective of the SkyScanner project[2], which brings together atmosphere and drone scientists, is to conceive and develop a fleet of micro UAVs to better assess the formation and evolution of low-altitude continental cumulus clouds. There remain uncertainties in the modelling of these natural phenomena, which could be alleviated by the mapping of a variety of data within and in the close vicinity of the cloud. Wind currents, pressure, temperature, humidity, liquid water content, radiance and aerosols are variables of interest that must be *collected over a spatial and temporal extent*: a fleet of UAVs let foresee the ability to fully characterize the evolution of such atmospheric phenomena.

Contribution: The development of a fleet of drones able to acquire data in such conditions calls for a series of technological and scientific advances, that span from the conception of the UAVs and their on-board sensors to the definition of a cooperation strategy so as to optimize data throughput and endurance, via flight control algorithms to track 3D trajectories in the presence of significant winds. Designing, testing and deploying such a complex system certainly benefits from the use of simulation tools, which allow preliminary testing and validation before field trials. The work presented here reports on the development of a *system level simulation infrastructure*, which is meant to support, validate and benchmark the integration of the various software components that drive the fleet.

Although the exploration of clouds context is a specific one, the principles on which the simulation relies on are generic, and can apply to any other kind of mission for fleets in which UAVs embark environment perception and decisional capacities.

Outline: The following section describes the problem at hand, briefly reviews the state of the art and sketches our approach. Section III states the requirements of a simulation for a fleet of UAVs, and presents its architecture. Section IV is the core of the paper: it depicts the implementation of various components that constitute the system simulation. It ends with an illustration, and a conclusion recaps the paper.

II. Probing clouds with UAVs

A. Problem statement

To fulfill the needs of atmosphere scientists, the fleet should collect data with a spatial and temporal resolution of respectively about 10 m and 1 Hz over the cloud lifespan. For this purpose, the overall control of the fleet must address the two following challenges:

- It is a poorly informed problem. On the one hand the UAVs perceive the variables of interest only at the positions they reach (all the atmosphere sensors perform pointwise measures at their position), and on the other hand these parameters evolve dynamically. The mapping problem in such conditions consists in estimating a 4D structure with a series of data acquired along one-dimensional manifolds.
- It is a highly constrained problem. The mission duration must be of the order of a cumulus lifespan, that is about one hour, and the winds considerably affect both the possible trajectories of the UAVs and their energy consumption – all the more since we are considering small sized motor glider aircrafts with a maximum take off weight of 2.0 kg. Winds are the most important variables that influence the definition of the trajectories and are mapped as the fleet evolves: mapping the cloud is a specific instance of a classic "explore vs. exploit" robotic problem.

Exploring clouds with a fleet of UAVs is therefore a particularly complex task. The challenge to overcome is

[1]LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France
[2]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
[3]ENAC, 7 avenue Edouard-Belin, F-31055 Toulouse, France
[4]Météo-France / CNRS, CNRM/GAME, Toulouse, France
[1]https://isarra.org/
[2]https://www.laas.fr/projects/skyscanner/

to develop *non-myopic adaptive strategies* using myopic sensors, that define UAV motions that maximize both the amount of gathered information and the mission duration.

B. Related work

In the context of atmosphere probing, so far UAVs follow pre-planned trajectories to sample the atmosphere. In the robotics literature, recent works have tackled the problem of *autonomously* exploring or exploiting atmospheric phenomena. The possibility of using dynamic soaring to extend the mission duration for sampling in supercell thunderstorms has been presented in [2]. In this case, only the energetic consumption is optimized, and the gathered information does not drive the planning. [3] presents an approach where a glider explores a wind field trying to exploit air flows to augment flight duration. The problem of tracking and mapping atmospheric phenomena with a UAV is also studied in [4]. The authors use Gaussian Process Regression (GPR) to map the updraft created by a smoke plume. Even though the mapped currents are not taken into account for the navigation, it is worth to remark that contrary to the previous contributions, here experiments with a real platform are presented. This shows the possibility of online mapping of atmospheric phenomena by a fixed-wing UAV using GPR. An other significant contribution on wind-field mapping is presented in [5]: aiming at autonomous dynamic soaring with a small UAV, the authors present and approach in which the wind field is modelled by polynomials, which parameters are estimated with a Kalman Filter. Experiments in which the mapped wind-field is compared to an "air-truth" obtained by tracking lighter than air balloons small are presented. Finally, autonomous exploration of current fields is not exclusively related to aerial applications: the use of Autonomous Underwater Vehicles for oceanographic studies has been recently investigated (*e.g.* [6]). Apart from this latter contribution, in all the aforementioned work only the use of a single vehicle to achieve the mission is considered, and no multi-UAV systems are proposed.

C. Overall approach and architecture

The mapped phenomenon being a dynamic one, we opted for an adaptive data collection scheme, which can be much more efficient than predefined acquisition patterns. A global approach has been defined, which casts the overall problem in a hierarchy of two modeling and decision stages. A macroscopic parametrized model of the cloud (Fig. 1) is built and exploited at the *higher level* by an atmospheric scientist, which sets information gathering goals, such as "map this volume". A UAV or a subset of the UAV fleet is allocated to each goal, considering the UAVs current position in the cloud, their on-board energy level, and their sensing capacities. These high level goals typically consist of cloud regions to explore, and are autonomously handled by the lower *fleet control level*, which optimizes the selected UAVs trajectories using an on-line updated dense model of the variables of interest.

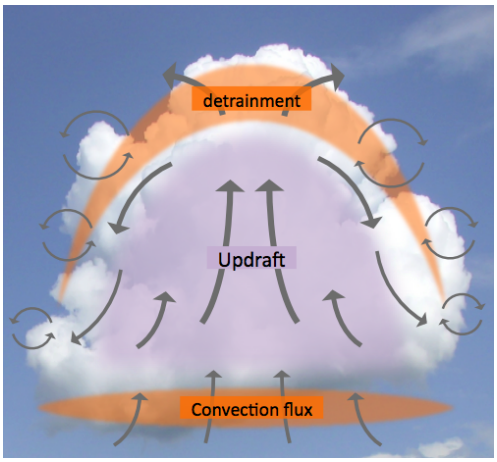

Fig. 1: Schematic representation of a cumulus cloud. The arrows represent wind velocities, the orange blobs denote areas where mixing is occurring between the cloud and the surrounding atmosphere. This representation is very coarse: for instance the updrafts in the center of the cloud are known to behave as "bubbles" when the cloud is young. The cloud dimensions can vary from one to several hundreds of meters. Such a conceptual model encodes various laws, *e.g.* that relate the cloud dimensions to the inner wind speeds.

So far we have concentrated our efforts on the fleet control level, iming at mapping the wind, which is the most relevant variable to guide the UAVs. The various processes required to implement the fleet control are organized along the architecture depicted figure 2. The approach follows a classic sense / plan / act scheme for autonomous systems : the wind measures acquired by the UAVs [7], [8] are integrated in a centralized manner within a map of the cloud using GPR (sense), which is exploited to define the trajectories of the UAVs that optimize the gathering of information while minimizing the energy expense (plan), the trajectories being then executed by the UAVs flight controller (act). This architecture is centralized: the acquired data are sent through a serial link to a ground central station, which runs the mapping and planning algorithms, and the resulting trajectories are uploaded to the AUVs. Details on the mapping and planning algorithms can be found in [9].

III. Simulation overview

The simulation infrastructure must allow the deployment and testing of the same software that is to be eventually deployed on the field. We have developed the overall architecture using ROS (the Robot Operating System [10]), a now well-known framework for robot software development and integration. ROS provides an operating system-like environment, with libraries and tools for heterogeneous computer clusters. Within this scheme we can engineer a modular multi-UAV system with well-defined interfaces such that real and simulated drones run the same software, connecting the necessary independent realistic simulators, and letting the possibility to easily integrate new functions.

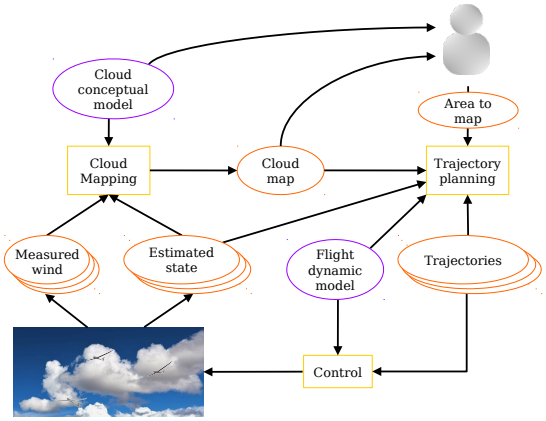

Fig. 2: Overall architecture of the fleet control. Orange, yellow and purple boxes respectively denote information, processes and models. Given a coarse conceptual model of the cloud and the map built by the UAVs so far, the user tasks the fleet with an area to explore. The trajectory planner exploit the cloud map to find the trajectories that optimize the information gathering and the energy expenses, which are sent to the UAVs. Note the importance of the Flight Dynamic Model of the UAVs, which is used by three different processes.

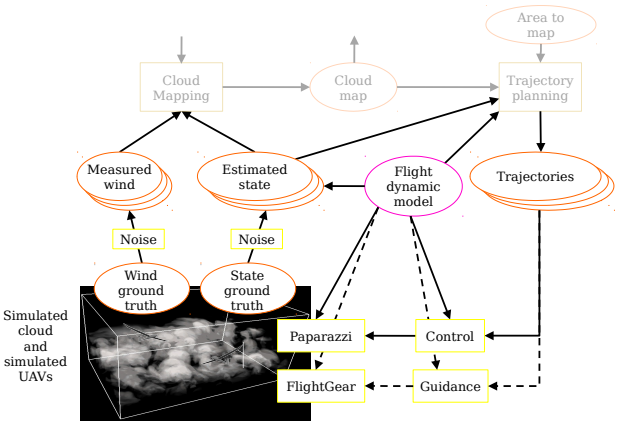

Fig. 3: Schematic view of the architecture of the simulated system. The figure focuses on the simulated processes. The same processes of figure 2 are integrated in the overall simulation, and a second option to control the UAVs using a guidance scheme to follow the trajectories has been added (dashed arrows with the Guidance process).

A. Simulating the environment

Realistic cloud simulations that involve microphysical, dynamical, optical and radiative properties of clouds have been used to produce the evolution of a cloud model. The atmospheric model used for this simulation is Meso-NH [11], a Non-Hydrostatic model with the flexibility to simulate atmospheric phenomena at a wide range of resolutions that extends from one meter up to tens of kilometers. For this work, non-precipitating shallow cumulus clouds over land are simulated with the Large-Eddy Simulation version of Meso-NH. To capture the details about clouds and their surroundings, the atmospheric model has been set at its highest resolution. The considered simulation domain is a cube representing a volume of $4\,\mathrm{km} \times 4\,\mathrm{km} \times 4\,\mathrm{km}$ with spatial resolutions of $10\,\mathrm{m}$, and a time-step of $0.2\,\mathrm{s}$. This setup is a compromise between the desired high resolutions and a reasonable simulation computation time (days of computing on a large cluster are required to run such simulations). The simulation estimates the following atmospheric variables: cloud liquid water content, water vapor, pressure, temperature, and the three components of wind.

The lower left picture of Figure 3 illustrates the 3D cloud water content of simulated convective cumulus clouds at a given time. The overall simulation covers a time period of $15\,\mathrm{hours}$, but the variables of interest have been saved every second only during one hour that corresponds to the maximum of surface fluxes. These variables are the 3D wind components, pressure, humidity, temperature and liquid water content (this latter defining the presence or not of a cloud). As wind is the variable that significantly affects the flight, it is the only one introduced into the ROS architecture loop. Future work will include the rest of the atmospheric variables for cloud mapping purposes – but these variables do not influence the UAVs flights.

B. Simulating the UAVs

Two options have been developed to simulate the UAVs:

- Paparazzi[3] [12] is an open source autopilot and ground control software for small UAVs. It is able to drive both simulated planes, using the JSBsim Flight Dynamic Model (FDM), and real planes with the same interface. It has been chosen as the first flight simulator for our architecture.
- Despite the a priori convenience of Paparazzi, the guidance law it provides is a simple one, and can hardly follow arbitrary trajectories. In order to overcome this problem, we implemented an interface with a second flight simulator: FlightGear[13]. FlightGear is initially designed to manually fly regular planes: it provides minimal controllers for autonomous systems, only a stabilization autopilot. This required the implementation of a guidance control loop based on [14]. Figure 4 shows the main differences between Paparazzi and FlightGear.

The ROS software architecture allows to seamlessly switch from one UAV simulation option to the other.

IV. Simulation Implementation

The overall simulation architecture, further denoted as the "SkyScanner package", is composed of two types of ROS nodes: the ones that are required by the simulation, and the

[3]http://paparazziuav.org

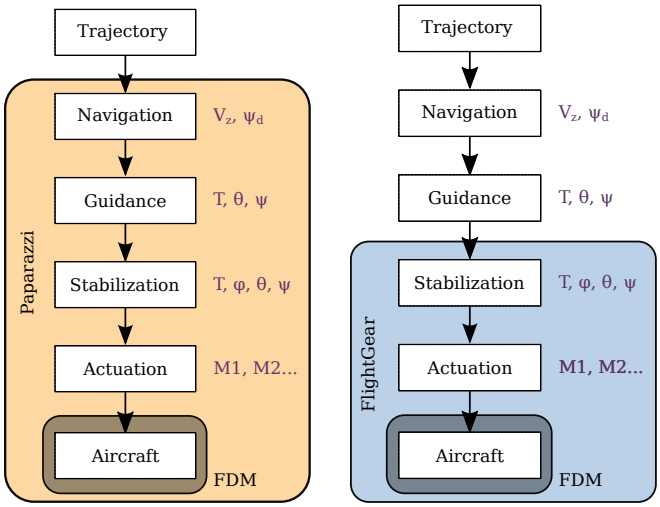

Fig. 4: Two different options to simulate the UAVs. Left, Paparazzi solution, where the full control stack is available and trajectories are tracked using a carrot chasing approach. Right, FlightGear solution: the guidance stage of the control stack must be ensured by an external process to track trajectories.

functional nodes that encapsulate the functional software to evaluate, and that will run on the real system.

A. Interface nodes with simulation backends

This section describes the interface of the simulation architecture with the three simulators. Prior to this, one must introduce a specific time management service to ensure synchronization among all the nodes that compose the system, so that they share timely consistent data.

1) Time management: ROS does not provide dedicated synchronization signals between nodes, yet it exploits a *clock* topic which uses by default the computer's system clock, known as "wall-clock". This can be changed by a node publishing in this topic, which allows to have a fine control of simulation execution by setting the speed of time, *e.g.* to run faster than real time simulations if the simulators allow it, or on the contrary to pace the system at the speed of the slowest simulator.

The SkyScanner package includes a node *clock_generator* that handles ROS time. It sets the ROS clock by publishing the wall-clock multiplied by a constant into *clock*, so ROS sees time faster or slower than the computer. This constant can be set arbitrarily in the ROS parameter server, and time can be stopped and resumed at any time by sending a message to the *clock_control* topic.

Using the ROS simulated time, the execution is synchronized at two speeds with clock messages on these topics (figure 5):

- *tick*, running at 1 Hz, for data gathering and path planning.
- *fast_tick*, running at 50 Hz, for the UAV guidance and control loops.

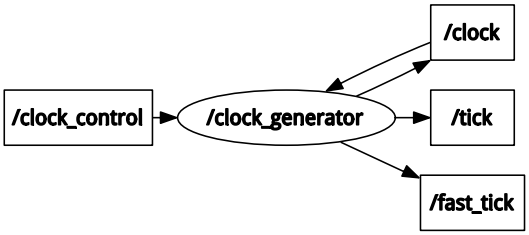

Fig. 5: Clock generator ROS graph

2) MesoNH: The *atmosphere* ROS node, which runs the *get_wind* service, queries wind speed from MesoNH off-line simulated data, stored in a hard drive. Then, the *paparazzienivironment* and *flightgearenvironment* nodes feed the wind into the respective flight simulators. As shown in fig. 6, these two nodes provide position and time to *atmosphere* and put the corresponding wind speed into the simulation. Any other flight simulator could be attached following the same procedure.

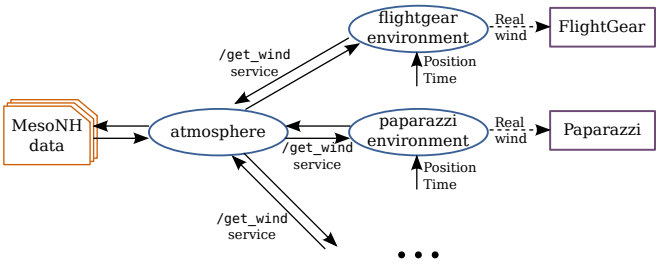

Fig. 6: Interface between MesoNH data (the cloud environment) and the two flight simulators

3) Paparazzi: Communication between UAVs and Paparazzi ground control software is done through the Ivy bus protocol[4] [15]. Involved systems put on the bus messages that are broadcasted along this channel. Then, subscribers filter the information they are interested in by using regular expressions. Each Paparazzi aircraft sends periodical telemetry messages with its position, attitude, energy consumption and task execution status (among others). The ground control station or other custom software subscribes to these messages and put back in the up-link commands for the planes.

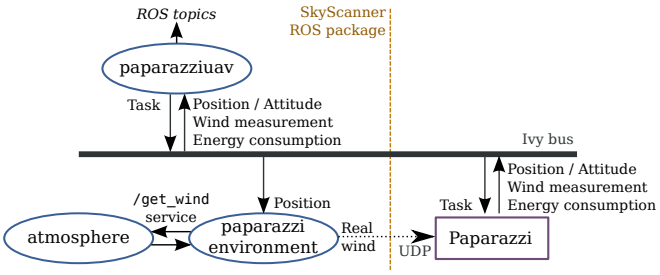

Fig. 7: Interface scheme between Paparazzi and SkyScanner ROS package

[4]http://www.eei.cena.fr/products/ivy/documentation/ivy/index.html

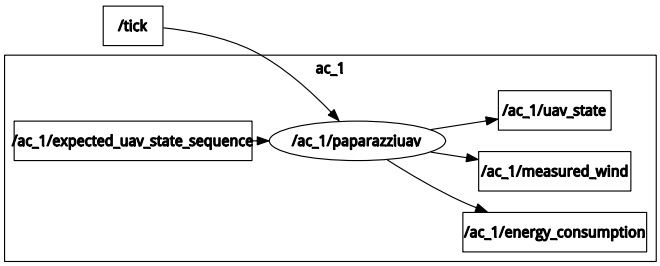

Fig. 8: ROS graph of paparazziuav

Fig. 7 shows how the SkyScanner ROS package communicates with the Ivy bus and exposes the data of interest into ROS topics (possibly after having added noise to the wind and state estimates). Paparazzi puts on the bus all the messages received from the UAVs telemetry link. This includes position, attitude and energy consumption which are the ones the *paparazziuav* node reads. As the information from all aircrafts is in the same channel, each node filters the ones that matches the matching aircraft id. Fig. 8 shows the ROS graph of a *paparazziuav* node and related topics.

On the other hand, *paparazziuav* puts task messages which can be waypoints, circles, segments and a combination of those. Given that the trajectories produced by the path planner are more complex [9], Paparazzi path options are quite restricted. Moreover, Paparazzi uses a carrot-chasing guidance algorithm: if this guidance law is suitable for tracking steady circular and linear trajectories, it does not perform well for the complex trajectories regularly updated by the trajectory planning node (see section IV-B.1). Approximating these trajectories with a sequence of close waypoints or short line segments did not show satisfactory trajectory tracking behavior.

4) FlightGear: Due to the trajectory following issues of Paparazzi, we explored an other option and implemented a back-end for the open source FlightGear flight simulator. FlightGear was selected for the following reasons:

- It can use JSBSim as Flight Dynamic Model library, the same one as Paparazzi,
- It comes with numerous aircraft models,
- Most of simulation parameters, commands and states can be controlled externally with remote protocols.

FlightGear simulation state, inputs, outputs and settings are organized as a categorized tree. Every value can be read and modified from the program itself or by an external software. Communication between FlightGear and other software can be achieved by UDP, TCP, HTTP or Telnet. HTTP and Telnet servers in FlightGear have a slow update rate, but are easy to use. HTTP allows to show data in a web browser and Telnet provides the capability of monitoring and changing values through the command line using client software available in all operating systems. On the other hand, communication is faster using UDP and TCP: since data transfer between FlightGear and the SkyScanner ROS package should happen at 50 Hz at least, the UDP protocol was chosen over TCP because it is stateless.

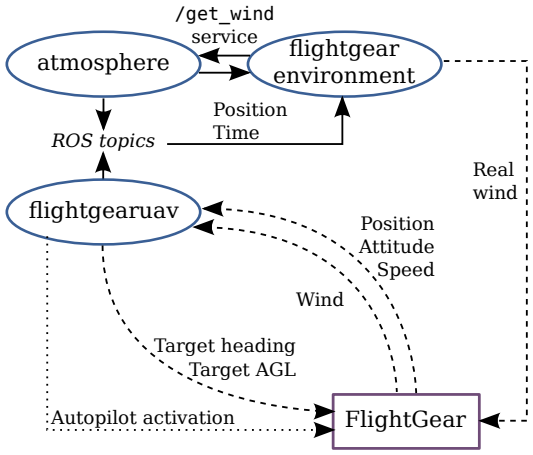

Fig. 9: Interface scheme between FlightGear and SkyScanner ROS package

The interface between FlightGear and the SkyScanner package is shown Figure 9. Dashed lines correspond to the values sent by UDP. To each line is associated a socket: position, attitude and speed are received together at *fast_tick* frequency; the heading and height Above Ground Level setpoints are transmitted via an other socket at the same frequency; and finally wind is sent and measured back in two different channels running at 1 Hz. Other control messages that are not sent regularly, like autopilot activation or deactivation, are transmitted using Telnet, and are shown as dotted lines in fig. 9.

Furthermore, an important difference between Paparazzi and FlightGear interfaces is that FlightGear sets a *target heading* and a *target AGL*. This is due to the fact that Paparazzi is a whole autonomous flight control system and FlightGear just a flight simulator. As a consequence, to use FlightGear we had to introduce the *guidance* node which translates geometric trajectories into heading and height AGL setpoints (see section IV-B.3).

B. Functional nodes

1) Path planning and mapping integration: The path planner and mapping libraries have been developed in previous stages of the project [9], but were not designed to work in a time managed environment, nor to communicate with realistic simulators. Because of the amount of data exchanged between these two libraries (the current map of the cloud), we opted to integrate them in a single *pathplanner* ROS node, that takes as inputs for mapping and planning the wind and UAV state estimates, and outputs the trajectories.

At each *tick* message, the module updates the mapping according to the received data. The mapping and planning algorithms have not been optimized and do not operate in real-time, hence some time management is needed. The loop works as shown in Fig. 10 for every drone in the fleet. While a plan is being executed, the next one is created. At the beginning of each cycle, e.g. $t = t_0$, the state of planner's virtual UAVs are updated to the expected position at $t_0 + \Delta t_p$, where Δt_p is the planner's cycle period. This time is actually

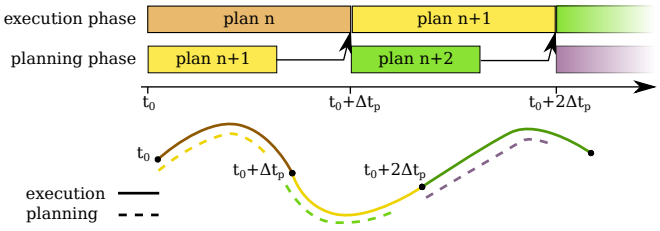

Fig. 10: Planning chronogram

the fraction of the planning that is executed. Then, a new plan for the next period is generated from this base position.

The time horizon of the planned trajectories depends on the desired optimization level and is not formally bounded, but it is limited in practice to 10 seconds, which is a sufficient time span to plan new trajectories, while not being too far in the future, where the map becomes unreliable.

Fig. 11 presents the input and output topics related to the *pathplanner* node. It takes the position and wind measurements from each aircraft namespace. After processing the data, the library gives two types of output: a set of commands for the plane, turn radius and propulsion power; and a sequence of expected states, position and attitude, which forms the trajectory to track. Using these outputs, the node forms a trajectory expressed as mathematical functions, which are fed to the guidance controller (encoding trajectory as time valued mathematical functions is a requirement from the chosen guidance law [14]).

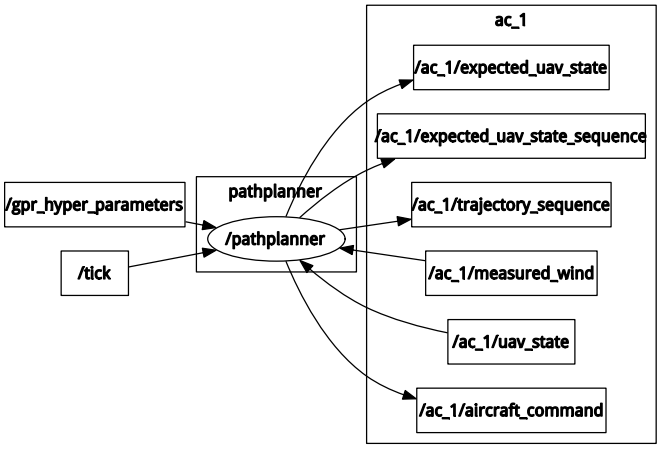

Fig. 11: Path planner ROS graph

2) Improving mapping through machine learning: The mapping algorithm computes a map of the atmosphere using GPR. Its prediction accuracy depends on some process parameters that can be improved from observed data. This optimization is a very time-consuming operation, and is therefore integrated within a different node: the *gpr_optimizer* node. Fig. 12 shows its inputs, wind samples from each UAV; and outputs, the new GPR hyperparameters, which are then exploited by the mapping algorithm. The execution of this optimization process is not mandatory, so the implementation as an independent ROS node allows to not launch it if we want to reduce the computer load.

While the current implementation of this node is very specific to the GPR mapping approach, it is an instance of a low-rate complex procedure. Other learning or optimization algorithms that behave similarly could be integrated within the same architecture.

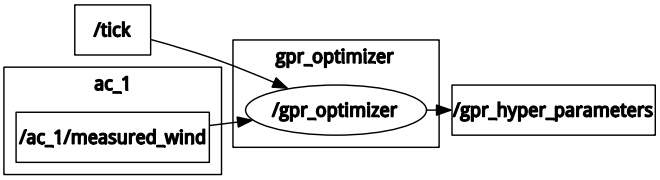

Fig. 12: GP optimizer ROS graph

3) The guidance node: The *guidance* node, fig. 13 takes the mathematical expression of a trajectory and the actual position of the aircraft to compute the heading that will lead the UAV to the right path, using the law presented in [14]. This node first determines which part of the track to follow and accordingly computes headings. The Height AGL setpoints are directly taken from the expected position, leaving the task of tracking it to FlightGear, which is able to do so.

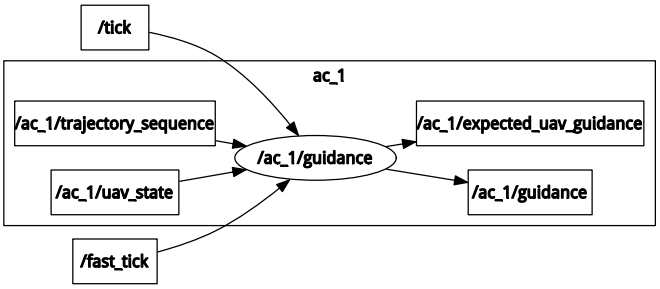

Fig. 13: *guidance* node ROS graph

C. Deploying multiple UAVs

The main reason of designing this architecture is to be able to deploy easily a reasonable number of UAVs, not necessarily homogeneous: each UAV can have its own set of parameters. Every aircraft to be launched runs in a different namespace. Involved nodes, topics and parameters are preempted by an aircraft code so that no conflicts occur. Common nodes like *paparazziuav* or *paparazzienvironment* nodes run on their own namespace but they read and put messages on the aircrafts namespaces. Note also that the system could work with aircrafts running in different flight simulator backends.

Figure 14 shows all the developed ROS nodes to implement the simulation of the SkyScanner package, and Figure 15 shows the whole ROS architecture for the UAV *ac_1* ("AirCraft 1"). The source code of the ROS nodes that encapsulate the Paparazzi and FlightGear simulators can be found in this repository[5].

[5]https://github.com/rafael1193/skyscannner_integration

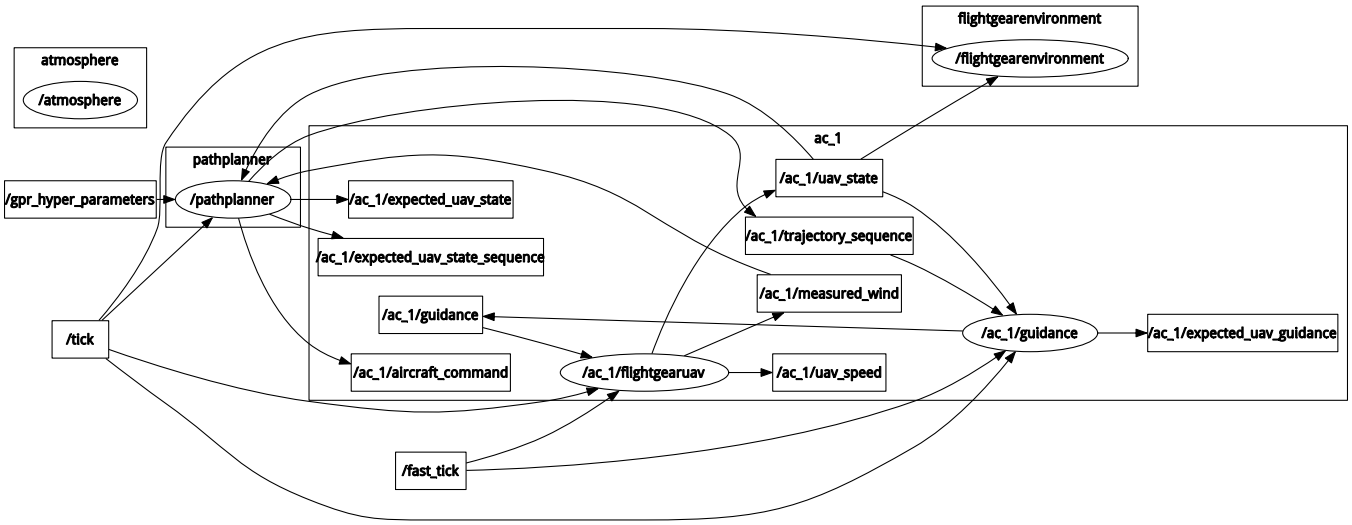

Fig. 15: ROS graph of the complete FlightGear execution loop.

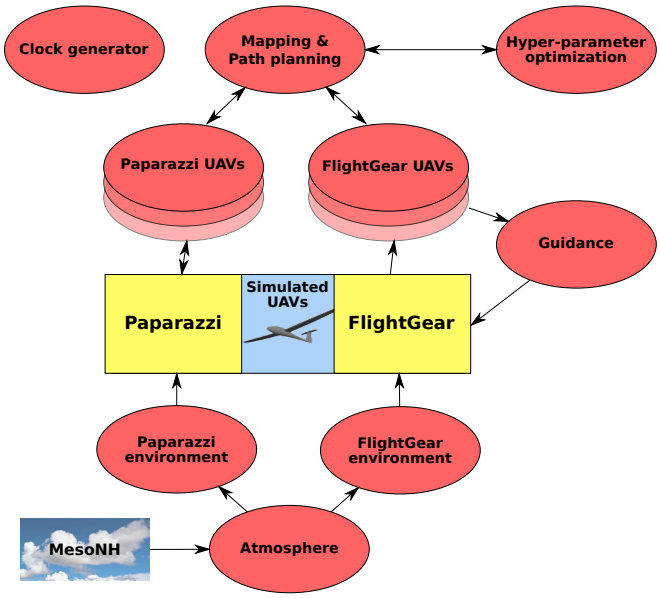

Fig. 14: All the ROS nodes involved in the SkyScanner package when running in simulation (blue ellipses). This figure depicts the ROS implementation of the conceptual architecture shown in figure 3.

D. Illustration

The overall architecture has been fully implemented, and can be used to evaluate the developed algorithms to achieve the cloud probing mission. Figure 16 show an example of the achieved trajectory in a simulation trial. It is actually thanks to the simulation that the poor performance of Paparazzi to properly track complex trajectories has been assessed, which called for the development of the guidance law presented in [14], and the use of the FlightGear flight simulator. Indeed, before the fully integrated simulation, the cloud mapping and trajectory software, which constitute the heart of the system, were simulated in an open-loop fashion, the planned trajectory being considered faithfully executed by the UAVs: such a non-integrated simulation does not allow to assess the overall system integration issues.

V. Conclusion

This paper presented the development of a system simulation infrastructure for a fleet of UAVs designed to probe clouds, according to an adaptive sampling scheme. Such an infrastructure is essential to validate software integration before field trials: it allows to evaluate the required communication bandwidths, the proper functioning of each developed functionality, and their sound integration. It also allows to assess statistical performances of the developed algorithms, by thoroughly testing the developed system under various circumstances, which span from the definition of algorithms parameters to the simulated environments. In this work, the use of ROS to integrate all the developed functions and the simulators has shown to be very efficient, particularly by allowing easily to control time.

Future work will consider the development of a distributed architecture, in which the cloud mapping and trajectory planning processes would run on-board the UAVs, thus removing the need for a permanent communication link with the ground station. The proposed simulation infrastructure will undoubtedly play a key role to support these developments and assess their validity: thanks to its modular structure, its evolution can easily be foreseen.

References

[1] J. Elston, B. Argrow, M. Stachura, D. Weibel, D. Lawrence, and D. Pope, "Overview of small fixed-wing unmanned aircraft for meteorological sampling," *Journal of Atmospheric and Oceanic Technology*, vol. 32, pp. 97–115, 2015.

[2] J. Elston and B. Argrow, "Energy efficient UAS flight planning for characterizing features of supercell thunderstorms," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 6555–6560.

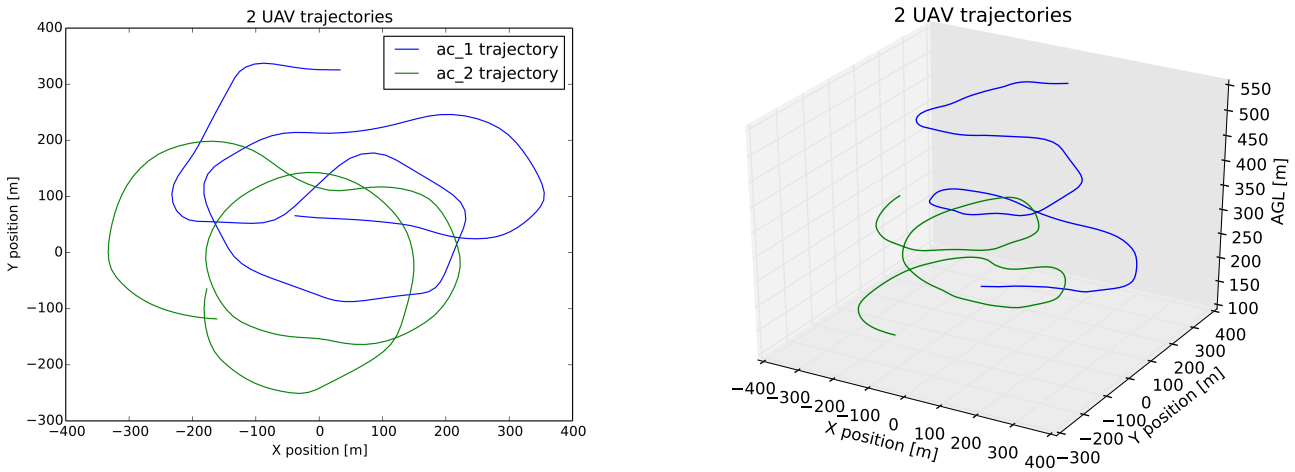

Fig. 16: Trajectories resulting from the simulation of the exploration of a cloud by two UAVs

[3] N. R. Lawrance and S. Sukkarieh, "Autonomous exploration of a wind field with a gliding aircraft," *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 3, pp. 719–733, 2011.

[4] S. Ravela, T. Vigil, and I. Sleder, "Tracking and Mapping Coherent Structures," in *International Conference on Computational Science (ICCS)*, 2013.

[5] J. W. Langelaan, J. Spletzer, C. Montella, and J. Grenestedt, "Wind field estimation for autonomous dynamic soaring," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012.

[6] M. Michini, M. A. Hsieh, E. Forgoston, and I. B. Schwartz, "Robotic tracking of coherent structures in flows," *Robotics, IEEE Transactions on*, vol. 30, no. 3, pp. 593–603, 2014.

[7] J.-P. Condomines, M. Bronz, G. Hattenberger, and J.-F. Erdelyi, "Experimental wind field estimation and aircraft identification," in *International Micro Air Vehicles Conference and Flight Competition (IMAV), Aachen (Germany)*, Sept. 2015.

[8] L. Rodriguez, J. Cobano, and A. Ollero, "Wind characterization and mapping using fixed-wing small unmanned aerial systems," in *International Conference on Unmanned Aircraft Systems, Arlington, VA (USA)*, June 2016.

[9] C. Reymann, A. Renzaglia, F. Lamraoui, M. Bronz, and S. Lacroix, "Adaptive sampling of cumulus clouds with a fleet of UAVs," *Autonomous robots*, Jan. 2017.

[10] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3. Kobe, Japan, 2009.

[11] J. P. Lafore, J. Stein, N. Asencio, P. Bougeault, V. Ducrocq, J. Duron, C. Fischer, P. Héreil, P. Mascart, V. Masson, J. P. Pinty, J. L. Redelsperger, E. Richard, and J. Vilà-Guerau de Arellano, "The Meso-NH Atmospheric Simulation System. Part I: adiabatic formulation and control simulations," *Annales Geophysicae*, vol. 16, no. 1, pp. 90–109, 1998.

[12] G. Hattenberger, M. Bronz, and M. Gorraz, "Using the Paparazzi UAV System for Scientific Research," in *International Micro Air Vehicle Conference and Competition*, 2014.

[13] A. R. Perry, "The flightgear flight simulator," in *Proceedings of the USENIX Annual Technical Conference*, 2004.

[14] H. G. de Marina, Y. A. Kapitanyuk, M. Bronz, G. Hattenberger, and M. Cao, "Guidance algorithm for smooth trajectory tracking of a fixed wing UAV flying in wind flows," in *IEEE International Conference on Robotics and Automation, Singapore*, 2017.

[15] S. Chatty, "The ivy software bus," ENAC, Tech. Rep., 2003.