CASCADE Report: Simulation of Unmanned Air Vehicle Systems DRAFT

May 8, 2020

1 Introduction

> Different models can live in peace with each other, without conflict, as long as they serve different purposes. [5]

This report describes a collection of different approaches, tools and methods for numerical simulation of the flight of Unmanned Air Vehicles (UAVs). As expected, the right tool to use varies greatly dependent on the particular purpose of the simulation, so this report covers a selection of methods rather than a single one-size-fits-all solution. It is not an exhaustive survey, but instead captures the collected experience of the authors.

1.1 Example Software

Software for the simulation case studies in this report can be downloaded from:
`https://github.com/arthurrichards77/cascade_sim_tools`.

1.2 CASCADE Simulation Architecture

Figure 1 shows a typical architecture for a UAV control system. Starting in the air segment, the top right covers the interactions between the flight platform and the environment, including aerodynamic effects, weather, collisions, etc. In the inner control loop, the autopilot uses sensor information about the environment (e.g. GPS, inertial measurements) to stabilise the platform by commanding the actuators. The effects of the actuators on the platform will differ depending on the nature of the platform, *e.g.* for rotorcraft, commanding speed changes of the various motors will have different effects depending on their layout. The effects is shown separate from the dynamics of the platform as they are often separable in the model, especially for multi-copters.

While the autopilot is itself a computer, it is typically small, embedded, and dedicated to the sole task of stabilization. We therefore refer to any additional on-board computer as a 'companion' computer, which may undertake various automation or assistive tasks from simple data logging or communications relay to adnavced path-planning or vision processing. The autopilot gets its instructions, e.g. demand signals for control loops, from either the companion computer on-board, directly from a human via the ground station, or from a companion computer on the ground via the ground station. Again, the ground station and its companion are shown separated here as they reflect different functions, with the ground station typically devoted to human-machine interface and communications marshalling. However, in practice, ground station software and companion software may share a common physical computer. The final control interface is the supervisory control link from the human pilot (who may or may not also be the person operating the ground station). This latter link typically offers a simple, direct control of the autopilot demands, e.g. velocities, using radio-control protocols such as pulse-width modulation (PWM) driven by joystick input. Finally, the visual interface from the humans to the platform, i.e. the fact that they observe its motion, is recorded, as a key aspect of some simulators is the provision of visual rendering of the scene for training or observation.

1.3 Brief Survey of Simulation Tools used by CASCADE Partners

Table 1 compares the simulation tools used most within the CASCADE team. Each tool is categorized in the following ways:

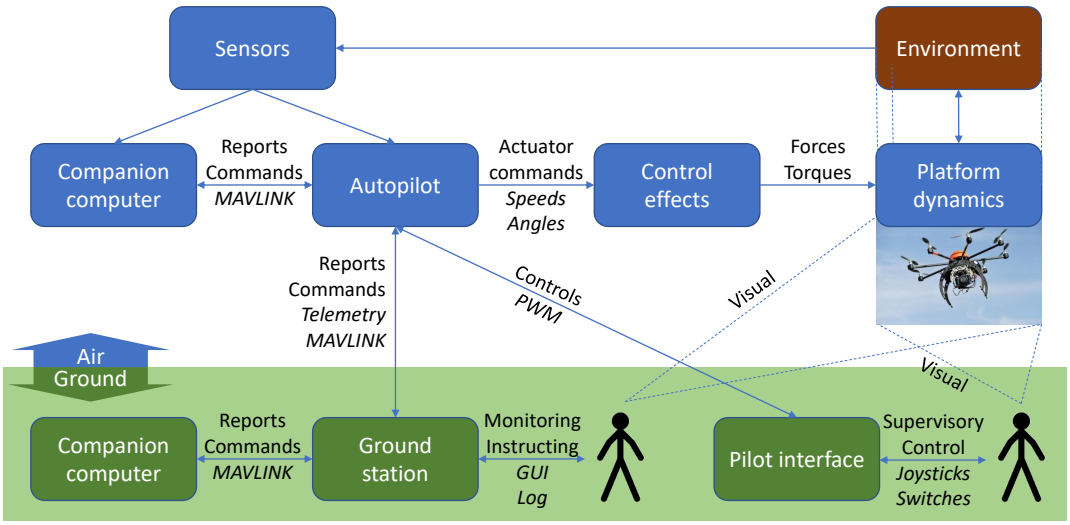


Figure 1: Typical UAV control system architecture

- **Platform types**: The types of flying vehicles that can be modeled. 'Generic' means that any type of vehicle can be modeled in principle and it is up to the user to provide an appropriate mathematical model for each.
- **Operating systems**: Self-explanatory
- **Timewarp**: Here, 'Real' implies that the simulation runs in real time, *i.e.* one simulated second every one wall clock second. 'Fast' means many simulated seconds (ideally) in one wall clock second. Fast-time simulation is preferred for design evaluation, but is inappropriate if the human interfaces in Figure 1 are to be modeled, *e.g.* for training purposes orhuman-in-the-loop trials.
- **License**: Self-explanatory
- **Fidelity** Divided into three categories:
  - Physical: does the model accurately represent the behaviour that would be shown by a real physical system given the same inputs? This corresponds to good handling of the 'forces and torques' interface in Fig. 1 and possibly the 'Actuator commands' interface. Physical fidelity is especially vital for autopilot control system development.
  - Systems: does the model accurately represent the systems present in a real flight? This corresponds to the 'Reports' and 'Commands' interfaces in Fig. 1. Systems fidelity is required for verification of subsystems prior to real flight, *e.g.* software running on a companion computer or assisting the ground station.
  - Visual: does the model accurately portray a visual representation of the flying vehicle(s) and the environment? This corresponds to the visual interfaces shown in Fig. 1. Visual fidelity is important for pilot training, wider stakeholder engagement, and simply offering an intuitive view of how a flight will work.

Note that all three 'axes' of fidelity are important and no single tool does well on all of them. This is perhaps inevitable given the conflicting requirements of each of the three axes. For this reason, this report does not recommend one single tool for UAV simulation work. Instead, a suite of tools has been identified to span the space of requirements. Efficiency of working across tools may be helped by the development of standard workflows, *e.g.* ways of working with these tools that eases transitions between them. This is left for futher development.

Name	**Platform types**	**Operating systems supported**
Simulink	Generic	Windows; Linux; Mac
Timewarp	**License**	**Fidelity**
Fast	Commercial	Physical: High; Systems: Low; Visual: Low
Description: Simulink is a graphical tool for mathematical modeling of dynamic systems. Given an appropriate model (see Section 2.1) it can simulate the motion of any UAV, including its control system. Although it has no native support for MAVlink or graphics, toolboxes can be used to interface it to FlightGear for visualization or to a MAVlink system. It has been used extensively across the CASCADE team for control system development. **Link**: `https://uk.mathworks.com/products/simulink.html`		
Name	**Platform types**	**Operating systems supported**
Ardupilot SITL	Multirotor; fixed-wing	Windows (via Cygwin); Linux
Timewarp	**License**	**Fidelity**
Real (typically)	Open source	Physical: Medium; Systems: High; Visual: Low
Description: Provided as part of the ardupilot open source autopilot software stack, this tool enables simulation of a generic platform with the real ardupilot firmware in the loop, offering great fidelity of the interfaces used for real flight, especially for MAVlink companion computers. Its physics model is simple although the plane simulator uses the same trusted JSBsim tool as found in FlightGear. No visualization is provided but it can be connected to FlightGear for that. It has been used in CASCADE to develop autonomous ground assistance software for the BRIDGE case study, among other things. It can be run directly from Mission Planner. (Tip: if you get it crashing, delete the 'sitl' folder in your 'Mission Planner' folder. To compile, download the source and follow the 'readme' instructions, as the online guide is out of date.) **Link**: `http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html`		
Name	**Platform types**	**Operating systems supported**
FlightGear	Large fixed-wing, some rotorcraft	Linux; Windows
Timewarp	**License**	**Fidelity**
Real	Open source	Physical: Medium; Systems: Medium; Visual: High
Description: Supported by a large (serious?) gaming community, this simulator focuses on providing a pilot-like experience of flight, so although good simulation of flight physics is possible, it is more focused on scenery and aircraft rendering. It does have a very good API for customization and real-time interfacing, plus flexible provision for autopilot customization. It also simulates aviation infrastructure such as VORs. It's been used for autonomy and air traffic research plus the production of some extremely pretty videos. **Link**: `https://www.flightgear.org/`		

Table 1: Comparison of Drone Simulation Software - Tools covered in this Report

Name	**Platform types**	**Operating systems supported**
Airsim	Multirotor; car	Windows; Linux
Timewarp	**License**	**Fidelity**
Real	Open source	Physical: Medium?; Systems: Medium; Visual: High
Description: This research tool from Microsoft targets AI development with lovely visuals and sensor emulation. It runs on gaming engines Unreal or Unity. Unreal was easy to start with but less well supported generally than Unity. However, Airsim for Unity is still in development and requires much background expertise to get working. Taranis and Xbox 360 controllers work directly, and DJI controllers can be made to work but only with impractical supporting software. Google map integration is in the works, but may not be freely available. **Link**: `https://github.com/Microsoft/AirSim`		
Name	**Platform types**	**Operating systems supported**
DJI Flight Simulator	DJI multirotors	Windows
Timewarp	**License**	**Fidelity**
Real	Free (with DJI account)	Physical: Medium; Systems: High; Visual: Good
Description: Essentially a trainer for human-controlled flight of DJI products, this offers good visuals. DJI flight controllers plug in to act as joystick. There is no interface for programmatic control *e.g.* a companion computer. **Link**: `https://www.dji.com/uk/simulator`		
Name	**Platform types**	**Operating systems supported**
Gazebo	Multirotors... just	Linux
Timewarp	**License**	**Fidelity**
Real (optional speed-up)	Open source	Physical: Medium; Systems: Medium; Visual: Medium
Description: Gazebo is now part of the ROS ecosystem, mainly aimed at rigid body mechanics such as rover or arm simulation, for which it has a good physics engine. It also has good sensor simulation capabilities, e.g. virtual cameras detecting AR tags etc. Some quadrotor simulation models have been developed for it, *e.g.*by TU Munich for the ARDrone which helpfully emulates the ROS interface for the real thing. Although useful for preparing software for indoor flight, Gazebo is intended for rigid-body mechanics and doesn't simulate fluids. Therefore drones in Gazebi tend to have sluggish response due to simple friction models. Use for control tuning at own risk. The MIMree project is using it, as it offers the possibility of simulating different robot platform types together. **Link**: `http://gazebosim.org/`, `http://wiki.ros.org/tum_simulator`		
Name	**Platform types**	**Operating systems supported**
Webots	DJI Mavic	Linux; Windows; Mac
Timewarp	**License**	**Fidelity**
Real	Commercial	Physical: Low; Systems: Medium; Visual: High
Description: Robotics simulator offering a Mavic simulation alongside lots of rovers and arms. *No experience known in CASCADE.* Visual and sensor simulation appear good, but the flight physics fidelity is unclear. **Link**: `https://cyberbotics.com/doc/guide/mavic-2-pro`		
Name	**Platform types**	**Operating systems supported**
Morse	Generic quadrotor	Linux
Timewarp	**License**	**Fidelity**
Real	Open source	Physical: Low; Systems: Low; Visual: Medium
Description: Another free robotics simulator, with good ROS integration, used to good effect for driverless car development. *No experience known within CASCADE using this for drones.* It does offer a simple-looking drone model, but the 'friction' settings in the documentation indicate likely low fidelity of flight dynamics. **Link**: `https://www.openrobots.org/morse/doc/stable/user/robots/quadrotor_dynamic.html`		
Name	**Platform types**	**Operating systems supported**
jMAVsim	Generic quadrotor	Linux; Windows; Mac
Timewarp	**License**	**Fidelity**
Real	Open source	Physical: Medium; Systems: High; Visual: Medium
Description: Seems to be the equivalent of Arducopter's SITL for the PX4 autopilot firmware, available separately as a standalone Java application. Has the advantage of a MAVlink interface. **Link**: `https://dev.px4.io/master/en/simulation/jmavsim.html`		

Table 2: Comparison of Drone Simulation Software - Tools not covered in this Report

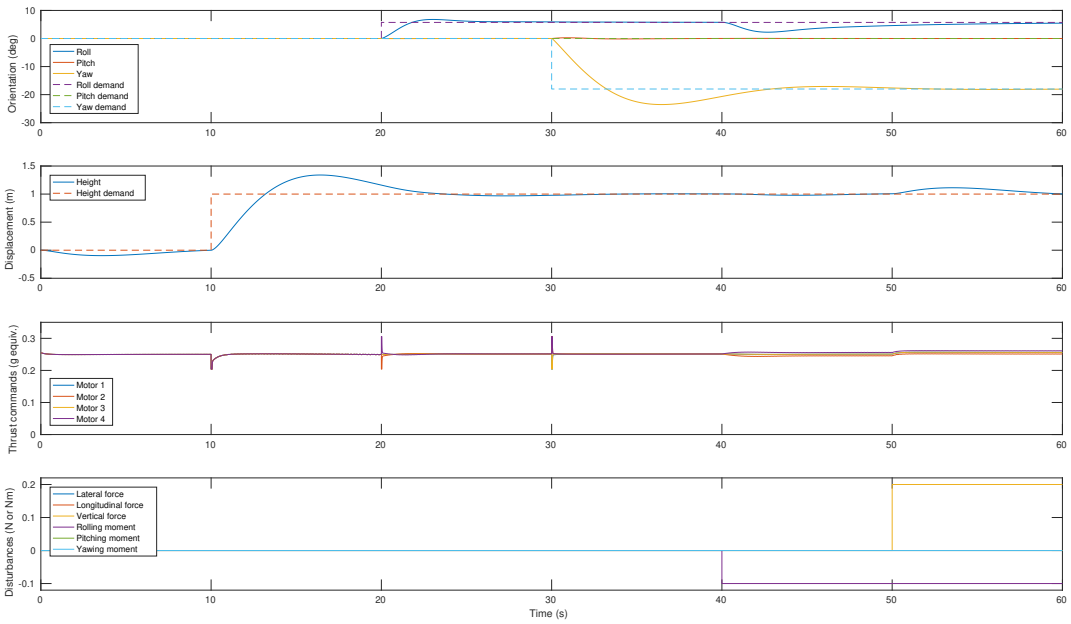


Figure 2: Example Simulink results

2 Single UAV Simulations

2.1 Multi-rotor Dynamic Simulation with Simulink

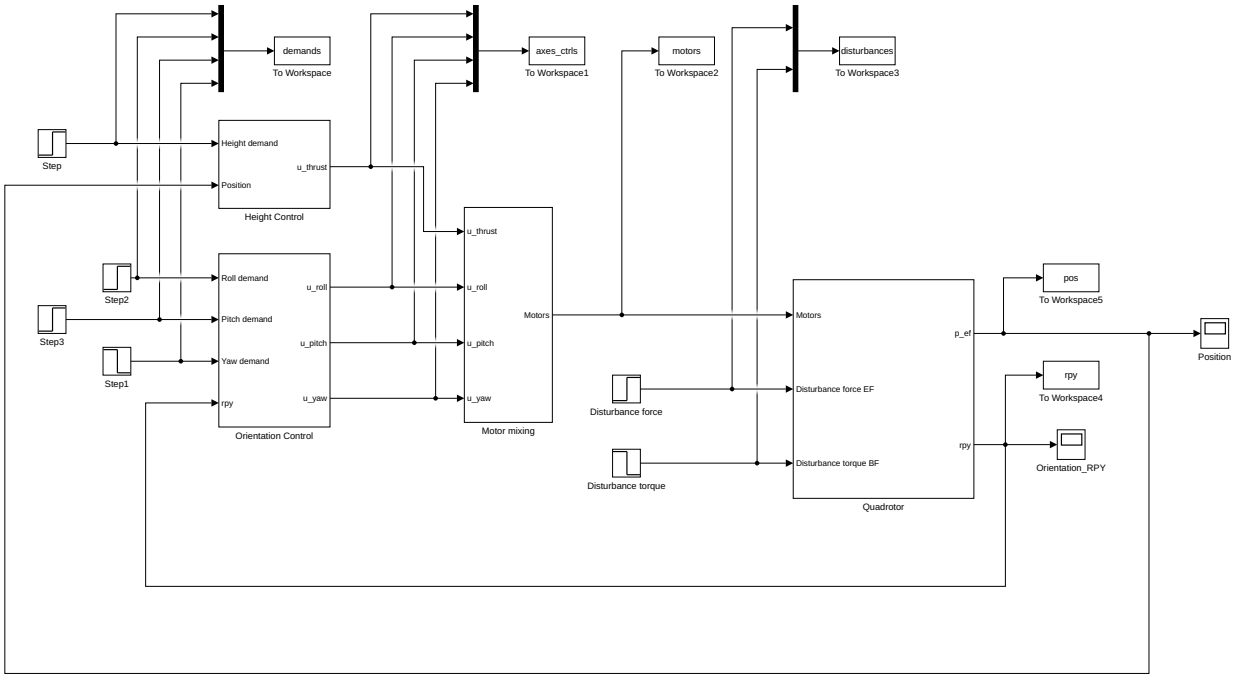
This section covers the pure dynamic simulation of a multi-rotor UAV, typically carried out in the early stages of developing a low-level control system such as that found in an autopilot. The approach is to encode the equations of motion in a pre-existing simulator for differential equations, such as Simulink from Mathworks. Suitable equations of motion can be found in different works, and typically divide into two parts: (i) the 'control allocation' term, usually just a matrix, relating the individual motor speeds to the forces and torques applied to the UAV as a whole; and (ii) the dynamics and kinematics of the UAV in response to those forces and torques. Dai *et al.* [4] discuss the different models in detail, including the control allocation matrices for a variety of configurations. The same team have undertaken a significant modeling endeavour [6] and provided an online tool with parameters for numerous popular platforms[1] including a full description of the complete dynamics model[2]. Seeking a simpler model, Cowling *et al.* [3] adapted the work of Castillo *et al.* [1] and assumed small yaw angle to arrive at a decoupled, but still nonlinear, dynamics model. Furthermore, since many multirotor operations stay close to steady hovering flight, linearization is applicable, resulting in a simple state-space or multivariable transfer function model from force/torque inputs.

Note that none of these works consider flexibility of the airframe or the rotational or electrical dynamics of the motor/rotor systems, which are typically considered to be sufficiently fast as to be ignored. Researchers interested in very fast control should therefore use these models with caution.

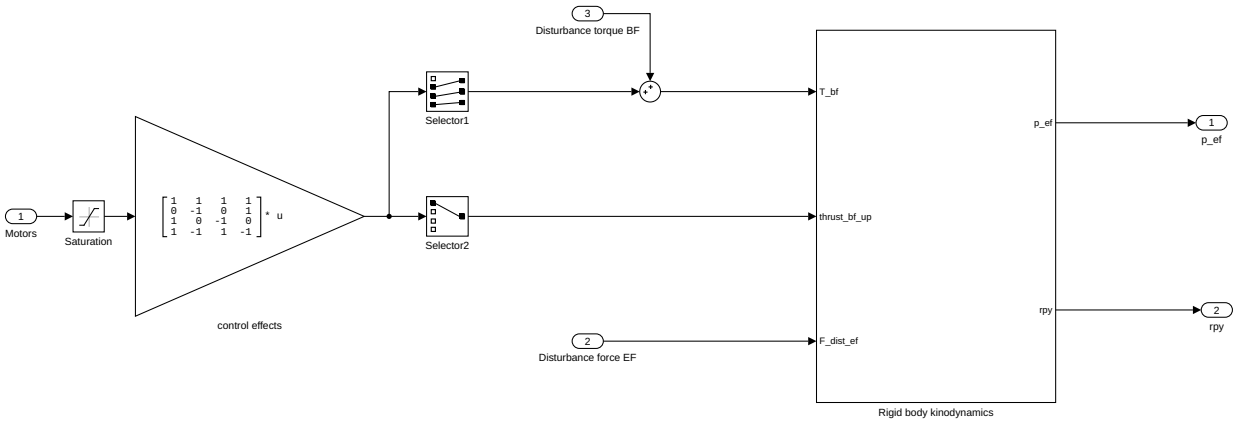
Figure 3(a) shows the topmost level of a Simulink model of a quadrotor, based on the models from Ref. [4], plus a simple control system based on four loops – pitch, roll, yaw and height – each with a PID controller. Figure 3(b) shows the contents of the 'Quadrotor' subsystem, representing the response of the platform, in which the separation of the control effects and the rigid body dynamics can be seen, representative of the 'control effects' and 'platform dynamics' elements of the architecture in Figure 1. The environment is represented only by the disturbance force and torque inputs, which may be configured to represent various environmental effects such as steady or turbulent wind. The subsystems other than 'Quadrotor' in Figure 3(a) form a partial representation of the autopilot, in this case just the control laws for a single mode. Sensors are not represented, although could be, and this type of simulation has no representation of ground control or human interaction. Figure 2 show examples of results from this simulation, in this case the closed-loop responses to changes in demands and disturbances.

[1] `https://www.flyeval.com/`

[2] `https://www.flyeval.com/js/MathModelDocEn.pdf`



(a) Top-level of model



(b) Quadrotor dynamics model subsystem

Figure 3: Simulink model of quadrotor and control system

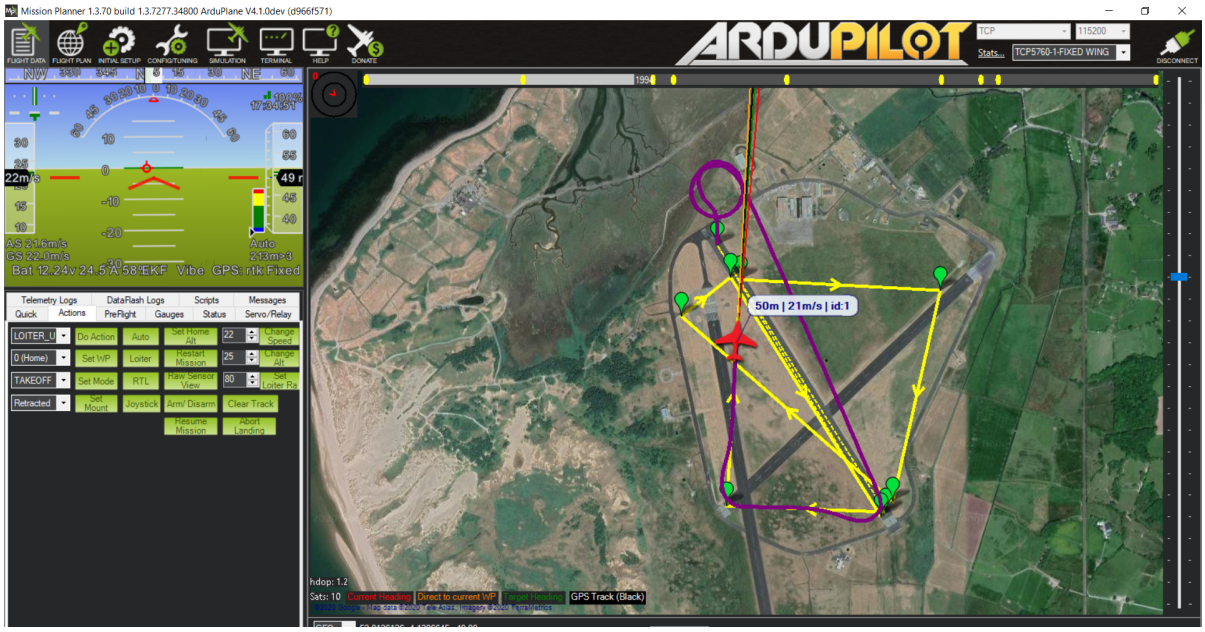


Figure 4: Screenshot of Mission Planner during SITL simulation of fixed-wing UAV at Llanbedr

2.2 Software-in-the-Loop Simulation with Ardupilot

The open-source Ardupilot supports a software-in-the-loop (SITL) simulation mode[3] in which the autopilot firmware is linked with a simple simulator to form a executable program on a desktop PC under Linux or, via Cygwin, Windows. The great advantage of this approach is that the MAVLINK interfaces are faithfully reproduced, meaning that the Ground Control Station and both companion computer elements can be tested in simulation without modifying flight code for flight. There is some support for the environment, *e.g.* adding a simulated steady wind. However, the underlying simulator seems to assume a flat earth and battery representation is variable. Limited support for sensors is provided, with workarounds for LIDAR ranging simulation. Overall this is only a loose representation of the platform dynamics, but made up for by the exact reproduction of the autopilot and its interfaces, making this the ideal environment for testing command automation.

As an example, the first command below (assuming that the software has been installed, built for SITL, and added to the path) will launch a simulation of a quadrotor at the Clifton Suspension Bridge. The `--wipe` command is recommended as good simulator practice. Otherwise, the autopilot loads its last configuration from a stored data file, which might lead to complicated interdependencies between simulation runs, *e.g.* since parameter settings are carried over rather than re-initialized. The second command (assuming the simple MAVproxy ground control software[4] has been installed) launches a ground control station instance that will communicate with the SITL simulator. Once this is achieved, additional companion computer software can connect to the SITL instance via TCP port 5762. It will be necessary to set parameters `FRAME_CLASS=1` and `ARMING_CHECK=0` before flight, as these are missing on start-up by default, and the pre-arm checks such as inertial calibration are impractical in the context of SITL.

```
arducopter --wipe --model quad --home 51.454962,-2.627718,584,270
mavproxy.py --console --map --master=tcp:127.0.0.1:5760
```

2.3 Large Aircraft Simulation in FlightGear

The open source simulator software Flightgear[5] is based on a high-fidelity flight dynamics model, although it has recently tended to focus more on graphics. Figure 5 shows a screenshot of a Grob G109 simulated aircraft[6] over a simulated Bristol in FlightGear. A custom autopilot modehas been added to give the aircraft a loitering capability, hence the circular trajectory, and a custom livery has been applied to the aircraft for promotional purposes. With

[3] `https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html`
[4] `https://ardupilot.github.io/MAVProxy/html/getting_started/index.html`
[5] `https://www.flightgear.org/`
[6] `https://github.com/FGMEMBERS/Grob-G109`

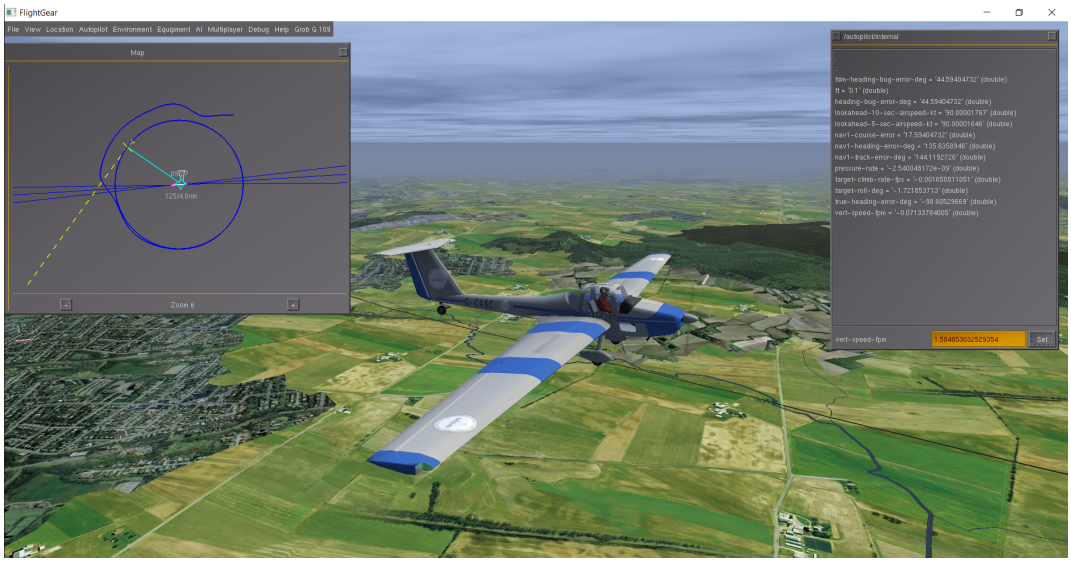


Figure 5: Screenshot of FlightGear simulator with G109 aircraft model

Aircraft	Wingspan (m)	Length (m)	MTOW (kg)	Cruise speed (knots)	Propulsion (kW)
Sagem Patroller UAV	18.0	8.5	1100	?	85
Grob G109B*	17.4	8.1	850	97	71
Hermes 450 UAV	10.5	6.1	450	70	39
Fantasy Allegro*	10.8	6.1	599	95	60
Sherwood Scout (G-OUAV)	8.7	5.8	499	80	60
B&F Fk9*	9.9	5.9	544	106	60

Table 3: Example UAV characteristics and similar aircraft available in FlightGear (denoted by *)

reference to the architecture in Figure 1, FlightGear nicely handles the dynamics and control effects, as well as the visual representations. The aerodynamic environment is reasonably well-handled and a simple terrain representation is provided. However, by design, FlightGear is best suited to large aircraft at high altitudes. At low altitudes, it becomes clear that FlightGear terrain si simply painted tiles over a coarse elevation grid, so it cannot represent small drones flying around structures. It has some representation of an autopilot including a waypoint following facility. It also supports custom autopilot development including nested loops and various control modules, such as a PID. It can also support companion computer work as all internal workings are exposed through a property get/set interface, accessible via a web or socket API. However, it is a proprietary standard, so cannot be used directly in preparation for MAVLINK autopilot deployment.

FlightGear is primarily intended as an interactive simulator. To avoid the need for manual take-off, extensive start-up customization is needed to position the aircraft in motion, in the air, and with the autopilot engaged. The startup command below launches FlightGear with the G109 aircraft 3,000ft over the Clifton Suspension Bridge and with its autopilot engaged.

```
fgfs --fg-aircraft=/home/aeagr/flightgear/aircraft --aircraft=g109
--lat=51.454962 --lon=-2.627718 --heading=270 --altitude=3000 --vc=90
--prop:/autopilot/locks/speed='speed-with-throttle' --prop:/autopilot/settings/target-speed-kt='90'
--prop:/autopilot/locks/altitude='altitude-hold' --prop:/autopilot/settings/target-altitude-ft='3000'
--prop:/controls/engines/engine/magnetos='3'
```

The Grob G109 aircraft model seen in Figure 5 is used as a UAV surrogate for its relatively slow flight, low power, and straight wings with high-aspect ratio. FlightGear does support custom aircraft model development, with everything from the aerodynamics to the lighting systems available for modification through a well-documented modeling framework based on XML files. However, the learning curve is steep. As a pragmatic approach, it is easier to try and match flight characteristics of the UAV in question with an existing FlightGear aircraft model. Table 3 shows the characteristics of some UAVs and a comparable aircraft model available in FlightGear for each.

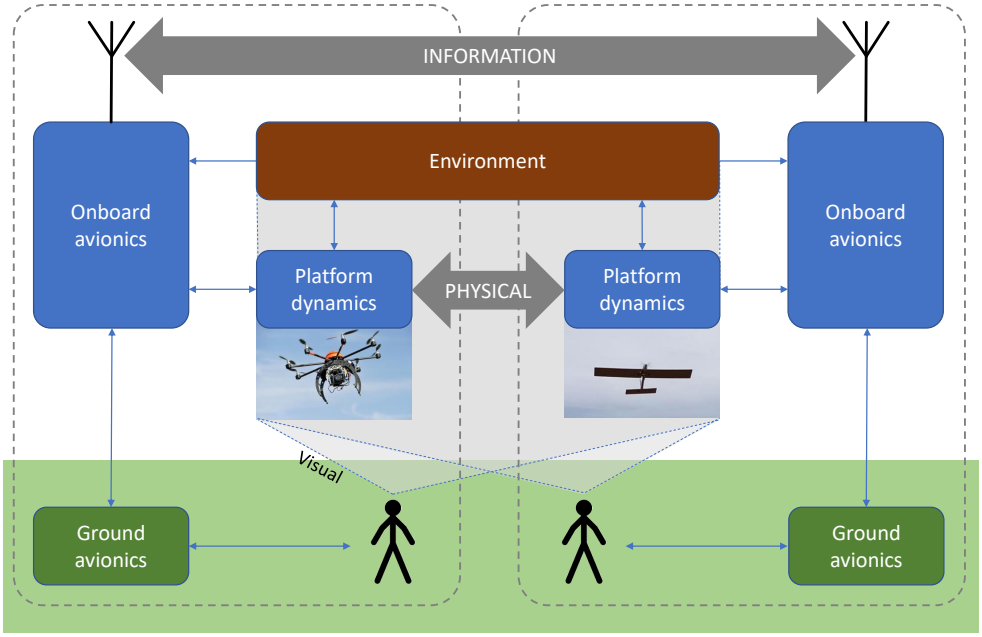


Figure 6: Extension of Simulation Architecture to Multi-agent (*e.g.* $n = 2$) case

3 Multi-UAV Simulations

Multi-vehicle simulation, especially for the purposes of traffic management research, is left for Phase 2 of CASCADE through project SEEDPOD. This section will introduce the topic, discuss considerations, and describe briefly some off-the-shelf solutions.

Figure 6 illustrates an extension of the architecture in Figure 1, in simplified form, to a multi-vehicle case, in this case the smallest meaningful case of $n = 2$. It highlights the three forms of interaction between agents in a multi-UAV simulation:

- Shared environment: vehicles see the same surroundings and aure subject to the same disturbances (*e.g.* correlation of navigation errors)
- Physical interaction: vehicles can sense each other and possibly influence each other (*e.g.* collisions, or one drone's wake disturbing another drone)
- Shared information: on-board and ground systems for different agents can communicate over shared networks (*e.g.* WiFi or telemetry)

The importance of interactions between the humans involved, not shown on the diagram, will depend on the nature of the work being undertaken. Typically, the solution to a multi-UAV simulation will depend on the relative importance of the different types of interaction to the research underway. For example, it would be extremely challenging to capture collisions and aerodynamic interactions in real time, but then a good control system would be trying to avoid such things anyway. Therefore, it might be enough to ignore these issues in the simulation engine and simply validate the result afterwards by checking separations. This represents one design choice in the space of assumptions that might be made to simplify the challenge of multi-vehicle simulation. The remainder of this section uses three examples to illustrate the different software solutions that result.

3.1 Centralized Simulation in Simulink

Simulink offers good modularity, in the sense that systems can be easily encapsulated as subsystems, and reproduced by simple copying and pasting. However, the graphical interface limits the scalability that results, due to the extensive amount of clicking needed to join up growing numbers of agents. Simulink does, however, support *programmatic modeling*[7], which opens the possibility of automated generation of models for different numbers of

[7] `https://uk.mathworks.com/help/simulink/ug/approach-modeling-programmatically.html`

vehicles. For a large number of homogeneous vehicles, this would be an effective approach, and would also have the potential to capture physical interactions between them, especially if the interactions can be captured as a summation of forces or similar. Information interaction could be modeled using individual signals as explicit point-to-point communications, but these would have to be fixed at the time of model generation and would not therefore support dynamic changes in connectivity. More scalably, a multiplexed signal could be used as a form of blackboard system [2] from which all agents could access information about all others. Although easier to implement, this again limits the diversity of communication available.

Summary: a programmatically-scaled Simulink model offers a route to a *centralized* simulation. This has the potential advantage of accurately modeling physical interactions, but hinders the inclusion of a diverse set of vehicles.

3.2 Agent-based Modeling

The term Agent-based Modeling (ABM) relates to a vast field of research, popular in social science and often with comparatively little overlap with technology-focused multi-agent systems research [7]. Here, it is used as a sloppy shorthand for a particular object-oriented programming approach, in which an 'agent' base class is defined with a standard interface, such that all agents derived from the base class can join an integrated simulation. There are numerous free off-the-shelf tools available for ABM, but all seem focused at non-technical research and therefore make significant compromises in agent functionality in order to reduce the need for users to have strong programming backgrounds. It is therefore likely (if regrettable) that any multi-UAV simulation using this approach will be built from scratch. Care is needed particularly with timing in ABM. Time-stepping models are simpler to implement but come with the risk of poor results due to the artificial synchronization of the agents, overcome by tricks such as random re-ordering or reversals of agent sequencing. Discrete event simulation is more accurate but potentially harder and slower. Physical interaction is likely to be hard to capture, given the potential diversity of agents, but information interaction would be readily interfaced through message-passing methods in the base class, as would a shared environment.

Summary: an ABM-like approach offers good scalability with more flexibility over agent diversity, since the full expressivity of the programming language is available to each derived agent class, within the limits of the base class interface. For example, it would be easy to mix agents with radically different flight physics models, provided a common command interface (*e.g.* MAVlink) was employed.

3.3 Multi-player Gaming

A more distributed simulation capability is provided by the multi-player facility within FlightGear [8]. Users can connect to a public server or can host their own private server using the open-source FGMS software [9] and a suitable hosting platform, such as Amazon Web Services. Each local simulation instance (a 'client') is then responsible for simulating the flight physics and local control system of one aircraft. The server simply shares location updates between clients, such that each displays a visual representation of all other traffic. While a shared visual environment is enabled by a common scenery database, weather and other physical environment features may not be shared: it would be quite possible for two pilots to fly in formation over the same places, one in strong wind and the other calm. For physical interaction, crude collision detection is provided, but wake interactions and direct sensing, *e.g.* different RADAR reflections depending on relative attitude, are not supported. Information sharing is limited to a simple native chat facility between pilots. More could be done, either by writing companion code to communicate using the chat, or by using other networking tools in parallel.

Summary: the most scalable approach, by having each simulation engine on a different computer, a multi-player game also offers great flexibility in terms of the diversity of agents and their controls. However, compromise is significant to enable this level of distribution, as the interactions supported are limited by the client/server protocol and the available bandwidth.

[8] `http://wiki.flightgear.org/Howto:Multiplayer`
[9] `https://sourceforge.net/projects/fgms/`

References

[1] P. Castillo, A. Dzul, and R. Lozano. Real-time stabilization and tracking of a four-rotor mini rotorcraft. *IEEE Transactions on Control Systems Technology*, 12(4):510–516, jul 2004.

[2] Daniel D Corkill. Blackboard systems. *AI expert*, 6(9):40–47, 1991.

[3] Ian D Cowling, Oleg A Yakimenko, James F Whidborne, and Alastair K Cooke. Direct method based control system for an autonomous quadrotor. *Journal of Intelligent & Robotic Systems*, 60(2):285–316, 2010.

[4] Xunhua Dai, Chenxu Ke, Quan Quan, and Kai-Yuan Cai. Unified simulation and test platform for control systems of unmanned vehicles.

[5] I. Elishakoff. Convex versus probabilistic modeling of uncertainty in structural dynamics. *Structural Dynamics Recent Advances*, pages 3–21, 1991.

[6] Dongjie Shi, Xunhua Dai, Xiaowei Zhang, and Quan Quan. A practical performance evaluation method for electric multicopters. *IEEE/ASME Transactions on Mechatronics*, 22(3):1337–1348, jun 2017.

[7] Michael P. Wellman. Putting the agent in agent-based modeling. *Autonomous Agents and Multi-Agent Systems*, 30(6):1175–1189, apr 2016.