ROS/Gazebo based simulation of co-operative UAVs*

Cinzia Bernardeschi[1], Adriano Fagiolini[2], Maurizio Palmieri[3], Giulio Scrima[1], and Fabio Sofia[1]

[1] Department of Information Engineering, University of Pisa, Italy
cinzia.bernardeschi@unipi.it
giulio.scrima@gmail.com
fabiosofia@hotmail.it
[2] Department of Energy, Information Engineering and Mathematical Models, University of Palermo, Italy
adriano.fagiolini@unipa.it
[3] Department of Information Engineering, University of Florence, Italy
maurizio.palmieri@ing.unipi.it

Abstract. UAVs can be assigned different tasks such as e.g., rendezvous and space coverage, which require processing and communication capabilities. This work extends the architecture ROS/Gazebo with the possibility of simulation of co-operative UAVs. We assume UAV with the underlying attitude controller based on the open-source Ardupilot software. The integration of the co-ordination algorithm in Gazebo is implemented with software modules extending Ardupilot with the capability of sending/receiving messages to/from drones, and executing the co-ordination protocol. As far as it concerns the simulation environment, we have extended the world in Gazebo to hold more than one drone and to open a specific communication port per drone. In the paper, results on the simulation of a representative co-ordination algorithm are shown and discussed, in a scenario where a small number of Iris Quadcopters are deployed.

Keywords: ROS/Gazebo · Co-operative UAVs · Simulation.

1 Introduction

Simulation applied at the early stages of system design allows developers to gain confidence that the system behaves as expected and it is an important tool to visualize and validate systems before industrial production or deployment. Many different languages and environments have been introduced to support modeling and simulation.

* Postprint. Published in: Mazal J. (eds) Modelling and Simulation for Autonomous Systems. MESAS 2018. Lecture Notes in Computer Science, vol 11472. Springer, Cham. The final authenticated publication is available online at https://doi.org/10.1007/978-3-030-14984-0

During the last decades, much research has been carried out showing the potentialities of multi-robot systems in many real applications, ranging from precision farming, surveillance, patrolling, etc. [2, 7, 21, 29].

ROS (Robot Operating System) [28] is a standard de facto for robot software development. Analogously, for simulation of UAV aircraft, Gazebo [12] has been widely used in the scientific community. In Gazebo, Ardupilot [3] is the open-source software that allows to carry out the control of different unmanned vehicles mainly through its four different components: the Antenna Tracker, the APM Rover, the ArduPlane and the Arducopter. In particular, Arducopter implements the actual drone control. So far, the current ROS/Gazebo architecture only allows for the simulation/emulation of a single aircraft

In this work we present a possible extension of the architecture enabling the simulation of possibly multiple heterogeneous vehicles, adhering to their own individual dynamics, as well as interacting with each other according to shared co-operation strategies. In particular, we consider UAVs with an underlying attitude controller based on Ardupilot, which uses the MAVLink protocol (Micro Air Vehicle Link) [6] for the communication. The integration of the co-ordination algorithm in Gazebo is implemented with software modules extending Ardupilot with the capability of (i) sending/receiving MAVLink messages to/from drones, and (ii) executing the co-ordination protocol. An abstraction of the communication channel by which drones exchange information is implemented with a co-ordination script, which is executed locally by each drone instance. Every fixed time interval a drone sends information (e.g. actual position) to other drones. Each drone uses the data received from the other drones via the co-ordination script to compute a new target point, based on the task the drones have to perform. A case study has been developed, where a small number of Quadcopters are deployed and perform space-coverage operations by applying a slightly modified version of the Olfati-Saber et al. co-ordination algorithm [13].

The paper is organized as follows. Section 2 reports related work. Section 3 provides a short overview of the state of the art of the ROS/Gazebo simulation environment. Section 4 describes the modifications made to the simulation environment to allow multi-UAVs simulation. Section 5 shows the application of the developed framework to a case study. Finally, Section 6 contains a discussion on the presented framework and the conclusions.

2 Related Work

The problem of co-ordination of UAVs has been addressed in many works. Among others, the works [25] and [26] report on co-ordination systems by focusing on the problem of optimizing the path and/or time of flights to cover an area of interest. The works [19, 23] and [27] report on the problem of tracking a target moving on the ground. In the context of civil security for disaster management, [11] introduces a decisional architecture for co-ordination of multiple UAVs (such system was developed in the framework of the AWARE Project[14],

which considers scenarios of surveillance with multiple UAVs, sensor deployment and fire threat confirmation).

The work [24] reports on a further application of co-ordination and surveillance of UAVs; in this case co-ordination algorithms are applied inside a urban environment, with particular attention to surveillance approaches of a territory.

In [5] the complexity in the co-operation and co-ordination of independent UAVs is studied: clustering techniques are applied for the division of aircrafts in sub-teams according to the objective which they have to achieve.

The ROS/Gazebo simulation environment is a complex framework that involves many different elements and allows realistic simulations of robotic systems. Other simulation environments are available, like for example [20], where a control and co-operation strategy exploiting the CATA, Control Automation and Task Allocation, is used; or [10], where Matlab/Simulink is used for the real time control of UAVs. These environments are easier than ROS/Gazebo but they do not allow a graphical representation of the UAVs.

3 ROS/Gazebo Simulation Environment

The ROS/Gazebo development simulation environment involves three main elements:

- the open-source autopilot software Ardupilot [3];
- the collection of software frameworks for robot programming ROS (Robot Operating System) [16];
- the 3D simulation environment Gazebo[15].

In particular, the UAVs attitude controller is based on the Ardupilot software. ROS is instead exploited like a middleware to help programmers in developing robot applications. Gazebo allows a visual, tridimensional simulation of a scenario consisting of cyber-physical systems, e.g. ground-rovers, UAVs, and different other objects, like, e.g., simple obstacles, or surrounding environmental elements, together composing what it is called a Gazebo world.

3.1 Ardupilot

The base element that comes into play to start a software simulation of a single vehicle is the open-source software Ardupilot (http://ardupilot.org/). Developed by the community DIY Drones, it allows to carry out the control of different unmanned vehicles, including drones.

The Ardupilot software offers different flight modes, which can be distinguished in manual flight modes and automatic ones, with a combination of customizable parameters. For instance, it is possible to control the UAV flight through a guided mode, by which Ardupilot will adjust automatically the values of yaw, roll, and pitch, according to the position given as input. Moreover, through other flight modes, it is possible for the UAV to keep the desired altitude, a given position, to move the UAV with a circle trajectory (fixing the

radius of the trajectory), to make the UAV come back to the launch point or simply land, and many others.

For the communication with the unmanned vehicle, the MAVLink protocol [6] is exploited. It allows vehicles communications through the exchange of packets, which are represented at low level by strings in C language, in which each bit has a specific function in the communication. These packets are provided with header, payload, and checksum, and are transmitted through some serial communication channels. Through the MAVLink protocol it is possible to exchange some predefined commands: the fundamental heartbeat message (characterized by the `ID 0`), which is used to keep under control the communication state with the UAV, or the set_mode message (characterized by the `ID 11`), which is used to set a given flight mode, together with many other messages which are useful to control the vehicle. Moreover, it is possible to create custom messages.

3.2 ROS

ROS is not actually an operative system, but rather it represents an open source collection of frameworks/libraries [18] for the development of software for the programming of robots. ROS was developed with the objective of facilitating and expediting the prototyping of a robotic software. In particular, the principle of software reuse is exploited, enabling interoperability among all the tools involved with the ROS environment and solving problems such as real-time collection of data from cyber-physical systems sensors, implementation of the publish/subscribe model in a network of ROS nodes for the communication from/to robots, management of commands received by a user and related actuation actions.

ROS provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is based on FreeBSD, the open source operating system developed by Berkeley Software Distribution. In ROS, the components of the robots can be represented like nodes in a network, which communicate one another, via the anonymous and asynchronous publish/subscribe mechanism. This is a powerful design pattern that can significantly reduce the development effort and promote flexibility and modularity in a system.

Another important element in ROS is the definition of the physics of the robot, and a part of tools was developed to this aim. In particular, it is possible to use the URDF (Unified Robot Description Format) files to describe the robot physical parameters. This improves the integration with 3D simulators, like e.g. Gazebo[15].

3.3 Gazebo

Exploiting a visual simulator is useful in case one wants to test the functioning of a given algorithm. In Gazebo [15], realistic scenarios for cyber-physical systems, including the surrounding environment[8], can be created. This simulator is complete with dynamic and kinematic physics, and a pluggable physics engine.

Integration between ROS and Gazebo is provided by a set of Gazebo plugins that support many existing robots and sensors. Since the plugins present the same message interface as the rest of the ROS environment, a Gazebo user can write ROS nodes that are compatible with simulation, logged data, and hardware. A relevant aspect is that a user can develop an application directly in the simulation environment and then deploy the physical robot with little or no changes at all in the code. Simulation of UAV aircraft through Gazebo has been widely used in the scientific community.

4 Multi-UAV Simulation

This section shows the approach we have followed to extend the base environment of Gazebo with the simulation of multiple robotic systems. The architecture of the simulation environment has been modified since the connection between Ardupilot and ROS/Gazebo is provided through a unique port, while, to have a multi-vehicle simulation, it is indispensable the creation of a number of connection ports equal to the number of UAV instances.

Moreover, the Ardupilot software was enhanced with some modules allowing drones exchanging MAVLink messages to each other directly.

Finally, Gazebo was extended to allow the graphical visualization of different UAVs, which cooperate communicating with each other, by means of the MAVLink protocol using the previously input/output inserted ports, in order to execute the chosen co-ordination protocol for the specific application (e.g., a space coverage application).

More precisely, our extension uses:

- the guided mode for controlling the UAV flight, by which Ardupilot adjusts automatically the values of yaw, roll, and pitch, according to the target position given as input to the drone;
- the Python script `mavproxy.py`, which is used for configuring the communications among the unmanned vehicles using the MAVLink protocol;
- the Python script `simvehicle.py`, which is used to create instances of vehicles with their base parameters and their MAVLink connections, and to control such instances. The `simvehicle.py` script contains also the algorithm to be executed locally for the implementation of the co-ordination protocol.
- the SITL (Software In The Loop) simulator which allows us to run Ardupilot without any physical hardware, emulating the behavior of the drones;
- the command `roslaunch` to run the Gazebo simulator, together with a launch file holding the Gazebo environment, i.e. all the elements within the so-called Gazebo world, and one or more simulated drones in such an environment.

Figure 1 shows the architecture of our simulation for the case study in Section 5: there is the Gazebo world, which is activated by the `roslaunch` command, there are N drones, each of which is activated by a `simvehicle.py` script that

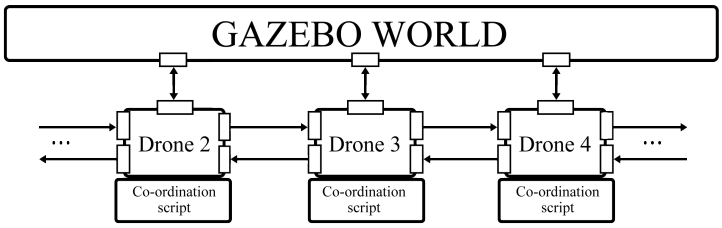


Fig. 1: Multi-UAVs architecture for the case study in Section 5.

has been enhanced with a local co-ordination script to enable the data exchange among drones and the execution of a co-ordination algorithm.

To carry out the simulation of cooperative UAVs we use a fixed, defined apriori, number of Iris Quadcopters. These quadcopters, developed by 3D Robotics, represent commercial, state-of-art drones which can be exploited in the professional sector for different base applications. In particular, Iris UAVs hold the typical equipment of a drone, including the remote controller, radio for communications, Wi-Fi card for control using Android devices, high-resolution cameras, and other useful sensors. The Ardupilot software runs on top of the Pixhawk flight controller board [17]. Even though Iris drones have the limitation of a time of flight of about 20 minutes, which is, however, a typical feature of commercial drones, the rationale behind these quadcopters was their simplicity of programming and usage.

4.1 Base Environment Modifications

To allow the creation of the extension described in the previous subsection, some modifications to the base development/simulation environment Ardupilot-Gazebo were needed. In particular, for the multi-UAV extension, the following versions of the environment tools were exploited:

- version 16.04 of Ubuntu;
- the most recent version of Ardupilot;
- ROS Kinetic;
- version 8 of the Gazebo simulator, together with the `ardupilot_sitl_gazebo` plugin for the integration of the two environments (Ardupilot and Gazebo), in a context in which the emulation of drones by means of SITL is exploited, i.e. we can work even in the absence of hardware.

First of all, it was necessary to make some modifications to the Iris drone configuration files, so as to allow the presence of different, uniquely identified drones. To this objective, we modified, in the base configuration files, the model name exploited for the insertion of the particular vehicle in the Gazebo world,

and the input/output communication ports for each Iris drone, so as to have them different for each UAV. This enhancement allows a direct information transfer among drones. This will be described in a detailed way in the following subsections.

After having created different Iris UAV instances, we created a Gazebo multi-UAV world, i.e. a world that could contain all the created instances together. To this aim, all the single models, identified in a unique way through the model name, were inserted, each one in a given position, which was chosen in the Gazebo world configuration according to the co-ordination algorithm of the application object of the simulation.

For the case study reported in Section 5, the Gazebo world with multi-UAVs we created is shown in the code below. For each UAV instance, identified through the `model name` field, information about the initial position (`pose` field) inside the Gazebo world and the physical model of the drone (`uri` field) are provided. For instance, the first drone, identified by the name `iris_uav_instance1`, is placed at position `(0,0,0)`, and it is a Iris UAV (`model://iris_with_standoffs_demo`).

```
<model name="iris_uav_instance1">
  <pose>0 0 0 0 0 0</pose>
  <include>
    <uri>model://iris_with_standoffs_demo</uri>
  </include>
</model>
<model name="iris_uav_instance2">
  <pose>0 10 0 0 0 0</pose>
  <include>
    <uri>model://iris_with_standoffs_demo</uri>
  </include>
</model>
<model name="iris_uav_instance3">
  <pose>0 20 0 0 0 0</pose>
  <include>
    <uri>model://iris_with_standoffs_demo</uri>
  </include>
</model>
<model name="iris_uav_instance4">
  <pose>0 50 0 0 0 0</pose>
  <include>
    <uri>model://iris_with_standoffs_demo</uri>
  </include>
</model>
<model name="iris_uav_instance5">
  <pose>0 100 0 0 0 0</pose>
  <include>
    <uri>model://iris_with_standoffs_demo</uri>
  </include>
```

```
</model>
```

Then, for the purposes of applying the co-ordination algorithm to the drones swarm, whose number is apriori defined, some small changes have been made to the Ardupilot Python script `sim_vehicle.py`, for the control of simulated vehicles. Finally, we extended the script `mavproxy.py`, which manages the commands given to drones, using dedicated options.

4.2 Co-ordination Algorithm

For the execution of the co-ordination algorithm:

- we implemented a Python script (named co-ordination script in the follow), executed locally by each drone. Each UAV performs the operations envisaged by the algorithm and communicates with the other drones. The Python script is implemented using the Python Dronekit[1] library, through which it is indeed possible to connect and communicate to a drone. The co-ordination script will be described in detail in subsection 5.1.
- we added a port in `sim_vehicle.py` to communicate with the co-ordination script, to get real-time information about the position of drones.
- we defined a new command in `mavproxy.py` in order to start the co-ordination script in an automatic way. In particular, a new function was written in the code, to start a new non-blocking process, executing the co-ordination script.

For the computation of the distance between two drones, the Haversine formula[22] is used, which is the one shown below. Considering two points, each one at a given latitude and longitude, their distance will be given by:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2-\phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right) \tag{1}$$

where ϕ_1 and ϕ_2 are, respectively, latitude of point 1 and latitude of point 2, and λ_1, λ_2 are, respectively, longitude of point 1 and longitude of point 2. Moreover, `r` is the radius of the earth.

Formula (1), which is commonly exploited in navigation, allows computing the distance between two objects in the Earth, known the positions in the form of geographic coordinates, i.e. the couple (latitude, longitude).

5 An Application Scenario

Among the many possible interaction policies, we focus on the problem of co-ordination of a team of drones, and we present a variation of the classical formation control scheme, based on the well-known consensus protocol described e.g. in [13]. The algorithm in [13] is distributed and allows drones to asymptotically converge to a target point. The co-ordination algorithm we simulate, instead, is obtained by the original one, simply assuming that two drones are fixed at

the extreme of a line segment. This variant allows the uniform placement of the drones along the interval and it has not been studied in that work.

As an application scenario, we considered the case of 5 drones that are supposed to coordinate on the interval $[0, 100]$. The first and fifth vehicles are supposed to be stationary at the outer position of the interval, while the other three must recursively adjust their positions according to the shared co-ordination policy.

5.1 Co-ordination Script

The co-ordination script involves the following operations:

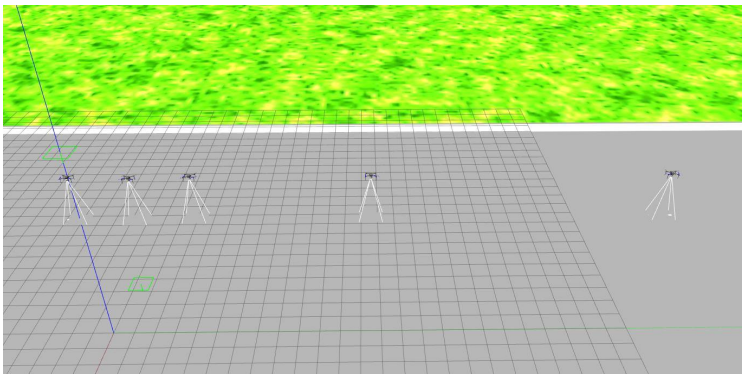
- a connection to the actual drone instance is created, through which it is possible to acquire information by the UAV about its position. In our case, we use only the longitude. The same information will be sent by this drone instance to the closest drone on the left and to the closest drone on the right.
- a control about the type of drone instance, fixed drone (stubborn) or mobile drone, is added to the code. Indeed, in case the drone is a stubborn one, the script will terminate, since fixed drones do not have to perform any movement, according to our modified version of Olfati-Saber et al. algorithm, but they have rather to be fixed in the position assigned to them. Mobile drones have to change their position according to the formula in the previous paragraph. The control on the drone instance is made exploiting the port in the UAV-address which is passed to the script, which identifies uniquely such instance. According to the architecture, the ports related to the single aircraft, and thus their ordering in space, are already known apriori, whereas their relative (geographical) position, at the beginning of the execution of the co-ordination script and in any subsequent instant, are not known.

In the first phase of the script execution, some functions are used to arm drone motors and to make them take off until they reach a height of 10 meters, which represents an arbitrarily-chosen height. Then the script contains a loop where each iteration computes a step of the co-ordination algorithm, using the positions of adjacent drones. In particular, a reference longitude is used, the one of the first stubborn UAV, which is considered for the application of the algorithm, and the distance of the other UAVs is computed in relation to this stubborn. Each drone executes the co-ordination algorithm using the actual position of its adjacent drones and its own desired position.

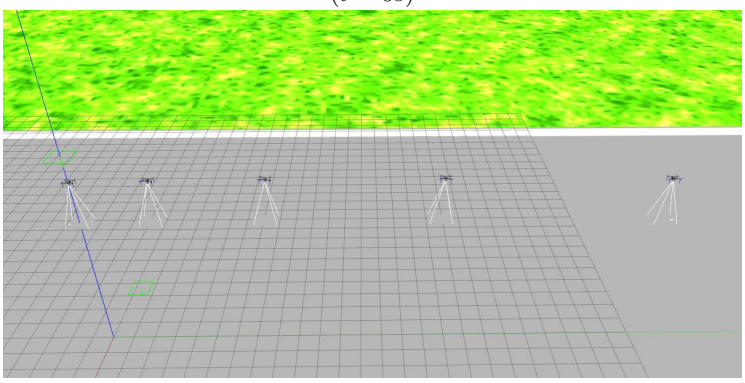
5.2 Simulation

A typical simulation scenario is reported in Figure 2. Assuming as a reference the leftmost UAV in position 0, in the initial deployment the second, third and fourth drones, are, respectively, placed 10, 20, and 50 meters after the first one. The last UAV is placed at the last extreme of the line segment, i.e. 100 meters after the first drone. For each of the five drones, the `sim_vehicle.py` script

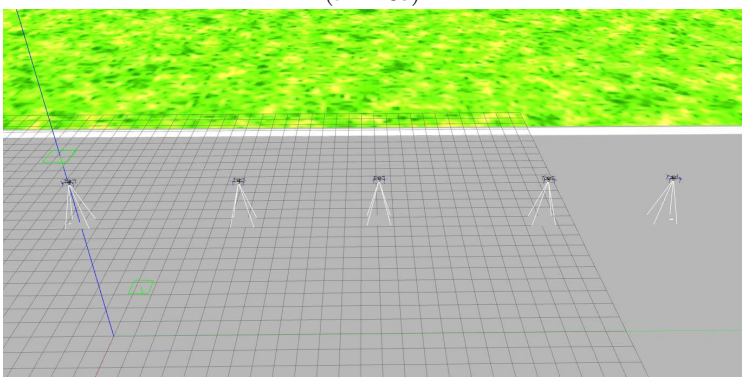
is launched, in order to create the corresponding MAVLink connection for the control of the vehicle, and the launch file containing the Gazebo world is executed through the `roslaunch` command. At this point, Gazebo will start, and the 3D simulation environment will show the five drones in their initial placement. The co-ordination script can now be launched. Aircraft will arm their motor and will take off at the predetermined height of 10 meters, and each mobile drone will start to communicate with the adjacent UAV and to make some position change, according to the formula shown above. Figure 2 shows the position of drones at the beginning (time $t = 0s$), in the middle (time $t = 28s$), and at the end (time $t = 51s$) of the simulation.



($t = 0s$)



($t = 28s$)



($t = 51s$)

Fig. 2: Dynamic behavior of a team of cooperative UAVs: a) initial deployment at $t = 0$ seconds, b) intermediate displacement at $t = 28$ seconds, and c) final deployment at $t = 51$ seconds.

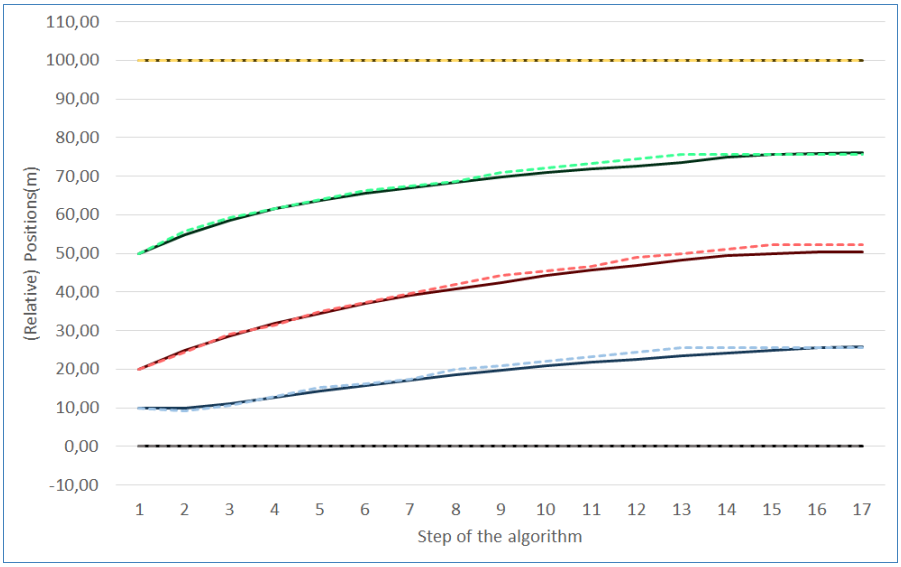
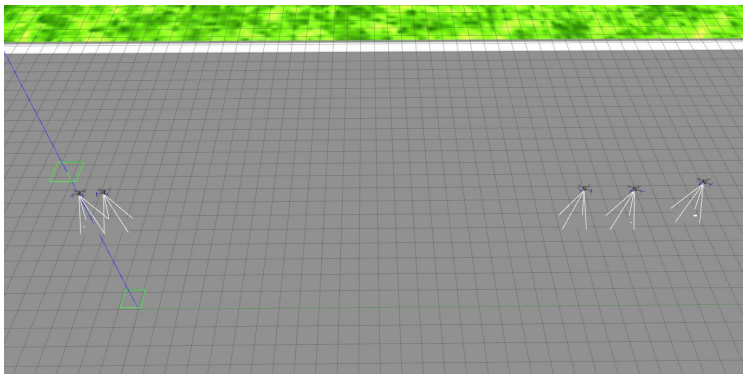


Fig. 3: Actual (dashed lines) and desired (continuous lines) positions of drones during the co-ordination script execution.

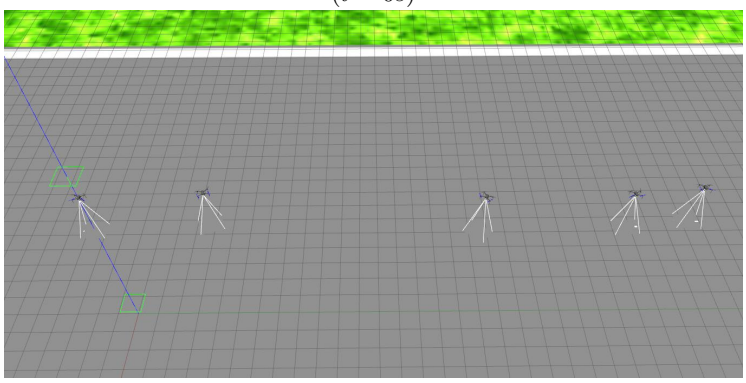
Figure 3 reports the desired positions of the drones, i.e. the positions after the computation of the modified version of the Olfati-Saber et al. algorithm, together with the actual positions of UAVs, during the execution of the simulation. Step #1, Step #9 and Step #17 of the co-ordination algorithm refer to simulation at t=0, t=28 and t=51, respectively.

Another simulation scenario is reported in Figure 4, where the initial position of mobile drones is at the boundaries of the interval. In the figure, we show again the dynamic behavior of drones and we consider three different phases during the execution of the coordination algorithm.

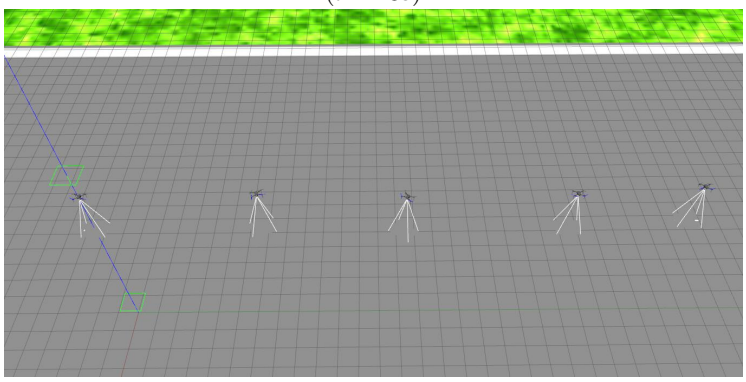
The results of the simulations show the convergence of the co-ordination algorithm, as expected.



($t = 0s$)



($t = 28s$)



($t = 51s$)

Fig. 4: Dynamic behavior of a team of cooperative UAVs: a) initial deployment at $t = 0$ seconds, b) intermediate displacement at $t = 28$ seconds, and c) final deployment at $t = 51$ seconds.

6 Discussion and Conclusions

This work extends the ROS/Gazebo architecture with the possibility of simulation of co-operative UAVs, adhering to their own individual dynamics as well as interacting with each other according to shared cooperation strategies. Two important features of the framework are its flexibility and modularity. Indeed, heterogeneous vehicles and new co-ordination algorithms can be easily plugged into the simulation, by simply modifying the corresponding Python script. In the paper, preliminary results on the proposed formation control scheme that combine actual positions and desired positions of the vehicles are presented. A limitation of the framework is that launching and configuring a multi-UAVs simulation requires some manual operations. Scalability poses another major issue mostly attributable to the physical-level simulation performed within Gazebo. Finally, although the use of trigonometric functions for distance measurements (Haversine formula) involves small approximation errors, simulations show that the computations have maintained near the theoretical expectations. The automated creation of the UAV instances and the application of the framework to more complex scenarios are objects of further work. Moreover, co-simulation techniques [4, 9], in which different sub-systems, possibly modelled and simulated with different tools, are co-ordinated by a co-simulation engine will be investigated.

References

1. 3DRobotics: Dronekit-pythons documentation (2016), `http://python.dronekit.io/`
2. Adams, S.M., Friedland, C.J.: A survey of unmanned aerial vehicle (UAV) usage for imagery collection in disaster research and management. In: 9th International Workshop on Remote Sensing for Disaster Response. p. 8 (2011)
3. ArduPilot-DevTeam: Ardupilot documentation (2016), `http://ardupilot.org/ardupilot/`
4. Bernardeschi, C., Domenici, A., Masci, P.: A PVS-Simulink Integrated Environment for Model-Based Analysis of Cyber-Physical Systems. IEEE Trans. Software Eng. **44**(6), 512–533 (2018)
5. Chandler, P.R., Pachter, M., Swaroop, D., Fowler, J.M., Howlett, J.K., Rasmussen, S., Schumacher, C., Nygard, K.: Complexity in uav cooperative control. In: Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301). vol. 3, pp. 1831–1836. IEEE (2002)
6. Dronecode-Project: Mavlink developer guide (2018), `https://mavlink.io/en/`
7. Ham, Y., Han, K.K., Lin, J.J., Golparvar-Fard, M.: Visual monitoring of civil infrastructure systems via camera-equipped unmanned aerial vehicles (uavs): a review of related works. Visualization in Engineering **4**(1), 1 (2016)
8. Koenig, N.P., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: IROS. vol. 4, pp. 2149–2154. Citeseer (2004)
9. Larsen, P.G., Fitzgerald, J., Woodcock, J., Fritzson, P., Brauer, J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., Sadovykh, A.: Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project.

In: 2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data). pp. 1–6 (April 2016)
10. Lu, P., Geng, Q.: Real-time simulation system for uav based on matlab/simulink. In: Computing, Control and Industrial Engineering (CCIE), 2011 IEEE 2nd International Conference on. vol. 1, pp. 399–404. IEEE (2011)
11. Maza, I., Caballero, F., Capitán, J., Martínez-de Dios, J.R., Ollero, A.: Experimental results in multi-uav coordination for disaster management and civil security applications. Journal of intelligent & robotic systems **61**(1-4), 563–585 (2011)
12. Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., von Stryk, O.: Comprehensive simulation of quadrotor uavs using ros and gazebo. In: Noda, I., Ando, N., Brugali, D., Kuffner, J.J. (eds.) Simulation, Modeling, and Programming for Autonomous Robots. pp. 400–411. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
13. Olfati-Saber, R., Fax, J.A., Murray, R.M.: Consensus and cooperation in networked multi-agent systems. Proceedings of the IEEE **95**(1), 215–233 (Jan 2007)
14. Ollero, A., Marron, P.J., Bernard, M., Lepley, J., la Civita, M., de Andres, E., van Hoesel, L.: Aware: Platform for autonomous self-deploying and operation of wireless sensor-actuator networks cooperating with unmanned aerial vehicles. In: Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on. pp. 1–6. IEEE (2007)
15. OSRF: Gazebo api reference (2017), `http://osrf-distributions.s3.amazonaws.com/gazebo/api/8.2.0/index.html`
16. OSRF: Ros wiki: Documentation (2018), `http://wiki.ros.org/`
17. PX4-DevTeam: Pixhawk series (2018), `https://docs.px4.io/en/flight\_controller/`
18. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA workshop on open source software. vol. 3, p. 5. Kobe, Japan (2009)
19. Quintero, S.A., Papi, F., Klein, D.J., Chisci, L., Hespanha, J.P.: Optimal uav coordination for target tracking using dynamic programming. In: Decision and Control (CDC), 2010 49th IEEE Conference on. pp. 4541–4546. IEEE (2010)
20. Rasmussen, S.J., Chandler, P.R.: Multiuav: A multiple uav simulation for investigation of cooperative control. In: Simulation Conference, 2002. Proceedings of the Winter. vol. 1, pp. 869–877. IEEE (2002)
21. Remondino, F., Barazzetti, L., Nex, F., Scaioni, M., Sarazzi, D.: Uav photogrammetry for mapping and 3d modeling–current status and future perspectives. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences **38**(1), C22 (2011)
22. Robusto, C.C.: The cosine-haversine formula. The American Mathematical Monthly **64**(1), 38–40 (1957)
23. Rysdyk, R.: Unmanned aerial vehicle path following for target observation in wind. Journal of guidance, control, and dynamics **29**(5), 1092–1100 (2006)
24. Semsch, E., Jakob, M., Pavlicek, D., Pechoucek, M.: Autonomous uav surveillance in complex urban environments. In: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 02. pp. 82–85. IEEE Computer Society (2009)
25. Techy, L., Woolsey, C.A., Schmale, D.G.: Path planning for efficient uav coordination in aerobiological sampling missions. In: Decision and Control, 2008. CDC 2008. 47th IEEE Conference on. pp. 2814–2819. IEEE (2008)

26. Tortonesi, M., Stefanelli, C., Benvegnu, E., Ford, K., Suri, N., Linderman, M.: Multiple-uav coordination and communications in tactical edge networks. IEEE Communications Magazine **50**(10), 48–55 (2012)
27. Wise, R., Rysdyk, R.: Uav coordination for autonomous target tracking. In: AIAA Guidance, Navigation, and Control Conference and Exhibit. p. 6453 (2006)
28. Yoonseok Pyo, Hancheol Cho, L.J.D.L.: ROS Robot Programming (English). ROBOTIS (12 2017)
29. Zhang, C., Kovacs, J.M.: The application of small unmanned aerial systems for precision agriculture: a review. Precision agriculture **13**(6), 693–712 (2012)