Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

Unified DEVS-based platform for Modeling and Simulation of Hybrid Control Systems

Ezequiel Pecker-Marcosig

Departamento de Electrónica
FI, UBA and ICC, CONICET
Av. Paseo Colón 850
C1063ACV, Buenos Aires, ARGENTINA

Sebastián Zudaire

División de Física Estadística e Interdisciplinaria
Instituto Balserio - UNCuyo
Av. Exequiel Bustillo 9500
R8402AGP, Río Negro, ARGENTINA

Martín Garrett

División de Vibraciones
CNEA, Centro Atómico Bariloche
Av. Exequiel Bustillo 9500
R8402AGP, Río Negro, ARGENTINA

Sebastián Uchitel

Computación, FCEyN, UBA and ICC, CONICET
C1428EGA, Buenos Aires, ARGENTINA
Computing Department, Imperial College London
SW7 2RH, London, UK.

Rodrigo Castro

Departamento de Computación
FCEyN, UBA and ICC, CONICET
Ciudad Universitaria, Pabellón 1
C1428EGA, Buenos Aires, ARGENTINA

ABSTRACT

Recent robotic research has lead to different architectural approaches that support enactment of automatically synthesised discrete event controllers over low-level continuous variable controllers. Synthesis promises correct-by-construction plans from user provided high-level specifications. However, not only are mission specifications error prone, but it is also non-trivial to understand and anticipate the emergent behaviour of the resulting discrete-event/continuous-variable control stacks, i.e. hybrid controllers. Simulation of hybrid control approaches to robotics can be a useful validation tool for robot users and architecture designers. Yet, for simulations, working with discrete and continuous representations of the robot, its environment and its mission plans is also a challenge. In this work we address this challenge showcasing a unified DEVS-based hybrid simulation platform and modeling a fixed-wing UAV with a hybrid robotic software architecture. We validate the approach experimentally on a typical UAV mapping mission.

1 INTRODUCTION

The fields of reactive synthesis (Pnueli and Rosner 1989), supervisory control (Ramadge and Wonham 1987) and automated planning (Nau et al. 2004) all pursue the automatic construction of strategies that can guarantee a system goal in the context of an adversarial environment. In the context of robotics, this synthesis can be cast as the problem of building a discrete event controller that listens to events that the robot can sense and executes high-level commands that the robot can achieve in such a way that a mission goal

for the robot is guaranteed. Thus, synthesis promises the automatic construction of complex operational strategies that are guaranteed to achieve user-provided mission goals described in a high-level language.

Recent developments in *reactive synthesis* (e.g., Piterman et al. (2006)) are increasingly impacting research on how cyber-physical systems (CPS) can be built in order to exploit automatically synthesised discrete event controllers. *Hybrid controllers* (Kress-Gazit et al. 2009) are an emerging approach to CPS design in which a hybrid middle-level layer serves as the interface between a low-level layer of continuous variable controllers and a high-level layer of discrete event controllers. A key challenge in this domain is to find adequate discrete abstractions that *i)* correctly model actuating and sensing capabilities of the robot, *ii)* can be managed by the hybrid layer, *iii)* and provide sufficient flexibility to achieve a variety of missions (DeCastro et al. 2015; Maniatopoulos et al. 2016).

In the domain of Unmanned Aerial Vehicles (UAVs), hybrid control approaches have been proposed to support from motion and path objectives (Wei and Isler 2018; Ji and Li 2015) to task and mission goals (Wolff et al. 2013; Zudaire et al. 2020).

Testing CPS such as UAVs is a challenging task (e.g., Dokhanchi et al. (2017)). Hybrid controllers add the difficulty of dealing with unexpected emergent behaviour as a result of the complex interplay between discrete event and continuous control. In addition, synthesising discrete event controllers from user-provided specifications may mask unintended specification errors. As a consequence, techniques that support testing and validation of such systems appear as key tools to help UAV developers and users. Simulation of CPS, and in particular of UAVs, allows for quick development and testing of components and frameworks. For this reason, many customizable toolkits have been adopted for simulating aircrafts (e.g., Gazebo, JSBSim, and FlightGear). Adapting these simulators to specific UAVs is in general a challenging task: typically one must first correctly model all the different flight dynamics and controllers in order to later test the developed components. In the case of ArduPilot-based drones, a simulator with Software In The Loop (SITL) capabilities is available that simplifies modeling and simulation tasks, thus it has been adopted by many as a testing platform (Baidya et al. 2018; Barros et al. 2016). However, in all the aforementioned examples, it is mainly the continuous variable dynamic system that is simulated, while all software layers (in charge of discrete and hybrid control aspects) remain as pieces of code invoked from the simulator as external delegates, lacking a model counterpart.

In this paper we present a unified and flexible toolkit for modeling and simulation of the full stack of controllers (discrete, hybrid and continuous) in the domain of UAVs. We showcase an application tailored for the scenario of high-level plans on a fixed-wing UAV.

1.1 Motivating Case Study

In (Zudaire et al. 2020) we presented a discrete abstraction for specifying and synthesizing mission plans for workspaces involving hundreds of thousands discrete locations, which can be the case for open-flight UAV missions. We provided empirical validation through real flights and also via simulations. We created realistic simulation scenarios relying on the ArduPilot SITL-oriented simulator for the continuous aircraft dynamics and feedback-controllers communicating to a higher-layer running the hybrid control software deployed on the on-board processor. Although this setup correctly captures the interaction of components in real-flight scenarios, we also found several limitations in the simulation methodology. The first is the difficulty in modifying the default aircraft model of the SITL. An external simulator (e.g., Gazebo or JSBSim) is required to adapt the dynamic model to our custom vehicle. A second aspect, is that in order to run full system-level simulations, in the SITL approach multiple independent components interact over TCP/UDP network sockets where time synchronization is treated in a best-effort manner, without a parallel/distributed simulation mechanism. User-defined simulation speed-up parameters must be estimated heuristically for each scenario to avoid incurring in overruns (loss of messages and derived impacts on accuracy) which can become both a challenge and a risk when dealing with evolving software components.

Therefore, our goal is to explore a different path, resorting to a unifying formalism and tool that permits combining all layers of models within a single hybrid simulation framework. Our hypothesis is

that such an approach can provide a sound simulation foundations and machinery while simplifying the typical workflow of model-based design and simulation-based testing of hybrid controllers for UAVs.

In this work we model the hybrid control architecture used in (Zudaire et al. 2020) and simulate it for several missions taken from the literature. We show that with our unified approach we were able to achieve simulation speed-ups up to one order of magnitude above our previous SITL-based simulation setup.

2 BACKGROUND

2.1 Hybrid Control Architectures

Hybrid control architectures (e.g., Zudaire et al. (2020) and Kress-Gazit et al. (2008)) structure the software in three distinct layers with different control notions. Figure 1, which represents the architecture of a simulation model, also helps with visualizing how the three software layers interact for the specific scenario of a UAV Hybrid Control System:

- **Discrete Event Layer:** Here we find a high-level discrete event controller that is the output of a synthesis algorithm. The algorithm's input is a mission specification and discretization of the robot's capabilities and sensors. The controller is responsible for calling commands as they become enabled and process any incoming events as they occur.
- **Hybrid Control Layer:** This layer provides the translation between the high-level discrete event controllers and the lower-level feedback controllers and actuators, doing complex computations when required (e.g., motion planning to avoid static and/or dynamic obstacles).
- **Robot Layer:** In addition to the hardware, this layer has the feedback loops for controlling continuous variables related to movement and other capabilities (e.g., height stabilization, speed control and payload deployment).

2.2 DEVS for M&S of Hybrid Control Systems

Modeling and simulation of Cyber-Physical Systems (CPS) involves the combined modeling of continuous and discrete time signals and of discrete events. Continuous variables typically describe the physics governing the system, while discrete time and discrete event variables describe the digital parts.

Continuous time systems are usually represented by sets of ordinary differential equations (ODEs) for which the classical numerical approach is to discretize time by means of discrete-time solvers (Cellier and Kofman 2006). Yet, the discrete time base chosen for solving an ODE does not necessarily match with that of other clocked signals or unclocked discrete events pervasive in CPS. From the point of view of an ODE numerical solver, any event occurring in-between discrete time steps represents a discontinuity, that needs to be correctly and efficiently detected and solved. All discrete-time solvers (DTS) must incur in expensive iterative algorithms to hit a discontinuity, reset their clocking and restart the simulation from there. Although possible, this strategy does not scale well.

A dual to the traditional idea of time discretization is state-quantization. This idea was first proposed by (Zeigler and Lee 1998) and later improved by (Kofman and Junco 2001), giving rise to the Quantized State System (QSS) family of methods (Cellier and Kofman 2006). QSS quantizes state variables instead of discretizing time, and solve ODEs using discrete-event approximations of continuous signals. QSS methods approximate the state variables through discrete quanta; new integration steps can only happen whenever a state variable deviates by a predefined amount (accuracy control) from its expected solution. Since each new step is a discontinuity in the quantized variable –strictly speaking, a discrete event– the method is naturally able to coexist with synchronous or asynchronous discrete signals. This class of solvers present some advantages compared to their discrete-time counterparts, highlighting stability, convergence and global error bounds (Kofman 2004).

The Discrete-EVent Systems Specification (DEVS) for modeling and simulation is a formal approach to represent discrete-events systems (Zeigler et al. 2018). It was shown that DEVS can represent any kind of

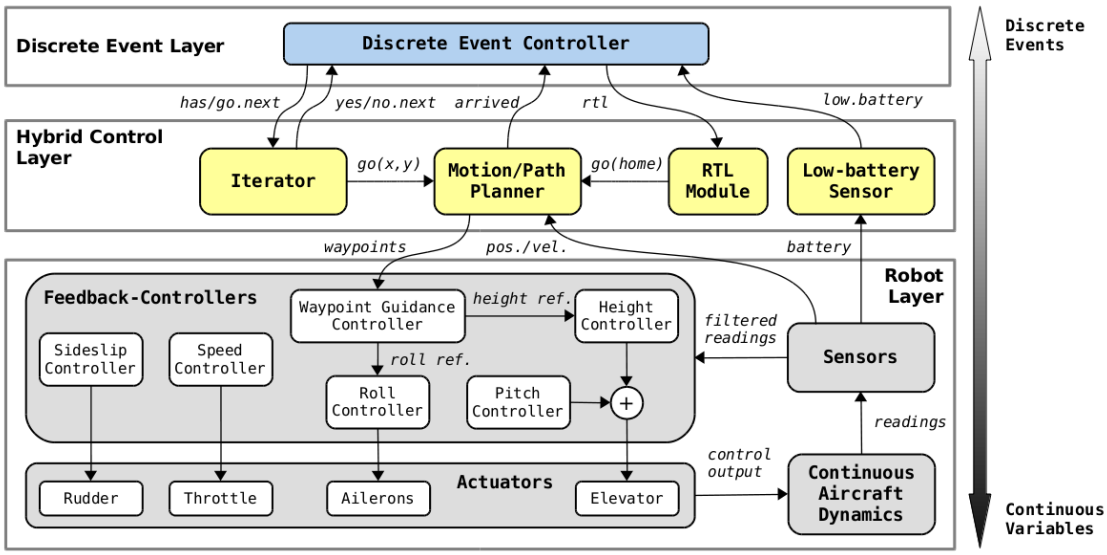


Figure 1: Simulation model for the UAV Hybrid Control System: components and interaction.

discrete (time or event) system and approximate continuous systems with any desired level of accuracy (e.g. using QSS integrators, which can be described as DEVS models). Thus, the DEVS formalism allows for full modeling and simulation of both the continuous and discrete (time and event) components of a hybrid CPS, as shown in (Pecker-Marcosig et al. 2017). PowerDEVS (Bergero and Kofman 2011) is the flagship DEVS simulator for the QSS family of solvers, providing excellent performance features (Van Tendeloo and Vangheluwe 2017), thus standing as a natural choice for CPS simulation.

3 SIMULATION MODEL ARCHITECTURE

Figure 1 provides an overview of the simulation model for our target fixed-wing UAV hybrid control system. Note that the structure of the simulation model mimics that of the software architecture in (Zudaire et al. 2020) and contains the three abstraction layers discussed in Section 2.1. Here we can see the different components that must be modeled (involving different formal paradigms) and their interactions.

For the Discrete Event Layer, we implement Finite State machines which can be straightforwardly represented by DEVS (Wainer 2009). For the Hybrid Control Layer we leverage the PowerDEVS support for writing custom atomic models and reused previously developed custom C/C++ libraries (Zudaire et al. 2020) and Python code (using a Python-C API). Finally, for the Robot Layer, we use the built-in *component engineering view* common in tools such as Simulink, where feedback-controllers and differential equations can be modeled by selecting and connecting simple boxes such as gains, integrators, math operators, etc.

In the following sections we explain how components in each layer were developed in PowerDEVS. The end result of this process is the box view shown in Figure 2.

4 CONTINUOUS AIRCRAFT DYNAMICS

We use as a reference the model presented in (Azocar and Valasek 2014): a high fidelity, non-linear model for fixed-wing aircrafts. The model consists of first-order differential *Motion Equations* and a set of function structures to model the *Aerodynamic* forces and moments. We stress the capability of the DEVS formalism to represent in full detail the complex non-linear continuous dynamics of an aircraft, for which we offer details below.

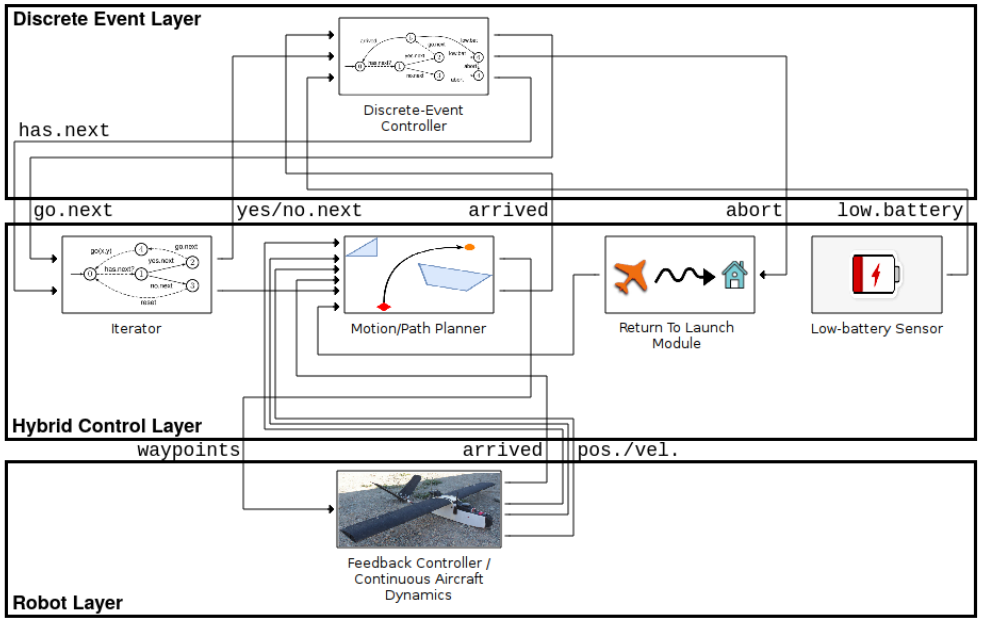


Figure 2: PowerDEVS simulation model

4.1 Motion Equations

This model consists of multiple non-linear differential equations. As an example we show one of the rotational equations of motion relative to the moving aircraft frame:

$$\dot{q} = \frac{\overline{q}_\infty \cdot S \cdot \overline{c} \cdot C_m - p \cdot r \cdot (I_{xx} - I_{zz}) - I_{xz} \cdot (p^2 - r^2)}{I_{yy}} \tag{1}$$

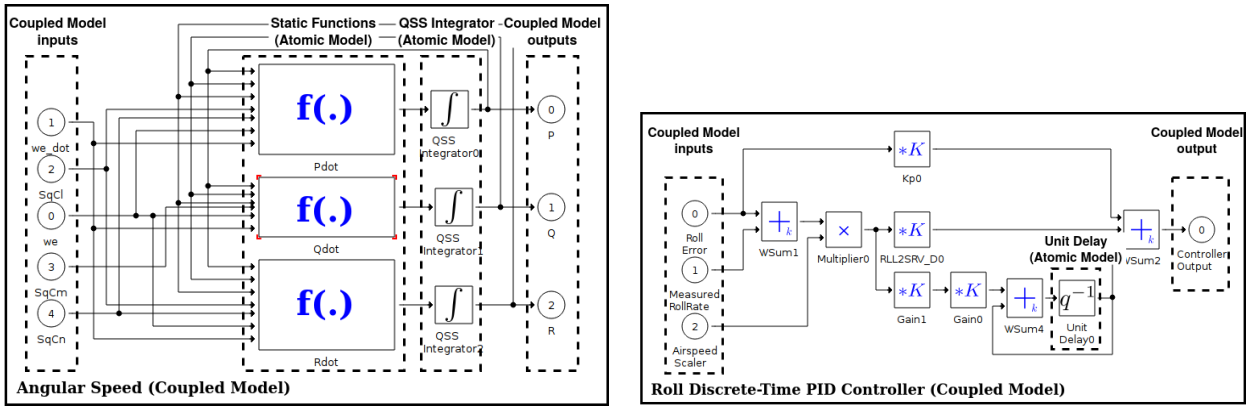
Equation (1) shows the evolution of the (locally-referred) angular velocity q (Y axis), and its dependence on several construction parameters of the aircraft $(S, \overline{c}, I_{xx}, I_{yy}, I_{zz}, I_{xz})$, the atmospheric coeficient $\overline{q}_\infty$, the aerodynamic moment C_m and other dynamic variables of the model (p, r). This equation together with similar ones for the angular velocities p and r can be easily implemented in PowerDEVS as shown in Figure 3a. Many of the construction parameters of the aircraft (mass, cord length $\overline{c}$, wing surface area S) can be measured directly from the real aircraft. Remaining parameters (i.e., the inertia tensor) were obtained from the SolidWorks model of the vehicle shown in Figure 4a.

4.2 Aerodynamic & Propeller Model

The work in (Azocar and Valasek 2014) provides modeling relations between the aerodynamic forces (C_X, C_Y, C_Z) and moments (C_l, C_m, C_n) and dynamic parameters from the aircraft (speed V, sideslip angle β, angle of attack α, rudder δ_r, aileron δ_a and elevator δ_e angles, etc.). As an example we show the relation for the C_m moment (Y axis):

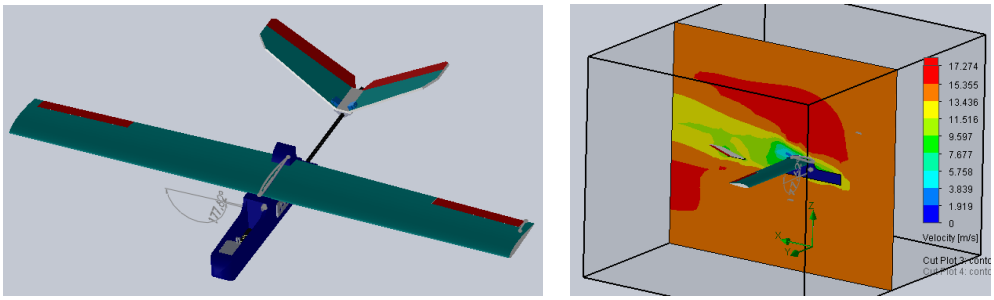
$$C_m = C_{m_0}(\alpha, \beta) + C_{m_{\delta_e}}(\alpha, \delta_e) + \frac{\overline{c}}{2V} \cdot C_{m_q}(\alpha) \cdot q \tag{2}$$

These aerodynamic relations were obtained by means of simulation using the SolidWorks model shown in Figure 4a together with its *Flow Simulation toolbox* which allowed us to create flow conditions with controlled parameters as shown in Figure 4b. The result was a table of derivatives for every C_i, such as $C_{m_0}(\alpha, \beta)$, $C_{m_{\delta_e}}(\alpha, \delta_e)$ and $C_{m_q}(\alpha)$ of equation (2), which were implemented in PowerDEVS as a static function which performs a linear interpolation for every external event.



(a) (b)

Figure 3: PowerDEVS block-oriented interface. (a) Differential Equations for equations of motion, (b) Difference equations for Discrete-time aircraft's roll controller.



(a) (b)

Figure 4: (a) SolidWorks model for the aircraft, where control surfaces (ruddervators and ailerons) are shown in red. (b) SolidWorks flow simulation output (velocity profile) for a stream of $15\,\mathrm{m\,s^{-1}}$ and $\alpha = 15°$.

The $10'' \times 6''$ propeller model was taken directly from the manufacturer's specifications and performance data, which provides a relation between the thrust and the engine shaft speed. Here we made a minor simplification, ignoring the internal (very fast) dynamics of the Electronic Speed Controller present in the real aircraft and connected its output directly to the engine shaft speed through a gain.

5 FEEDBACK-CONTROLLERS

Complex non-linear feedback-controllers are the main components of a UAV autopilot. These controllers live in a mixed-domain, where most control laws are continuous in nature (or discretized for a fixed time step) but with many event-driven aspects such as the waypoint guidance controller, where different control policies are applied depending on the proximity and history of the aircraft regarding the waypoints. Modeling sensor or actuator dynamics is similar in essence to modeling the dynamics of the aircraft. Since these components have fast dynamics, we provide simplified *Sensor* and *Actuator* implementations.

We modeled and implemented in PowerDEVS most of these controllers exactly as in the ArduPilot control diagrams, namely: sideslip, roll, pitch and waypoint guidance controller. We made minor modifications to reduce development time, replacing the joint height and speed controller (i.e., Total Energy Control System) with two independent PID controllers as shown in Figure 1.

Except for the waypoint guidance system, we implemented the controllers using the component engineering view of PowerDEVS which allows for fast and easy implementation. We show as an example the roll controller implementation in Figure 3b. The waypoint guidance system was implemented in Python and embedded within the PowerDEVS simulation, which allowed us to reuse previously developed modules.

6 HYBRID CONTROL LAYER & DISCRETE EVENT LAYER

Key components of the *Hybrid Control Layer* are the *Iterator* and *Motion/Path Planner* modules. They are responsible for most of the two-way discrete/continuous conversions between the top-level Discrete Event Controller and the low-level continuous Feedback-Controllers. We also implement an auxiliary *Low-battery Sensor* showing how discrete threshold-like hybrid sensors can be easily incorporated and simulated.

6.1 Iterator

The key component of *iterator-based* mission plans consists of an iterator data-structure. An iterator is an abstract data type that manages a set of cells derived though the discretization of a region (the approach is oblivious to which discretization method is used). The iterator is initialized with a current element c set to null. An API is provided that works similarly to high-level languages:

- **has.next?**: Queries if there are remaining elements in the iterator which have not yet iterated over. If there are, it returns **yes.next**, otherwise returns **no.next**.
- **remove.next**: Removes the current element from the iterator and selects a new current element from the remaining list or null if there are no more elements.
- **reset**: Resets the iterator inserting again all the removed elements.
- **go.next**: Commands the *Motion/Path Planner* to go to the location currently selected.

For simplicity, we model this API using the FSM shown in Figure 5b, allowing for easy integration with the *Discrete Event Controller*. We implemented this behavior in PowerDEVS by translating the FSM to a DEVS atomic model (Wainer 2009) and integrating the action remove.next into has.next? since usual specifications of iterator-based missions involve removing (remove.next) after each has.next? (Zudaire et al. 2020). The set of cells is read from a CSV file at the beginning of the simulation. An important component related to the iterator that we chose not to model was the location sorter, as it does not add new layers of continuous/discrete interaction nor helps showcase new features from PowerDEVS.

6.2 Motion/Path Planner

This module is in charge of computing a sequence of waypoints that allows the aircraft to reach a target location. The Motion/Path Planner is a hybrid dynamic submodel, interfacing mostly continuous (or discrete-time) and discrete event submodels. The PowerDEVS implementation receives the aircraft position and orientation consisting of QSS signals from the Robot Layer, and interfaces with the Discrete-Event Controller, Iterator and Abort Module through discrete events.

We leveraged the fact that PowerDEVS is written in C++ to include straightforwardly, within a DEVS atomic model, the exact same code module that runs onboard the real UAV in (Zudaire et al. 2020), which is written in C. This module computes a Dubins Path (Song and Hu 2017) from the current to the target location and discretizes it into a series of intermediate waypoints for the UAV. Some of the parameters of the algorithm are the turn radius, heading and arrival direction. The latter can be set to a *parallel mode* (i.e., forcing arrival directions parallel to one of the grid axis) or to a *minimum distance mode* (i.e., setting an arrival direction that minimizes the flight distance). The *parallel mode* usually provides more orderly flight paths while the *minimum distance mode* can result in shorter trajectories when the mission involves visiting fewer locations.

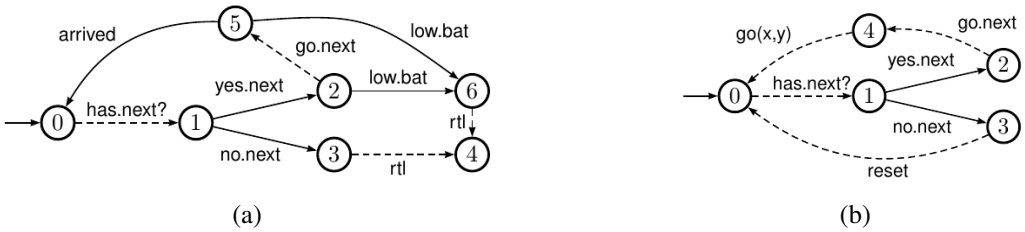


Figure 5: (a) Discrete-Event Controller. (b) Iterator. *Note:* Dashed-lines denote controllable actions and continuous lines uncontrollable events.

6.3 Discrete Event Controller & Auxiliary Modules

We modeled typical *iterator-based* controllers synthesized for missions described in (Zudaire et al. 2020). We purposely reduced the number of model states for these controllers to facilitate its manual implementation in PowerDEVS, given the proof-of-concept nature of the current approach. Yet, the automatic generation of DEVS models from FSM specifications of controllers is possible (a full integration of the controller synthesis algorithms with generators of DEVS models for PowerDEVS will be the subject of future work).

In Figure 5a we show an example of a simplified controller for a mission that consists of visiting all the locations in the iterator and aborting the mission either because there are no remaining locations to visit or a low battery (low.bat) event occurs.

For testing purposes we modeled a very simple *Low-battery Sensor* as a step function that generates a low.bat event at predefined timestamps. We modeled the *Return To Launch (RTL) Module* to command the *Motion/Path Planner* to return the aircraft home, i.e. to a known user-defined (x, y) coordinate.

7 SIMULATION RESULTS

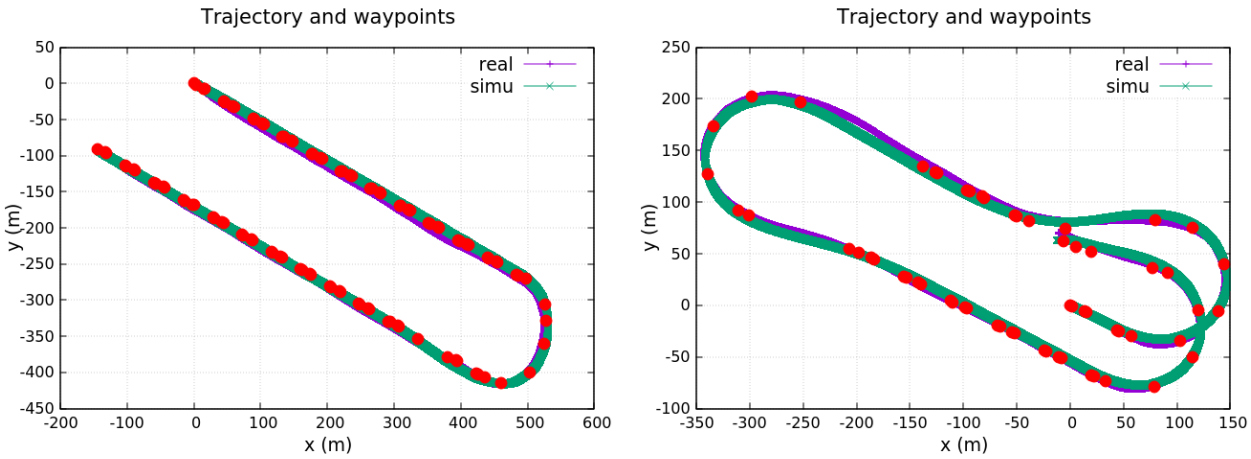
We first report on the assessment of the simulation model for the Robot Layer. We then report on the inclusion of the remaining layers and a mission run to cover a simple rectangular area with no obstacles nor no-fly zones. This mission is a classical example of the typical mapping application for an UAV, and at the same time represents a kind of mission that makes good use of an iterator-based planning approach (i.e., universally quantifiable locations). We assume a discretization of the coverage area into a set of rectangular non-overlapping cells of $50\,\mathrm{m} \times 50\,\mathrm{m}$.

7.1 Model Validation

In this section we aim to show the realism achieved for the *Robot Layer*. We use logged data from a real flight of the UAV executing a find mission as described in (Zudaire et al. 2020) (see Mission video (2019)) and fed the flown waypoints to the *Waypoint Guidance Controller* in the PowerDEVS simulation, setting the speed and height references with appropriate values.

We selected two sections of the mission to show both straight and turning paths. Without any modification or adjustment to the simulation we obtained the results shown in Figure 6a and Figure 6b. A great degree of similarity is observed between the real and simulated UAV trajectories, where the maximum difference observed was of 12 m. We note that the modeling assumptions and simplifications made for the speed and height controllers (described in Section 5) did not impact significantly in the accuracy of the simulated path (both speed and height references are kept constant during both the real and simulated flights).

Given the purpose of this work, we believe that further model validation is not required. However, validation of the inner parts of the *Robot Layer* (e.g., the *Continuous Aircraft Dynamics)* can easily be done by replacing the inputs to the corresponding block with the respective measured inputs from the real flight in a open-loop fashion (special care must be taken to select short enough time-windows, as the non-linear dynamics of a fixed-wing aircraft are very unstable as described in (Cook 2013)).



(a) (b)

Figure 6: (a) First section validation. (b) Second section validation. In both, real and simulated paths are shown in purple and green respectively, red marks are the waypoints and the acceptance radius is $\leq 40\,\text{m}$.

7.2 Coverage Mission

In Figure 7 we show the trajectory of a simulated mapping mission. Between two consecutive locations, multiple waypoints are generated by the *Motion/Path Planner*. Locations were fed to the iterator in a zig-zag order emulating the behaviour of the sorter component described in (Zudaire et al. 2020). Note that after every straight line, the *Motion/Path Planner* computes a looping trajectory (with a 65 m turn radius) to correctly arrive at the next line of locations with an arrival direction parallel to the grid axis. Also, once the aircraft reaches the final location and in absence of a next location to feed the Motion/Path Planner, the Discrete-Event Controller directs a return to home through an rtl command. Albeit simple, this mission demonstrates the correct behavior of the full Hybrid Control System in a simulated scenario.

Figure 8 depicts a variation of the previous mission, where a low.bat event may be generated by the *Low Battery Sensor* at one point of the simulated trajectory. Upon receiving the event, the Discrete-Event Controller sends an rtl command forcing the aircraft to turn right to return to home. This mission demonstrates how other non-movement related modules may be added into the model with minimal variations in the model architecture (see Figure 1) and minimal impact in the simulation performance.

Both no.next and low.bat are state-dependent discrete events that might appear at any time on a continuous time base depending on several factors (such as the region already covered, environmental conditions, etc.). Such events impose discontinuities to the numerical solution of the underlying differential equations in the system that are treated transparently from the modeling perspective (i.e., no special care is required when modeling) and efficiently from the simulation perspective (the DEVS-based QSS solver detects correctly the discontinuity at the exact timestamp with negligible computational cost).

8 DISCUSSION AND RELATED WORK

The simulation of UAVs and their associated missions can be approached in several ways, resorting to varied available toolkits. Some are typically used as a backend that simulates the continuous variable dynamics of the robots (e.g., Gazebo, JSBSim, ArduPilot), which are later interfaced with external middleware for robotic applications (such as ROS (Sagitov and Gerasimov 2017)) or Mission Planners (e.g., MAVProxy).

It was recently reviewed in (Hentati et al. 2018) that no single UAV-specific tool allows for implementing the full hybrid system; while some tools are intended for flight dynamics simulation, others focus on flight

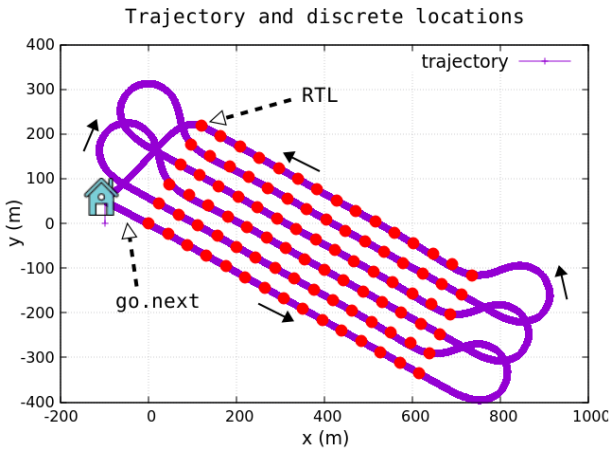


Figure 7: Coverage mission.

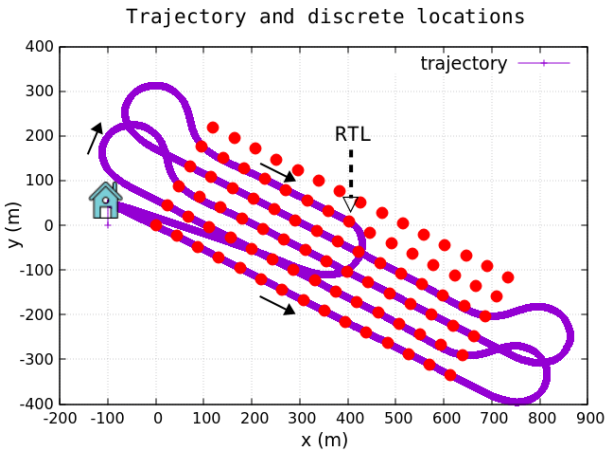


Figure 8: Mission with a low-battery alarm.

planning simulation. This can be seen as a natural split, a consequence of the inherently different concerns that dominate in the discrete and continuous domains.

Yet, when resorting to interfacing heterogeneous tools, coherent time management and event synchronization among simulators (or between a simulator and external piece of code) becomes a key issue and a challenge, impacting both on simulation accuracy and simulation performance.

In fact, in our previous experiences using an interfaced MAVProxy+ArduPilot SITL strategy for the full hybrid controller, synchronization losses between processes and threads kick in at a 7× speed-up factor (7 times faster than the real flight time) even when processor usage is never higher that 70% for each core on a Linux i7 3.5 GHz 12 GB laptop. This means for example that discrete events such as the arrival at a location (arrived) occurs when the simulated UAV is already several meters past the arrived location, leading to the *Motion/Path Planner* calculating a trajectory with wrong data that later causes the UAV to completely miss its next target location.

In this context, *co-simulation middleware* are usually a reliable solution, as they provide sound time and event synchronization guarantees in a tool-agnostic fashion: it is only required that the set of connected simulators implement the proper co-simulation interfaces. Salient technologies in this area are FMI (Functional Mock-up Interface) and HLA (High Level Architecture). Note that ROS (Sagitov and Gerasimov 2017), the popular Robot Operating System for interconnecting modules in robotic applications, is neither a co-simulation middleware nor an operating system with scheduling guarantees. Examples for ArduPilot-based UAVs are FlyNetSim (Baidya et al. 2018) (a middleware to interface ArduPilot with the ns-3 network simulator) and (Barros et al. 2016) that considers HLA to link SITL/ArduPilot with the Ptolemy toolkit.

Meanwhile, general-purpose all-in-one toolkits, such as Matlab/Simulink, offer the possibility of integrative hybrid simulation of UAVs by means of collections of proprietary plug-ins (or toolboxes) such as the Aerospace Blockset, FlightGear and/or StateFlow. They are expected to interact relying on the Matlab core engine that is in charge of resolving the synchronization of continuous, discrete time and event-based signals. In this case the algorithms that handle such orchestration are not available as they are part of a commercial product. For a comparison between Matlab/Simulink and JSBSim for flight dynamics please refer to (Cantarelo et al. 2016).

A crosscutting aspect present in all these tools is that they use *time slicing-based* integration algorithms to approximate the solutions of the continuous parts. This represents as a second challenge: the correct and efficient treatment of complex interactions between discrete events and the discrete time approximations of continuous variables (i.e., the numerical solutions of differential equations). In this context, the arrival of a discrete event represents a discontinuity that calls for expensive iterative algorithms to correctly re-synchronize the discrete and continuous parts.

Quantization-based integration algorithms such as QSS stand as naturally suitable for such hybrid setting (see Section 2.2) as they solve differential equations by means of sequences of discrete events. See (Migoni et al. 2012) for a comparison between time slicing vs. state-quantization in the context of hybrid systems. QSS typically outperforms time slicing solvers in systems with very frequent discrete events.

In (Camus et al. 2018) the authors adopt a strategy for hybrid CPS modeling and simulation resorting to DEVS and QSS while considering also co-simulation using the FMI standard.

As mentioned in Section 2.2, PowerDEVS (Bergero and Kofman 2011) is a natural option to use DEVS with QSS. As with most DEVS-based tools, libraries of reusable components can be developed for discrete event, discrete time and continuous simulation. Like in the Matlab/Simulink case, it represents an all-in-one approach, avoiding extra layers of communication among heterogeneous controllers, or the adoption of an extra middleware for co-simulation. This aspect eliminates potential risks of synchronization mismatches and reduces simulation overheads due to extra communication delays. Also, the correct and efficient orchestration of events is granted by the DEVS abstract simulator, which implements a well-known formalism. In (Pecker-Marcosig et al. 2017) we used PowerDEVS to simulate a hybrid controller (matching the layers of Figure 1) that stabilizes a UAV for data collection from ground sensors, but without modeling a path plan nor individual movements between patches of sensors.

Regarding performance, in our approach simulation speed-up is mainly limited by the hardware resources and by the efficiency of the underlying DEVS core engine. As a given real system admits many possible DEVS models, modeling decisions can affect also performance such as the degree of modularity and levels of hierarchy adopted. In this work we privileged model readability by a one-to-one mapping between DEVS modules an their real system's counterparts. This comes at the price of extra messaging impacting performance. Yet, even in this setting where performance is not a modeling goal, we achieved a $50\times$ speed-up with room for performance improvements by applying standard techniques such as model flattening.

In Table 1 we show a comparison of a non-exhaustive set of commonly used UAV-related simulators for several relevant aspects, showing the main features of our approach.

9 CONCLUDING REMARKS AND FUTURE WORK

In this paper we presented a unified and flexible DEVS-based toolkit for modeling and simulation the full stack of a hybrid control architecture (discrete, hybrid and continuous) targeted for a fixed-wing UAV. We then validated this tool against logged data from a real flight and simulated a typical high-level mapping mission, returning home after an rtl command.

Our simulator built on PowerDEVS is capable of managing all dynamics (discrete event, discrete time and continuous) involved in the hybrid system. We simulated from differential equations representing continuous dynamics of the aircraft on the bottom using QSS integrators to the FSM for the discrete event controller on the topmost layer, going through the different control notions. Either by linking a graphical representation of DEVS atomics or describing its behavior in a written manner, both approaches rely on well established formal guarantees provided by the formalism. To the best of our knowledge this is the

Table 1: Comparison of simulators for UAV-related applications. Aspects: *Native 3D Animation, Integration Algorithm, Licensing, Hybrid Simulation capabilities, ROS Integration, Block-oriented, Unified.*

Simulator	3D Anim.	Integ. Alg.	Licensing	Hybrid Sim.	ROS Integ.	Block-or.	Unified
Gazebo	✓	DTS	Apache	×*	✓	×	✓**
JSBSim	×	DTS	LGPL	×	×*	×	×
Simulink	✓**	DTS	Proprietary	✓	✓	✓	✓
ArduPilot	×	DTS	GPL	×	✓	×	×
PowerDEVS	×	QSS	LGPL	✓	✓	✓	✓

(*) by default. (**) not generally used this way.

first time that a highly detailed model of an UAV and its associated discrete-event and continuous variable controllers is built under the DEVS formalism.

We presented a non-exhaustive list of general-purpose robot and UAV simulators and some useful middleware technologies to interconnect different tools so as to cover all the layers required for hybrid control. Our unified approach allows for simulation speed-ups of up to one order of magnitude above our previous middleware-based simulation setup. An additional benefit of the choice of a single simulator that can provide strong simulation performance paves the way for running on-board in-flight simulations. We aim at running faster than real-time simulations on a modest processing hardware, allowing for in-flight evaluation of alternative mission outcomes and selection of planning options based on simulation outputs (e.g., predict potential mission failures and deploy a fallback plan).

Finally, its modular design leaves room for several improvements and the incorporation of additional features, such as an energy-consumption model, as well as automatic detection of critical remaining battery for the aircraft. Wind models might also be incorporated to test the effect of wind gusts on a mission/flight-plan and the interplay between the aircraft and its environment. Additionally, we hope to achieve full integration between the synthesis algorithm in (Zudaire et al. 2020) and PowerDEVS.

REFERENCES

Azocar, A., and J. Valasek. 2014. "High Fidelity Simulation of a Nonlinear Aircraft". In *Proc. of the AIAA Science and Technology Forum and Exposition 2014: 52nd Aerospace Sciences Meeting*: American Institute of Aeronautics and Astronautics.

Baidya, S., Z. Shaikh, and M. Levorato. 2018. "FlyNetSim: An Open Source Synchronized UAV Network Simulator Based on Ns-3 and Ardupilot". In *Proc. of the 21st ACM Int. Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 37–45. New York, NY, USA: ACM.

Barros, J., T. Oliveira, V. Nigam, and A. V. Brito. 2016. "A Framework for the Analysis of UAV Strategies Using Co-simulation". In *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*. Nov. 1st-4th, J.Pessoa, Brazil, 9-15.

Bergero, F., and E. Kofman. 2011, 01. "PowerDEVS: A Tool for Hybrid System Modeling and Real-Time Simulation". *SIMULATION* 87(1-2):113–132.

Camus, B., T. Paris, J. Vaubourg, Y. Presse, C. Bourjot, L. Ciarletta, and V. Chevrier. 2018. "Co-simulation of Cyber-Physical Systems Using a DEVS Wrapping Strategy in the MECSYCO Middleware". *SIMULATION* 94(12):1099–1127.

Cantarelo, O., L. Rolland, and S. O'Young. 2016. "Validation Discussion of an Unmanned Aerial Vehicle (UAV) Using JSBSim Flight Dynamics Model Compared to MATLAB/Simulink AeroSim Blockset". In *2016 IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)*. Oct. 9th-12th, Budapest, Hungary, 3989-3994.

Cellier, F. E., and E. Kofman. 2006. *Continuous System Simulation*. New York, NY, USA: Springer Science. & Business Media.

Cook, M. V. 2013. *Flight Dynamics Principles*. 3rd ed. USA: Butterworth-Heinemann.

DeCastro, J. A., V. Raman, and H. Kress-Gazit. 2015. "Dynamics-Driven Adaptive Abstraction for Reactive High-Level Mission and Motion Planning". In *2015 IEEE Int. Conf. on Robotics and Automation (ICRA)*. May 26th-30th, Seattle, USA, 369-376.

Dokhanchi, A., B. Hoxha, and G. Fainekos. 2017, December. "Formal Requirement Debugging for Testing and Verification of Cyber-Physical Systems". *ACM Trans. Embed. Comput. Syst.* 17(2).

Hentati, A. I., L. Krichen, M. Fourati, and L. C. Fourati. 2018. "Simulation Tools, Environments and Frameworks for UAV Systems Performance Analysis". In *2018 14th Int. Wireless Communications Mobile Computing Conference (IWCMC)*. Jun. 25th-29th, Limassol, Cyprus, 1495-1500.

Ji, X., and J. Li. 2015. "Online Motion Planning for UAV Under Uncertain Environment". In *2015 8th Int. Symp. on Computational Intelligence and Design (ISCID)*, Volume 2. Dec. 12th-13th, Hangzhou, China, 514-517.

Kofman, E. 2004. "Discrete Event Simulation of Hybrid Systems". *SIAM Journal on Scientific Computing* 25(5):1771–1797.

Kofman, E., and S. Junco. 2001. "Quantized-State Systems: a DEVS Approach for Continuous System Simulation". *Transactions of The Society for Modeling and Simulation International* 18(3):123–132.

Kress-Gazit, H., G. Fainekos, and G. Pappas. 2008, 10. "Translating Structured English to Robot Controllers". *Advanced Robotics* 22:1343–1359.

Kress-Gazit, H., G. E. Fainekos, and G. J. Pappas. 2009. "Temporal-Logic-Based Reactive Mission and Motion Planning". *IEEE Transactions on Robotics* 25(6):1370–1381.

Maniatopoulos, S., P. Schillinger, V. Pong, D. C. Conner, and H. Kress-Gazit. 2016. "Reactive High-Level Behavior Synthesis for an Atlas Humanoid Robot". In *2016 IEEE Int. Conf. on Robotics and Automation (ICRA)*. May 16th-21th, Stockholm, Sweden, 4192-4199.

Migoni, G., M. Bortolotto, E. Kofman, and F. Cellier. 2012, 04. "Quantization-Based New Integration Methods for Stiff Ordinary Differential Equations". *Simulation Modelling Practice and Theory* 35:118–136.

Mission video 2019. "Search and Cover Mission". https://www.youtube.com/watch?v=SGNfY8T8QSA, accessed 22.03.2020.

Nau, D., M. Ghallab, and P. Traverso. 2004. *Automated Planning: Theory & Practice*. USA: Morgan Kaufmann Publishers Inc.

Pecker-Marcosig, E., J. I. Giribet, and R. Castro. 2017. "Hybrid Adaptive Control for UAV Data Collection: A Simulation-Based Design to Trade-Off Resources Between Stability and Communication". In *Proc. of the 2017 Winter Simulation Conference*, 1704–1715. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Piterman, N., A. Pnueli, and Y. Sa'ar. 2006. "Synthesis of Reactive (1) Designs". *Lecture notes in computer science* 3855:364–380.

Pnueli, A., and R. Rosner. 1989. "On the Synthesis of a Reactive Module". In *Proc. of the 16th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, POPL. Austin, TX, USA, 179-190.

Ramadge, P. J., and W. M. Wonham. 1987. "Supervisory Control of a Class of Discrete Event Processes". *SIAM Journal on Control and Optimization* 25.

Sagitov, A., and Y. Gerasimov. 2017. "Towards DJI Phantom 4 Realistic Simulation with Gimbal and RC Controller in ROS/Gazebo Environment". In *2017 10th Int. Conf. on Developments in eSystems Engineering (DeSE)*. June 14th-16th, Paris, France, 262-266.

Song, X., and S. Hu. 2017. "2D Path Planning with Dubins-Path-Based A* Algorithm for a Fixed-Wing UAV". In *2017 3rd IEEE Int. Conf. on Control Science and Systems Engineering (ICCSSE)*. Aug 17th-19th, Beijing, China, 69-73.

Van Tendeloo, Y., and H. Vangheluwe. 2017. "An Evaluation of DEVS Simulation Tools". *SIMULATION* 93(2):103–121.

Wainer, G. A. 2009. *Discrete-Event Modeling and Simulation: A Practitioner's Approach*. 1st ed. USA: CRC Press, Inc.

Wei, M., and V. Isler. 2018. "Coverage Path Planning Under the Energy Constraint". In *2018 IEEE Int. Conf. on Robotics and Automation (ICRA)*. May 21th-25th, Brisbane, Australia, 368-373.

Wolff, E. M., U. Topcu, and R. M. Murray. 2013. "Automaton-Guided Controller Synthesis for Nonlinear Systems with Temporal Logic". In *2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. Nov. 3rd-7th, Tokyo, Japan, 4332-4339.

Zeigler, B. P., and J. S. Lee. 1998. "Theory of Quantized Systems: Formal Basis for DEVS/HLA Distributed Simulation Environment". In *Enabling Technology for Simulation Science II*, edited by A. F. Sisti, Volume 3369, 49 – 58. International Society for Optics and Photonics: SPIE.

Zeigler, B. P., A. Muzy, and E. Kofman. 2018. *Theory of Modeling and Simulation Discrete Event and Iterative System Computational Foundations*. 3 ed. San Diego, CA, USA: Academic Press.

Zudaire, S., M. Garrett, and S. Uchitel. 2020. "Iterator-Based Temporal Logic Task Planning". In *2020 Int. Conf. on Robotics and Automation (ICRA)*. In press.

AUTHOR BIOGRAPHIES

EZEQUIEL PECKER-MARCOSIG is an Electronics Engineer and a PhD student in the Facultad de Ingeniería, Universidad de Buenos Aires and ICC-CONICET. His work is supported with a PhD Fellowship from the Peruilh Foundation. His email address is epecker@fi.uba.ar.

SEBASTIÁN ZUDAIRE is a Mechanical Engineer and a PhD student in the Insituto Balseiro - Universidad Nacional de Cuyo, with strong link to the Laboratory on Foundations and Tools for Software Engineering (LaFHIS), UBA. His e-mail address is sebastian.zudaire@ib.edu.ar.

MARTÍN GARRETT is a Mechanical Engineer and a Magister student in the Instituto Balseiro - Universidad Nacional de Cuyo. He has designed and constructed several UAVs, which he currently uses for mapping applications and research. His e-mail address is medgarrett@cab.cnea.gov.ar.

SEBASTIÁN UCHITEL holds a Professorship at Universidad de Buenos Aires and a Readership at Imperial College London. He is the Head of the Laboratory on Foundations and Tools for Software Engineering and of the CONICET Research Institute of Computer Science. His email address is suchitel@dc.uba.ar.

RODRIGO CASTRO is a Professor with the University of Buenos Aires and Head of the Laboratory on Discrete Events Simulation at the CONICET Research Institute of Computer Science. His research includes simulation and control of hybrid systems. His email address is rcastro@dc.uba.ar.