Politecnico di Torino

UAVs SIMULATOR

USER MANUAL

Authors:

Andrea Cuzzi 317868
Loredana Logruosso 301158
Luca Nepote 315234

Supervisors:

Paolo Giaccone
Francesco Raviglione

Contents

1 Introduction

Welcome to the user manual for the UAV Simulator "DronetSim"! This brief guide is designed to familiarize you with the downloading, key features, functionalities, and operation of our Unmanned Aerial Vehicle (UAV) simulator. Whether you are a passionate UAV pilot or a company looking for the simulation and analysis of your UAV performance, this manual will serve as your resource!

The UAV Simulator offers a realistic and immersive virtual environment that tries to replicate real-world scenarios, enabling users to practice and train in a safe and controlled setting. You can analyze your systems under different environmental conditions, and make the UAVs collaborate with each other in order to overcome the presence of communication obstacles, that could bring some UAVs BLOS ("Beyond Line Of Sight").

In this manual, you will find step-by-step instructions on how to download and set up the simulator on your system, as well as a comprehensive overview of the user interface, controls, and output. We will guide you through the process of configuring the environment parameters, such as the temperature and the wind, and customizing your simulation sessions to satisfy your specific needs.

Whether you are using the UAV Simulator for recreational purposes or research activity, this user manual will serve as your indispensable guide. The simulator is Open Source, released under GPLv2 license on GitHub, so you can modify it adapting to your needs!

Get ready to fly with us!

2 Download of the Simulator

To get access to our simulator, you can simply run the following code from a terminal, into the desired directory.

```
git clone https://github.com/LoreLog/DronetSim.git
```

3 Flags

In this section, we explain the settings for the "flags" of the simulator. These parameters must be modified directly in the *main.py* file, at lines $65 - 71$, and they are:

- **SOUND**: when it is set to *true*, the sounds for the simulation are enabled, otherwise they are disabled;
- **FLAG_debug**: when it is set to *true* the simulation shows the profiles of the obstacles affecting the communication. It is useful when someone wants to debug the system;
- **DEFAULT_PARAMETERS**: when this flag is set to *true*, the default parameters are instantiated and the GUI does not show up. In this case, the parameters are taken directly by the code;
- **FLAG_OS**: with this we can choose the Operative System in which we can start the simulation. The two available options are '$Windows$' and 'MAC'. This affects the saving of the output graphs.
  The 'MAC' setup must be chosen also for Ubuntu OS.

4 GUI

If the flag **DEFAULT_PARAMETERS** is set to *false*, a graphic user interface (GUI), as represented below, will pop up at the beginning of the simulation.
With this, an external user is able to set the correct parameters for a specific simulation, according to his needs.

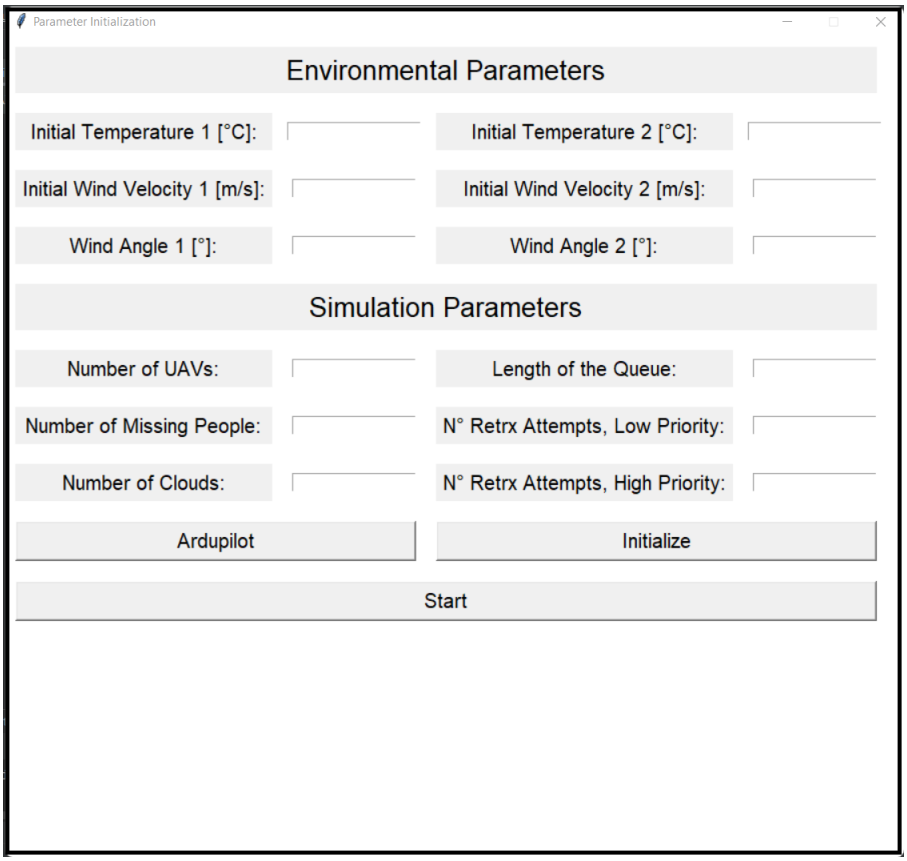


Figure 1: GUI

Now we clarify the instantiation of each of these parameters: the map has been divided into two parts, a left side in which the UAVs are able to communicate directly with the ground station and a right side, for which the presence of the mountain at the center interrupts the direct link. We assumed that being on a high mountain, the environmental conditions are constant on each side, but they are different from each other: considering this scenario, the external user can set up the values of temperature, the modulus of the wind velocity, and the corresponding angle. The variables "Initial Temperature 1", "Initial Wind Velocity 1" and "Wind Angle 1" are referred to the left side, whereas the ones with the number 2 correspond to the right side. Notice that for each parameter the corresponding unit of measurement has been specified.
There are also other three available settings, called "Number of UAVs", which specifies the number of UAVs used for the simulation (except for the UAV used as a relay for the communication on the top of the mountain), "Number of Missing People", useful when we have multiple people to look for, and "Number of Clouds", in which we can set the initial number of clouds.

Notice that not all the values can be given to the GUI, indeed each parameter is limited according to the mission planning or by the UAV specification [4]. The available ranges are:

- Temperature: $[-20; 40]$°C. Exceeding this range, the UAV is not able to fly due to the battery specification;
- Wind Velocity: $[0; 10]$m/s, according to the specifications;

- Wind Angle: $[0; 360]°$;
- Number of UAVs must be a number > 0. There is no upper limit, so any positive value can be used;
- Number of Missing People can't assume negative values;
- Number of Clouds must be a positive value, or at least equal to zero.

After all the parameters have been set up, the user has to click on the "Initialize" button, and a recap of all the values will pop up if they are all correct, otherwise an error message with the specified problem will occur.

The user has also the possibility to use Ardupilot for the mission planning: to do that he has to click on the "Ardupilot" button and make sure that the value is set to $True$.

In the end, when the user is ready, the "Start" button must be clicked and the simulation will start immediately.

5 Ardupilot

Ardupilot [1] is an open-source project that needs to control unmanned vehicles. Coupled with ground control software, unmanned vehicles running ArduPilot can have advanced functionality including real-time communication with operators.

It is consisting of three components:

- **Hardware**: peripheral sensors, controller.
  Using inputs from sensors, the controller is able to send outputs;
- **Firmware**: the code running on the controller;
- **Software**: interface to the controller. (Ground Control Station)

5.1 Download

With the aim of coupling our simulator with ArduPilot, the user needs to download the Firmware, the Ground Control Station (GCS), and the PyMavlink library. The solution coupling ArduPilot with our simulator requires Linux as Operating System (e.g., Ubuntu) since we choose a GCS that runs only on Linux.

Firmware

To get ready to utilize Ardupilot, it is first needed to install the ArduPilot Code through GitHub. To do that you can follow this guide [2].

GCS

As GCS we chose **QGroundControl** [6] because it is easy to exchange messages through the MAVLink protocol. The installation guide can be found here [5].

PyMavLink

Pymavlink is a Python implementation of the MAVLink protocol. With Pymavlink, it is possible to create a Python script to read sensor data and send commands to an ArduSub vehicle. Follow this guide for the installation of the library and for its comprehension [3].

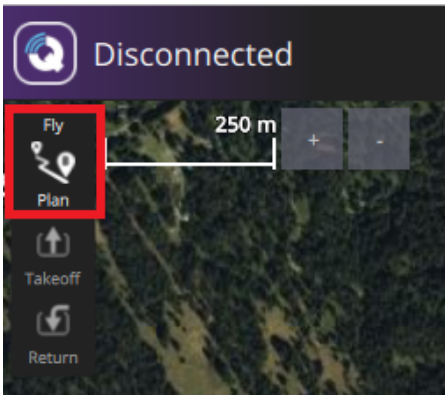
5.2 Mission Planning: From Ardupilot to DRONET Simulator

Let us implement the functionality of Mission Planning from QGroundControl on Ubuntu. It is first needed to move inside the directory in which it is installed and run it. This can be done through the following commands, by using the terminal.

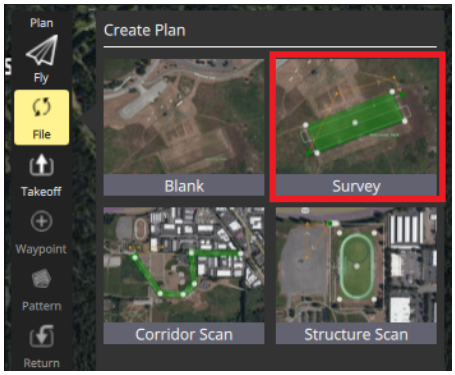
```
# Moving into the desired directory
cd Scrivania
./QGroundControl.AppImage
```

Once the QGroundControl application runs, proceed as follows:

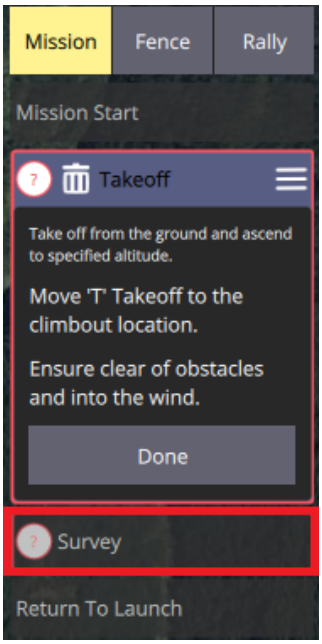
1- select Fly Plan

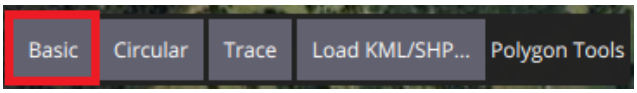


2- select Survey

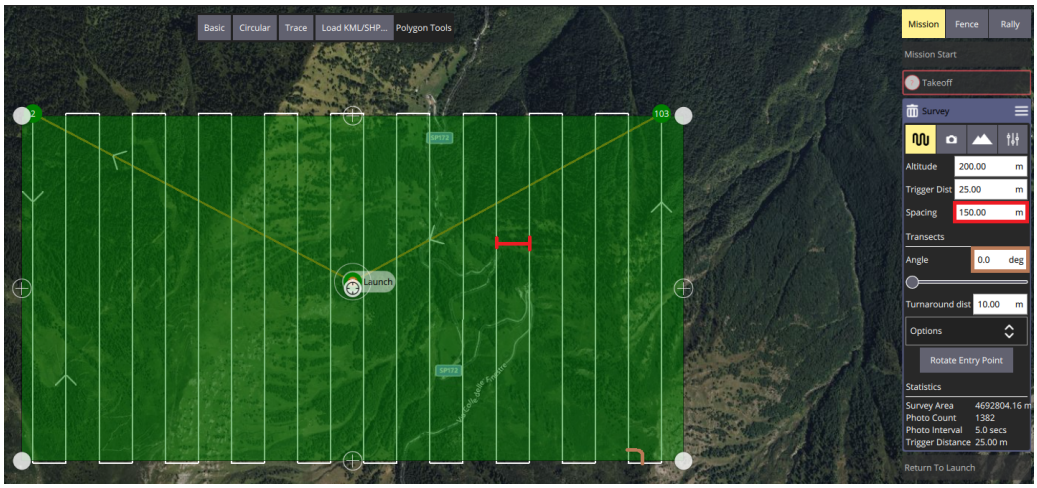


3- Select Survey & Basic

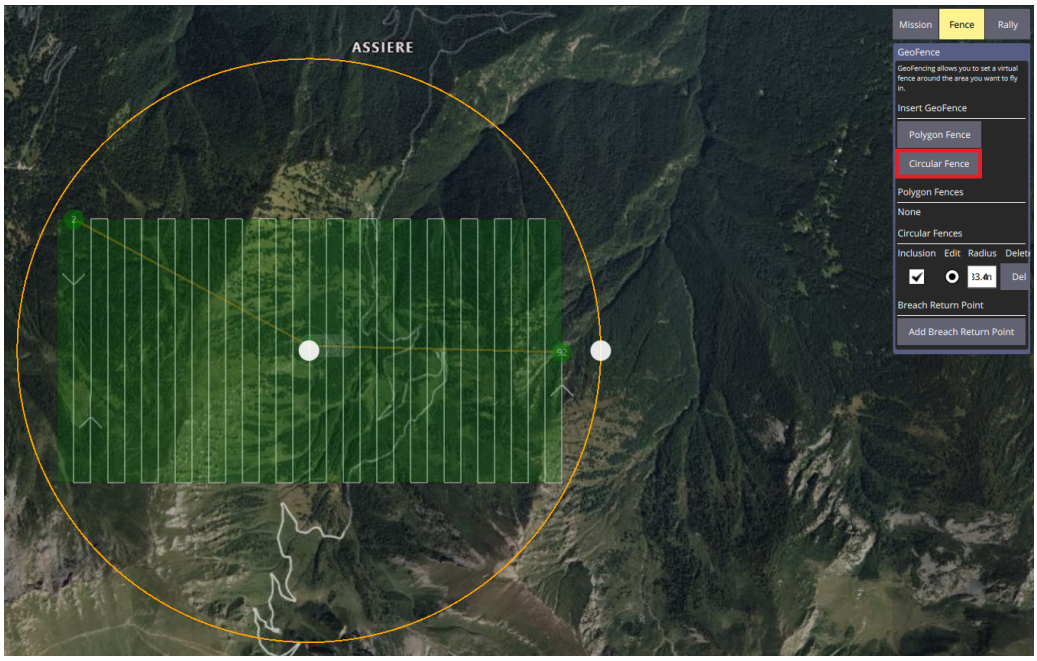




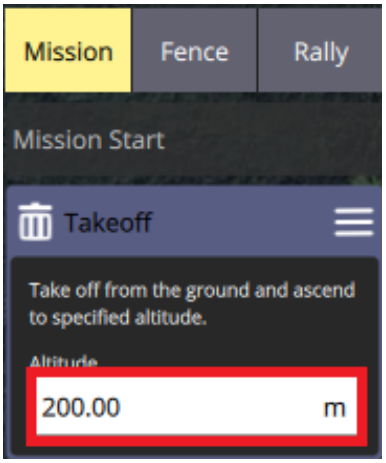
4- Select the Spacing and rotation



5- Set the fence



6- Select Takeoff altitude



7- Save in a JSON file

Make sure to save the JSON file in the directory "Ardupilot_Mission_Planning" inside the simulator. Then run the following Python script from the last directory.

```
python3 convert.py ./prova2.plan
python3 reading_Ardupilot_Waypoints.py
```

At this point lunch the Simulator and select the ArduPilot button. In this way, only one UAV is considered. Once the Waypoint CSV file is created, the mission planning can be utilized also in Windows OS.

5.3 Other Functionality

Via the PyMavLink library, it is possible to make a Python program communicate with QGroundControl. There is a need to initialize communication between the two processes via a socket, e.g. 'localhost:14550', and then send a message from the Python program to the QGC. This functionality can be exploited to make a Digital Twin of our simulator. However, the appropriate parameters of the UAV corresponding to those of the simulator must be set.

We were able to realize an offline version: we first run the simulation and then use the waypoints from the simulation to simulate the UAV's path in the QGC. In order to realize an online version (real-time communication) that corresponds to our simulator, however, motion-related parameters such as speed and battery-related ones would have to be passed.

To use our limited version, just follow these commands

```
# navigate into the dedicated folder
cd Scrivania/ardupilot/ArduCopeter
# run the UAV in writing mode, the first time
python3 sim_vehicle -w
# stop the simulation once done
Ctrl C
# restart
python3 sim_vehicle -v ArduCopter -L ColleDelleFinestre --console
mode guided
# into another terminal run the QGroundControl
cd Scrivania
./QGroundControl.AppImage

# finally start our python program, after waiting for the QGC to be ready
cd Project-Software-defined-communication-system-DRONET/
cd "Ardupilot - sim to QGC"
python3 ArduPilot.py
```

6 Simulator Output

The simulator returns several output plots related to the UAVs themself, their movement, and communication.
If the simulation is related to a maximum number of 3 UAVs, the plots represent the behavior of each one, whereas, if there are more vehicles, it returns the minimum, the maximum, and the mean behaviors, showing also the confidence interval for the UAV speed, the battery level, and the bitrate. In addition to these, other plots are produced, representing the behaviors of the first and the last UAVs to run out of energy.

6.1 Testing Plots Explanation

The first output plot is related to the battery level over time: it is possible to see how different temperatures affect differently the power consumption.

The second important outcome represents the bitrate, in which we can see the effects of the obstacles in the communication range, and of the distance from the Ground Station on data transmission.
The next output plots represent the distance for the UAVs with respect to the ground station, the x and y positions, and the waypoint distances.
There are also other plots representing the changing environmental conditions during the simulation. There is also a plot that shows the time autonomy of each drone vs its average temperature during the simulation. The one with the highest average temperature should be the one with the highest autonomy according to our battery model.
And finally, there is the plot for the UAV speed during the simulation: its maximum value should be lower than the UAV speed plus the maximum wind supported by it when their speeds have the same direction.

6.2 Communication Result Explanation

The analysis focused on communication and involved six simulations with different seeds. The key metrics examined are throughput, bit loss and latency, with separate evaluations for relay and normal UAVs.
Throughput is calculated by summing all received bits at the Ground Station over the simulation period.
Bit loss has been quantified considering the total number of lost bits, and all the lost packets throughout the simulation.
Latency was measured as the time elapsed from packet generation to reception. Specifically, for the relay, it included the time taken from packet generation within the hidden UAV, through its queueing in the relay, transmission by the relay, and finally, reception at the Ground Station.

In the simulation, each communication event saves its data in a CSV file. At the end of the simulation, all the metrics varying over time are available. For generating the graphs presented in the report, only the final values of these metrics were considered. These values were imported into MATLAB, where the metrics from each experiment with different seeds were stored. Finally, the confidence interval was calculated for the collected values, using a confidence level of 70%. It's important to note that with a limited number of simulations, the width of the confidence interval may not be very narrow.
The three MATLAB scripts used to calculate these metrics are named *throughput.m*, *bitloss.m*, and *latency.m*.

References

[1] Ardupilot. Ardupilot documentation. https://ardupilot.org/ardupilot/index.html.

[2] ArduPilot. Setting up the build environment (linux/ubuntu). https://ardupilot.org/dev/docs/building-setup-linux.html.

[3] ardusub.com. Pymavlink gitbook. https://www.ardusub.com/developers/pymavlink.html.

[4] dji ENTERPRISE. Uav specifications. https://enterprise.dji.com/it/matrice-300/specs.

[5] DroneCode. Download and install. https://docs.qgroundcontrol.com/master/en/getting_started/download_and_install.html.

[6] DroneCode. Qgroundcontrol user guide. https://docs.qgroundcontrol.com/master/en/index.html.