An Open-Source Framework to Design, Tune, and Fly Nonlinear Control Systems for Autonomous UAVs

An Open-Source Framework to Design, Tune, and Fly Nonlinear Control Systems for Autonomous UAVs

Mattia Gramuglia*, Giri Mugundan Kumar†
Virginia Tech, Blacksburg, VA, 24061

Giorgio A. Orlando‡
Politecnico di Torino, Torino, Italy

Andrea L'Afflitto§
Virginia Tech, Blacksburg, VA, 24061

This paper introduces a freeware open-source software ecosystem to design, tune, and test control systems for autonomous vertical take-off and landing (VTOL) multi-rotor uncrewed aerial vehicles (UAVs) such as quadcopters and quad-biplanes. This environment comprises C++-coded flight stacks with a suite of control systems, a high-fidelity simulator that allows model-in-the-loop and hardware-in-the-loop tests, computer-aided design (CAD) models of UAVs, and a website for a broad overview of this project. This ecosystem aims to serve as a common platform for the aerospace control community and ease comparative analyses for control design techniques produced by multiple research groups.

I. Introduction

Despite the decades of extensive experience in the area of control theory applied to multi-rotor and fixed-wing autonomous uncrewed aerial vehicles (UAVs), such as quadcopters, and the indisputable popularity of these platforms as challenging, and yet inherently simple, demonstrators for complex control solutions, it is safe to affirm that the aerospace control community does not share a reliable, commonly accepted hardware and software architecture. Some industrial operators provide platforms that could relieve researchers from the burden of designing their own vehicles and programming their own flight stack. However, several of these solutions include copyrighted components, which, in practice, play the role of black boxes in the control system, and, hence, pose notional challenges to control theoreticians and practitioners. Finally, there is no shared repository of control systems for quadcopters, which could serve as a data bank for the UAV control community and support those researchers interested in a comparative analysis between their own work and similar results by others. In this state of the art, numerous laboratories employ their own hardware and software architectures, and, in these conditions, performing fair comparative analyses is burdensome.

The proposed software ecosystem comprises two C++-coded open-source freeware flight stacks ([1] for multi-rotor UAVs and [2] for fixed-wing UAVs), a high-fidelity simulator that allows model-in-the-loop and hardware-in-the-loop tests [3], repositories of Matlab® codes for flight and simulation data post-processing [4, 5], computer-aided design (CAD) models of UAVs [4], and a website for a broad overview of this project [6]. Simulations are performed in a high-fidelity environment based on Project Chrono® [7, 8] (hereafter referred to as Chrono), which serves as a physics engine, enriched by JSBSim® [9] to model aerodynamic effects. For these features, the proposed software ecosystem can be consider unique in its kind.

Figure 1 provides a schematic representation of the key elements of the proposed framework. A CAD model of the UAV can be imported into the Chrono-based simulation environment for Model-In-the-Loop (MIL) and Hardware-In-the-Loop (HIL) simulations. Accurate CAD models are also useful to estimate inertial properties of the UAV to be used to tune the control system. In MIL tests, the proposed flight stacks are executed on the computer executing Chrono. In HIL and flight tests, the proposed flight stacks are exectured on a companion computer. In flight test mode, the companion computer is connected to a flight control unit (FCU), such as a PixHawk 6c, executing PX4. The FCU estimates the UAV state by emplying an extended Kalman filter (EKF) to merge data from its IMU (inertial measurement

*Graduate Student, Department of Mechanical Engineering.
†Graduate Student, Department of Mechanical Engineering.
‡Graduate Student, Department of Mechanical and Aerospace Engineering.
§Associate Professor, Grado Department of Industrial and Systems Engineering, AIAA Senior Member.

unit) and a motion capture (MoCap) system, a visual inertial odometry (VIO) system, or an RKT (Real Time Kinematic) GPS (Global Positioning System) antenna. MoCap data are transmitted directly to the companion computer via a WiFi antenna. The FCU is also employed to coordinate the propellers and ensure that the thrust force determined by the flight stack is produced.

The ultimate goal of this project, which has been recently launched, is to prime a community-led effort toward common, easily reproducible, hardware and software solutions within the multi-rotor UAV control community. At this stage, the classical proportional-integral-derivative (PID) control system and robust versions of the model reference adaptive control (MRAC) system have been disclosed. In the future, more advanced control systems will be introduced by the authors, and any researcher or practitioner will be able to add their controllers to the proposed GitHub repositories.

It is commonly acknowledged how tuning control systems for UAVs in simulation and on actual aircraft usually leads to substantially different sets of gains. The reasons for these discrepancies are numerous and usually rooted in key factors such as the accuracy of the underlying dynamical model or of the physics engine, the difference in programming language used to test a control system for UAVs on a PC, usually Matlab® or Simulink®, and code a flight stack, usually C++ or a variation thereof. For its high-quality physics engine and the ability to perform hardware-in-the-loop simulations, the proposed system aims to considerably reduce the gap between simulation results and flight tests.

This paper is organized as follows. Section II surveys the state of the art in flight stacks and associated simulators for autonomous UAVs. Section III outlines the hardware and communication architectures supporting the proposed flight stacks [1, 2] and the proposed simulator [3]. Section IV outlines the architecture underlying the flight stacks [1, 2]. Section V outlines the architecture underlying the simulation software [3], and Section VI details the physics engine that guarantees high fidelity simulation results. Finally, Section VII presents concluding remarks and outlines future work directions.

II. State-of-the-Art in Open-Source Flight Stacks

At the core of the proposed software ecosystem to design, test, and fly control systems for autonomous UAVs lie a flight stack for multi-rotor UAVs [1] and a flight stack for fixed-wing UAVs [2]. In the following, we survey the state of the art in open-source flight stacks for UAVs. As it is apparent from the following discussion, there are very few freeware, open-source flight stacks for UAVs and almost all of them are extremely recent, despite the lost-established popularity of UAVs.

The flight stack described in [10] is a high-level system designed to perform safe, fully autonomous, long-duration missions. Instead of concentrating on low-level control, this flight stack emphasizes continuous system integrity monitoring, initiating contingency actions, executing and adapting mission plans under non-nominal conditions, ensuring proper data logging, and providing standardized interfaces and integrity checks for external mission planners and localization modules. This flight stack uses ROS to interface with the PX4-Autopilot running on a Pixhawk board. The PX4 autopilot handles most low-level tasks, such as vehicle control, while higher-level functions like sensor failure detection, management, and mission re-planning are managed by the flight stack. Additionally, it offers a state estimator that can replace the PX4 autopilot's EKF. The source code for this flight stack is available at [11].

The flight stack referenced in [12] comprises a collection of software modules that form a foundational software stack for autonomous flight, built on the PX4 firmware and ROS 2 middleware with the FastDDS implementation. Its primary purpose is to facilitate communication and data exchange with the firmware while providing basic autonomous flight capabilities. This stack serves as a software abstraction layer for the low-level flight control firmware. It does not include localization or mapping functionalities, but can be integrated with other software stacks that offer these features, enabling the transfer of such information to the flight controller.

The work presented in [13] extends the PX4 autopilot firmware to support developers in integrating custom control algorithms alongside the existing control framework. This approach uses a template-based structure and introduces two new control modules, allowing users to leverage all firmware functionalities within their custom implementations while maintaining compatibility with standard modules and the QGroundControl interface. A key advantage of this framework is that the control algorithms run directly on the flight controller, providing access to all vehicle data at the highest available rate. Meanwhile, high-level tasks are executed on a companion computer that communicates with the Pixhawk via ROS2 (robot operating system 2) middleware, which subscribes to and publishes uORB topics. The source code for this software is available at [14].

The flight stack described in [15] is designed to facilitate the safe, real-world experimental validation of various approaches in planning, control, estimation, computer vision, tracking, and more. The software is built around ROS and is intended to run on a companion computer, interfacing with a Pixhawk (PX4-Autopilot). The flight stack outputs total

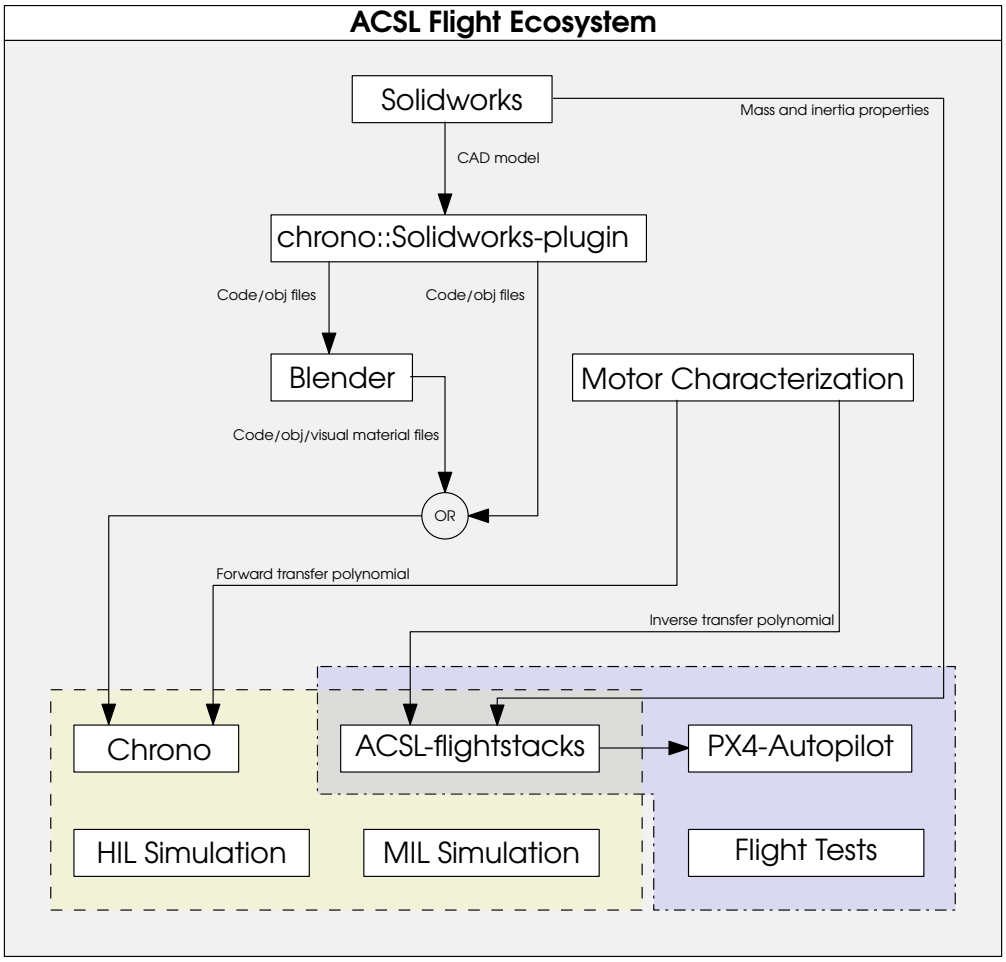

Fig. 1 Schematic representation of the the key elements of the proposed flight stack ecosystem. The same flight stacks ([1] for multi-rotor UAVs and [2] for fixed-wing UAVs) can be employed for model-in-the-loop (MIL) simulations, hardware-in-the-loop (HIL) simulations, or flight tests. The distributed computer-aided design (CAD) models of UAVs can be employed to perform numerical simulations or deduce key properties of the UAV to be used to tune the control systems. A package of Matlab® codes allows post-processing of flight and simulation data

thrust and desired body rates to the embedded flight controller, relying on the controller's inner loop, thus preventing direct testing of custom inner-loop algorithms. The software includes a state estimator that handles the estimation of translational states, while the embedded flight controller is used for estimating rotational states. This division is necessary because the companion computer does not have access to IMU measurements at full rate without delay. The source code for this flight stack is available at [16], and the documentation can be found at [17].

The flight stack presented in [18] is an open-source, multi-purpose software framework designed for developing autonomous multi-robot unmanned aerial systems. It assists developers in designing and constructing the control architecture of aerial robotic systems by integrating various heterogeneous computational solutions, such as computer vision algorithms, motion controllers, self-localization and mapping methods, and motion planning algorithms. The source code for this flight stack is available at [19] and detailed documentation is provided at [20].

Table 1 provides a synthetic comparison of the surveyed flight stacks and those proposed in [1, 5]. As discussed in Section VI below, the quality of the proposed simulator is considerably higher than the quality of the simulator associated to the surveyed flight stacks. Furthermore, only the flight stack [13] and the proposed flight stacks allow ROS2, which guarantees state-of-the-art fast communication among modules.

Project	Simulator	Simulator physics engine	Simulator aerodynamics	Direct actuator control	Documentation available	Post-Processing flight data	Vision Sensor Fusion	ROS2 support
ACSL Flight Ecosystem	Yes	Chrono	JSBSim	Yes	Yes	Yes	MOCAP/ VIO- Intel T265	Yes
CNS Flight Stack [10]	No	N/A	N/A	No	Yes	Yes	Custom VIO Laser Range Finder UWB positioning	No
PX4 Tilting Custom Control [13]	Yes	Gazebo	Gazebo	Yes	Yes	Yes	Intel T265 Intel D435i	Yes
MRS UAV System [15]	Yes	Gazebo MRS Multirotor Sim Unreal Engine CoppeliaSim	Gazebo	No	Yes	Yes	OpenVINS VIO Hector SLAM ALOAM SLAM	No
Aerostack [18]	Yes	Gazebo	Gazebo	No	Yes	Yes	Yes	No

Table 1 Comparison between UAV control-oriented projects

III. Hardware and Communication Architectures for the Proposed Flight Stacks

In MIL simulation mode, the proposed flight stacks [1, 2] are assumed to be executed on the same PC executing the Chrono-based simulator. In HIL and flight test modes, the proposed flight stacks [1, 2] are assumed to be executed on a Linux-based companion board computer, such as an ODroid® M1S, connected to a flight control unit (FCU), such as a PixHawk® 6c, which is used to estimate the UAV state and coordinate the propellers. The flight controller is assumed to execute the PX4 firmware.

The PX4 firmware utilizes a topic-based publish-subscribe framework known as *uORB*, which employs internal message definitions for communication between various modules. This system allows users to declare custom uORB definitions for internal use within the FCU and enables the exportation of all uORB topics for integration into a ROS2 package through the micro extreme resource constrained environment (Mircro-XRCE) DDS communication protocol. While PX4 can still receive traditional MAVLink messages from the companion computer via a serial connection, these messages must be converted to uORB messages to interface properly with critical internal functions, resulting in additional computational overhead that can slow down the communication link. Furthermore, customizing the MAVLink protocol requires significant effort from the user. In contrast, the current framework allows for direct publishing and subscribing to uORB topics, offering a much faster communication method compared to the previous generation of MAVLink protocol. This advancement also affords users greater control over the actuators, making redundant the previously accepted control inputs such as thrust and moment set points, rate set points, and position and orientation set points, although these control inputs continue to be supported within the PX4-Autopilot firmware.

In the proposed flight stacks [1, 2], the communication between the companion computer and the FCU are managed by ROS2. The two predominant frameworks currently employed in autonomous UAVs are the ROS and its successor, ROS2. The internal communication paradigms of these two frameworks differ significantly. ROS operates on a custom messaging queuing protocol analogous to AMQP (Advanced Message Queuing Protocol). This framework consists of two primary node types, namely ROS nodes and a ROS Master. A core element known as the *roscore* facilitates communication among nodes by enabling the publication and subscription of data. The ROS Master functions as a centralized hub within the ROS ecosystem, maintaining a registry of active nodes, services, and topics to streamline inter-node communication, specifically through discovery registration. While multiple ROS nodes can operate concurrently and register to a roscore, native support for cooperation between multiple roscorers is not incorporated within the ROS framework. In contrast, ROS2 adopts a decentralized architecture grounded in DDS (Data Distribution Service), thereby eliminating the necessity for a ROS Master and automating the discovery process. In this framework, ROS2 nodes possess the capability to transmit and receive data independent of central coordination, resulting in reduced latency when compared to ROS. Moreover, ROS2 facilitates various node types—including regular, real-time, intraprocess, composable, and lifecycle nodes, thereby enabling collaboration across different system layers, contingent upon the spatial alignment of ROS2 nodes within the same DDS domain.

Communication with the FCU is achieved via an FTDI cable (USB to UART), while the Micro-XRCE client, executed on the companion computer, is responsible for discovering and registering the uORB topics. Thus, the exchange of data through subscription and publication between the companion computer and the FCU is facilitated.

The DDS for eXtremely Resource-Constrained Environments (MicroXRCE-DDS) is an agent-client based communication protocol designed to run in devices with low computational resources, such as the microcontrollers used in

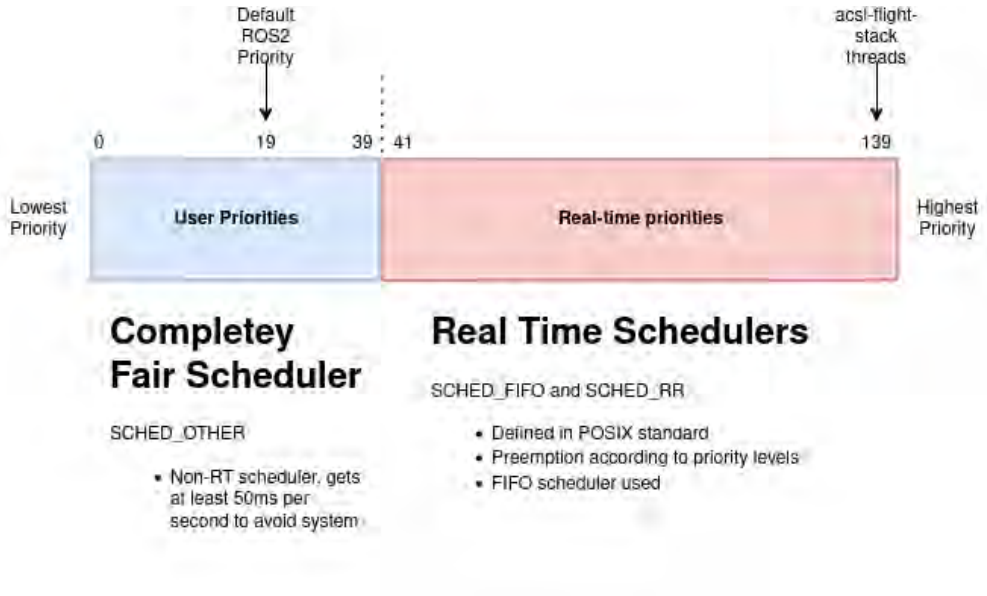

Fig. 2 Schematic representation of the Linux scheduler priority

FCUs. This protocol enables integration with a DDS network necessary for communication with a ROS2 network while ensuring real-time and reliable data distribution capabilities (RTPS). In this framework, more powerful agent nodes, such as the ROS2 nodes operating on the companion computer, serve as a bridge to connect resource-constrained client nodes, such as the uORB topics, to the DDS data space. The proposed flight stacks ecosystem leverages this communication protocol as the middleware layer for interaction between the FCU and the companion computer, significantly enhancing the speed of communication links and minimizing the risk of data package loss. It is noteworthy that, to the authors' knowledge, the proposed flight stack ecosystem is the first to employ the ROS2 framework.

IV. Software Architecture for the Proposed Flight Stacks

The flight stack for multi-rotor UAVs [1] is an open-source codebase specifically designed for companion computers, offering a reliable and high-performance platform for the control of multirotor UAVs. Developed in C++, it leverages the advantages of ROS2's multithreaded executor to run processes concurrently, ensuring a stable, rapid, and predictable system. The multithreaded executor node facilitates the concurrent publishing and subscribing of data between the FCU and the system. A provided bash script file prioritizes tasks in the Linux scheduler, effectively reducing jitter and minimizing the risk of data packet loss in the DDS network. A notable difference between this codebase and the flight stack for fixed-wing UAVs [2] is that the executed process, which is comprised of parallel tasks/nodes, is executed at the highest priority as shown in Figure 2 without custom core pinning, relying instead on the ROS2 scheduler to manage the allocation of CPU cores and threads to each task. By forgoing the callback-level executor, compatibility with older versions of ROS2 is enhanced. This codebase supports ROS2 Foxy and later versions and is closely integrated with unified time tracking and logging functionalities. Additionally, it can independently control the thrust of up to eight motors simultaneously and has been tested on a X8-copter shown in Figure 3.

The flight stack for fixed-wing UAVs [2] is an open-source flight control software designed for the companion computer, aimed at delivering high-performance and reliable control of fixed-wing and VTOl capable UAVs. Written in C++, it harnesses the advantages of the callback-group-level (CBG) executor [21, 22] to manage ROS2 nodes effectively, minimizing jitter and ensuring stable and predictable system behavior, compared to the already competent multithreaded executor utilized in [1] that comes standard in the ROS2 ecosystem. The software employs custom process threading by utilizing the Linux scheduler [23], specifically the real-time POSIX-based `sched_fifo` policy in code described in Figure 2, to prioritize critical processes, guaranteeing prompt and reliable execution. Additionally, user-defined core pinning for each individual process thread optimizes performance by enabling critical functions—such as communication with the FCU and the vision fusion methods employed by the MoCap or the VIO system to operate on dedicated CPU cores concurrently, thereby further reducing latency. Modularity stands as a key objective in the

Fig. 3 X8-copter UAV employed to test the flight stack [1] and the accessory files [4]

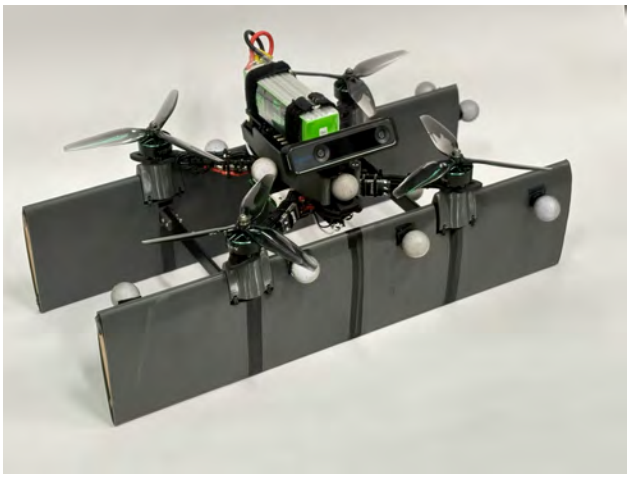

Fig. 4 Quad-biplane UAV employed to test the flight stack [2] and the accessory files [5]

development of this flight stack, allowing for the seamless integration of various pre-existing software solutions for tasks such as path-planning, trajectory-planning, and mapping, among others. Similar to [1], [2] supports individual thrust control for up to 8 motors. A unified time and logging mechanism ensures precise and synchronized timestamps for all operations, essential for debugging, visualization, and performance analysis. An additional distinguishing feature of [2] is that it incorporates aerodynamic models for the NACA-0012 rectangular wing utilized by the quad-biplane shown in Figure 4 to generate lift. These models, employed in are obtained by interpolating experimental data on the lift, drag, and moment coefficients measured for angles of attack over the interval $[-180, 180]$ degrees with a 1 degree precision [24]. Due to the inclusion of the CBG executor, the flight stack [2] is only compatible with ROS2 Galactic and higher version.

V. Model and Hardware-in-the-Loop Simulations

The proposed software ecosystem allows high-fidelity simulations [3] of the flight stacks to be employed on actual UAVs [1, 2]. One of the scopes of using a simulator to test a control system for UAVs is debugging the control algorithm in the early stage of the development process and find possible defects in the embedded code before the flight testing phase. An additional reason for employing reliable simulators lies in assessing *a priori* the performance of control

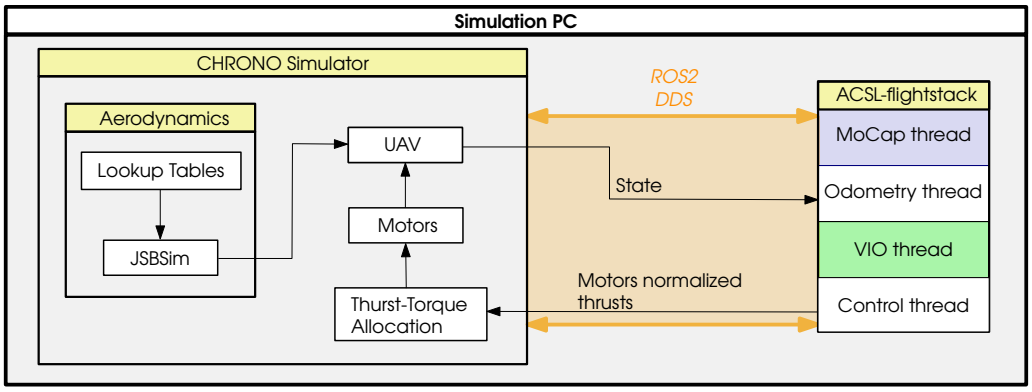

Fig. 5 Model-in-the-loop (MIL) architecture employed in [3]. Both the flight stack and the Chrono-based simulator are executed on the same PC

systems. Finally, high-fidelity simulators can be employed to simplify the tuning process before actual flight tests. Together with having a reliable simulator, it is also important to have a systematic approach to the problem of testing a control system for UAVs. The concept is to gradually increase the complexity of the setup, in order to target a different set of challenges in every distinct stage of the process, as described in [25]. To this goal, the open-source software available at [3] enables both MIL and HIL simulations. As discussed in the following, the scope of two simulation modalities is to address challenges that differ in both nature and complexity. In the following, we refer to X-in-the-loop as any of these simulation techniques.

A. Model-in-the-loop simulations

MIL testing comprises three essential elements, namely the *model*, the *simulated environment*, and the *software tool*. The model fully represents the system's behavior within the limits of the chosen environment. The environment describes a realistic setting in which the system would operate, and it is used to test different scenarios, as well as to generate inputs or to control external agents. The software tool is what binds the model and the environment, and it allows the collection of data and the analysis of the behavior of the model. The main goal of the MIL process is to check the logic's validity of the control algorithm in multiple test cases. Once the results from this stage are considered adequate, they are deemed the benchmark for any subsequent phase in the X-in-the-loop ladder. One of the reasons why the outcome of this step is taken as the reference for any future tests is that all the computations are performed with high precision with floating-point arithmetic, which is not the case for all the other stages as well as in the real final implementation. Furthermore, the underlying dynamical models are quite idealistic. Figure 5 shows a schematic representation of how MIL simulations are performed applying the proposed software [3].

B. Hardware-in-the-loop simulations

The purpose of HIL tests is to replicate as closely as possible the real operating conditions that the simulated system will encounter once in operation, without having to deal with the risks associated with the real world. The degree of realism of simulations is usually chosen as a function of the applications considered. For instance, in [26] and [27], the authors performed HIL tests to verify the attitude determination and control system of a small satellite. In this case, the authors tested not only the controller being executed in real-time on the onboard companion computer but also the rest of the hardware, including sensors and actuators. Figure 6 shows a schematic representation of how HIL simulations are performed applying the proposed software [3]. To our knowledge, no such complex HIL setup has been developed for UAVs.

C. Aerodynamic modeling

In MIL and HIL simulations, aerodynamic forces and moments are simulated by employing JSBSim® [9]. JSBSim® is an open-source, high-fidelity flight dynamics model (FDM) software that is widely utilized in aerospace research and simulation. Developed in C++, it is lightweight, cross-platform, and designed for real-time operation, making it suitable for both standalone applications and integration within larger simulation frameworks such as Chrono. JSBSim

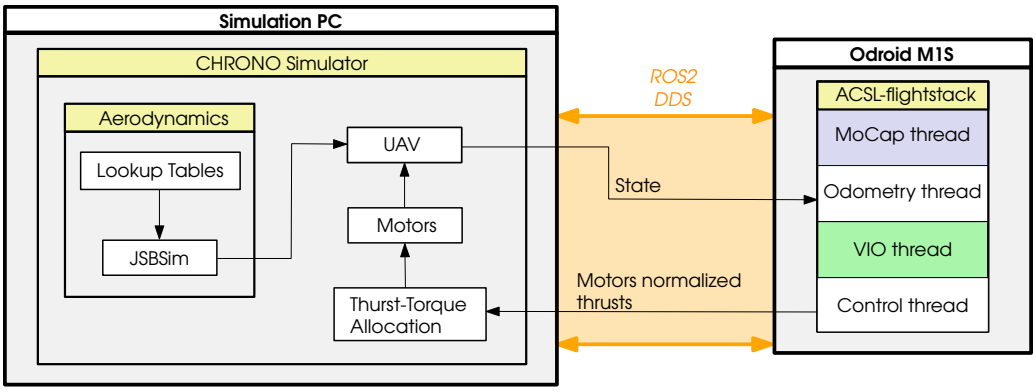

Fig. 6 Hardware-in-the-loop (HIL) architecture employed in [3]. The flight stack is executed on a companion computer connected to a PC executing the Chrono-based simulator

employs physics-based modeling to accurately simulate the rigid-body dynamics of aircraft, UAVs, and spacecraft, accommodating subsonic, supersonic, and orbital scenarios. Its modular architecture and extensive library of predefined aircraft models enable users to easily customize or create their own configurations via XML files. By combining accuracy, performance, and scalability, JSBSim serves as a vital tool for the development and analysis of advanced aerospace systems. Distinct from many other FDM packages, JSBSim utilizes look-up tables derived from actual wind tunnel data to model aerodynamic properties. This method ensures high computational efficiency without sacrificing accuracy, enabling simulations that are both remarkably fast and grounded in empirical validation. The CFD (computational fluid dynamics) modules of Chrono could be employed to simulate aerodynamic forces and moments. However, in this case, the simulation time would be substantially higher than using JSBSim®.

D. Simulation result visualization

Matlab® plots of numerical simulations and flight tests can be produced by the codes distributed at [4] or [5] according to the flight stack used. In the case of numerical simulations, test results can be also visualized employing Blender® [28]. Blender® is a free and open-source 3D creation suite. In the proposed ecosystem, its primary purpose is to modify the models exported by the chorno-solidworks-plugin, thereby enhancing the visual fidelity of simulations conducted in Chorno.

VI. Physics Engine for Moldel-in-the-Loop and Hardware-in-the-Loop Simulations

Model-in-the-loop and hardware-in-the-loop simulations are supported by Chrono [7, 8], which serves as the physics engine. In this section, we discuss the key features of Chrono, the state of the art of high-fidelity simulations for UAVs, and the advantages of employing Chrono over existing simulators.

A. Project Chrono

A key component of the proposed architecture lies in the underlying simulator Chrono [7, 8], which serves as a physics engine. This software package supports rigid and flexible body dynamics, such as nonlinear finite element analysis (FEA), friction and contact handling, deformable terrain simulation, modules for fluid-solid interaction based on Navier–Stokes equations with smoothed-particle hydrodynamics (SPH), and granular dynamics [29]. An example of a simulation in which many of these features played a role is shown in Figure 7. The main modules of the Chrono simulation environment employed in the proposed flight stacks [1, 2] are summarized in Table 2. To the author's knowledge, Chrono has been utilized by researchers for simulating multi-copters or other aerial vehicles on in [30–32], where advanced MRAC systems are tested on an X8-copter, whose CAD model closely mimics an actual UAV, and [33], where no autonomous control system is employed.

Chrono is almost entirely written in C++, and a wrapped Python version of it, namely PyChrono [35], is available. An advantage of PyChrono is that there are numerous libraries already available in Python that do not necessarily have a counterpart in C++. On the other hand, however, PyChrono does not cover the same range of functionalities as Chrono, such as, for example, the fluid-solid interaction module.

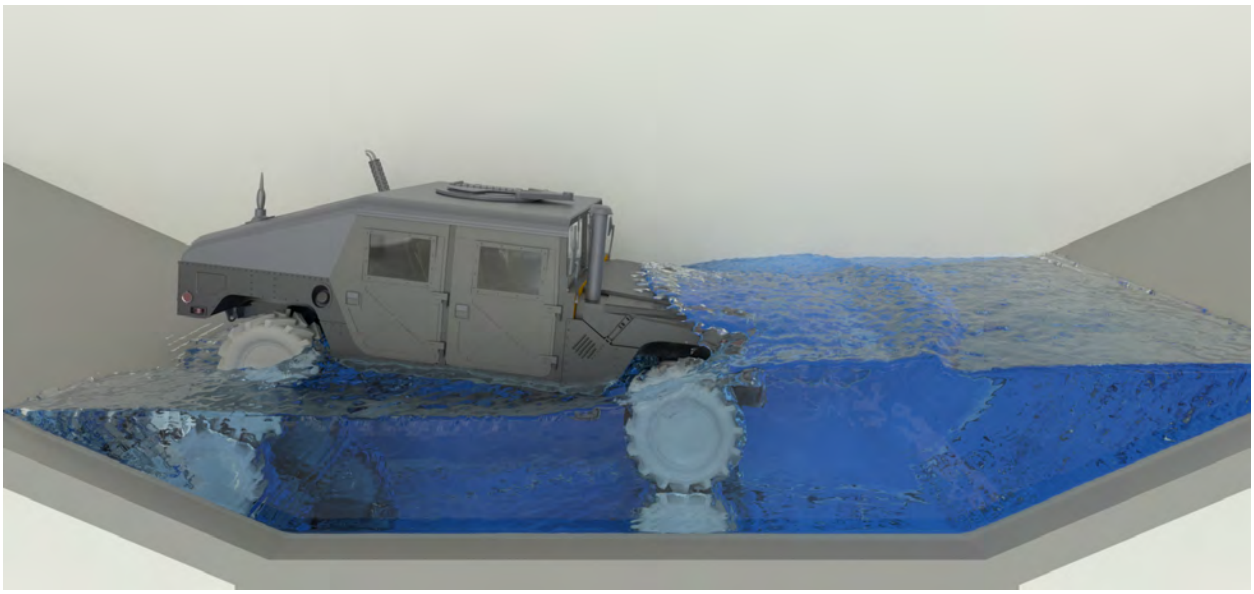

Fig. 7 Vehicle fording simulated using Chrono [34]. In this example, many features of Chrono are displayed, such as rigid body dynamics to simulate the interaction between the various components of the vehicle, flexible body dynamics to simulate the deformation of the tires, and SPH (smoothed-particle hydrodynamics) to simulate the interaction between the fluid and the vehicle body

In recent years, the multi-rotor UAV control community has oriented itself toward complex applications, wherein the UAV is tasked with transporting fluids, interacting with hard and soft surfaces, interacting with humans and other robots, and pushing or pulling heavy unsteady loads. For these reasons, a physics engine able to simulate a wide range of applications, such as Chrono, is essential. In the following, we survey a selection of very challenging applications in which Chrono has been employed with success. Although none of these applications is related to aerial robotics, they all give a sense of the potential of this tool. For instance, researchers have successfully employed Chrono in the field of autonomous ground vehicles as a development tool for validating control algorithms, as demonstrated in [36]. Chrono, as shown in [37], has also been employed to simulate multi-vehicle autonomous systems, whose interactions were enabled by the module SYNCHRONO, which uses a single centralized hub to synchronize all agents' states. In [38], Chrono has been used in conjunction with the SPH-oriented software DualSPHysics [39] to simulate the dynamic fluid-solid interaction between an incompressible fluid and a deformable structure, obtaining a remarkably close match between experimental and numerical data. In [40], the authors employ PyChrono to develop a virtual three-dimensional rotor test rig for balancing experiments.

Worthy of mention is how Chrono has been employed for machine learning and artificial intelligence training purposes. For instance, in [41], the authors trained a 14-degrees-of-freedom four-legged robot to navigate in harsh, hostile environments via DRL (deep reinforcement learning). A similar application was presented in [42], where the authors employed PyChrono to train a 7-link biped robot to walk on deformable terrain using DRL. In the study, the authors compared the results of direct shear simulations employing soft and hard contacts. The work presented in [43] merges the aforementioned frameworks of artificial intelligence for autonomous agents and deformable terrain. In this study, the authors investigate the training and assessment of DLR policies for autonomous vehicles driving in an off-road, obstacle-rich environment.

B. State-of-the-art in simulators for UAVs

The desired features of a simulator able to support both the testing and the tuning of control systems for UAVs are many. Firstly, this simulator should closely capture the dynamics of the vehicle and its payload. Additionally, this simulator should be able to capture a variety of effects, ranging from contact forces and moments, the effect of deformable components, or the presence of fluid payloads, with a reasonably small effort on the user's side. Furthermore, this simulator should account for detailed geometric and inertial properties of the vehicle to be imported, for instance, from CAD models. High-quality rendering would also be appreciated to dissect complex phenomena. Finally, this

Chrono Modules in the Proposed Flight Stacks [1, 2]	
Module	**Description**
Chrono::Engine	Core simulation functionality.
CASCADE	Import CAD designs into simulations.
FEA	Finite element analysis tools.
FSI	Fluid-solid interaction simulation.
GPU	GPU-based parallel simulation.
IRRLICHT	3D real-time simulation visualization.
MULTICORE	Multi-core parallel simulation.
OpenGL	3D real-time visualization module.
POSTPROCESS	Postprocessing and data rendering tools using POVray and Blender.
PYTHON	Scripting and integration with Python.
ROS2	communication module
SENSOR	Virtual sensor modeling and simulation.
SOLIDWORKS	Import SolidWorks CAD models.
VEHICLE	Vehicle dynamics modelling and simulation.

Table 2 Main Chrono modules used in the proposed flight stacks [1, 2]

simulator should be open-source to allow the user to customize some of its components.

The state-of-the-art in the area of simulators for aerial robotics applications is not particularly vast and is substantially captured by AirSim, FlightGear, RotorS, Flightmare, and FlightGoggles. AirSim [44] is a platform developed by Microsoft based on the physics engine PhysX [45] and on the rendering engine Unreal engine [46]. This software package shows exceptional rendering capabilities and its strength relies on the possibility of generating highly reliable on-board camera data to be used for deep learning, computer vision, and reinforcement learning algorithms. However, being based on PhysX, its dynamical model is not able to describe the UAV behavior accurately since, for example, it does not take into account the Coriolis force; see [47]. Therefore, while extremely powerful for vision-based navigation purposes, AirSim is unsuitable for control systems testing and tuning, especially in complex scenarios.

FlightGear [48] is an advanced flight simulator developed mainly for fixed-wing aircraft. One of the underlying flight dynamics models is JSBSim [49], which lays its foundations on data from look-up tables produced by means of experimental data. YASim [50] is another flight dynamics model that can be used in FlightGear, it does not consider experimental data, but it exploits the geometric and inertia characteristics of the aircraft to compute the forces and moments acting on it. Another option is based on the LaRCsim [51] flight dynamics model (originally developed by NASA), which was designed to analyze the effect of icing on aircraft. Without a doubt, FlightGear provides a good simulator for aircraft, but it lacks important features like the possibility to take into account contacts. Additionally, modeling some parts of the vehicle as deformable, such as sling ropes, is not yet possible.

RotorS [52] is a micro-aerial vehicle simulation framework based on Gazebo physics engine. This simulator has been explicitly designed to enable testing of estimators and controllers for multi-rotor UAVs and produce results that are sufficiently close to those that could be obtained from actual experiments. Some examples of where RotorS was used with success are presented in [53, 54]. However, this simulator is affected by some key limitations. For example, its dynamics model is based on Gazebo [55] and, hence, it does not allow to take into account deformable objects and fluid-solid interaction.

Flightmare [56] together with FlightGoggles [57] appear to be the two most advanced simulators developed specifically for multi-rotor UAVs. The high fidelity of these two simulators is assured by the fact that they require the collection of data on the UAV using a motion capture system and the IMU (inertial measurement unit) aboard the aircraft to emulate. Thus, Flightmare and FlightGoggles are not suitable tools for testing a controller's performance before any experimental attempt, but they are more suitable for those applications where the process to be assessed is based mainly on exteroceptive sensors that will be simulated in a virtual environment, such as cameras, RGB (red green blue) cameras, infrared beacon sensors, and range sensors. Thus, these two simulators let the user easily change the virtual environment and the exteroceptive sensor parameters, allowing to quickly evaluate visual-inertial odometry

algorithms on many different scenarios without an actual risk of collision

An exhaustive example of how FlightGoggles has been used is shown in [58]. In this case, the researchers developed and assessed a visual-inertial odometry estimation and control algorithm for a micro aerial vehicle whose goal is to fly through a physical window. Since performing this kind of missions in the real world is prone to numerous crashes, at least in the first stages of the project, the authors simulated a virtual window by creating synthetic camera images that are sent in real-time to the UAV, which is flying in a safe environment provided with a motion capture system. In [59], the authors employ Flightmare to develop a high-velocity collision avoidance algorithm that allows the UAV to fly through challenging environments like dense forests, snow-covered terrain, and collapsed buildings. The authors showed how the simulator can be used to perform the totality of the developing and tuning process, achieving zero-shot from simulation to the real world. However, FlightGoggles suffers from the same limitations as Flightmare, which is that if they are not used with the vehicle-in-the-loop configuration, then they do not provide much more than the other simulators already described above. Additionally, the two simulators have limited documentation and few resources are shared with the community.

C. Chrono vs state-of-the-art UAV simulators

Having presented the promising features of Chrono and PyChrono discussed in Section VI.A, in the following, we compare this software package to the most promising simulators presented in Section VI.B. In particular, we compare Chrono to Flightmare and FlightGoggles.

The proposed comparative analysis excludes AirSim because, despite being renowned for its exceptional rendering capabilities, the underlying dynamical models are overly simplistic. Indeed, AirSim relies on a physics engine primarily designed for video game applications, prioritizing speed over accuracy. To ensure the success of our research objectives, it was imperative to find a simulator that not only demonstrated good rendering capabilities but also met the stringent criteria for dynamic accuracy relevant to our study. Our comparative analysis excludes FlightGear as well. Indeed, FlightGear was primarily developed with a focus on fixed-wing aircraft, whereas the proposed system is primarily focused on multi-rotor UAVs such as quadcopters. Additionally, FlightGear lacks of tools to effectively model multi-body interactions, such as impacts, which posed a limitation to testing advanced control techniques in complex scenarios. Finally, we exclude RotorS since it can not account for contacts between objects within the simulated environment and to model deformable bodies.

Table 3 summarizes key differences between Chrono, Flightmare, and FlightGoggles. Chrono certainly has lower rendering capabilities compared to the other two simulators since they use the high-quality rendering engine Unity, initially developed for the gaming industry. However, although there is not yet a ready-to-use module, Chrono's versatility does not preclude it to be interfaced to rendering engines such as Unity or Unreal Engine. Furthermore, only Flightmare and FlightGoggles allow the user to use real-world dynamics while performing vehicle-in-the-loop simulations. Chrono is the only simulator that lets the user consider deformable bodies and fluid-solid interactions. While all three simulators can detect if a collision has occurred, Chrono is the only simulator that can model contacts between bodies. Chrono is also the only platform that can be used to simulate a sling payload, thanks to the cable FEM element. Flightmare and FlightGoggles also do not have a direct method to import a CAD model into the simulation environment, while Chrono has a plug-in developed specifically for Solidworks that lets the user transfer the geometry into the simulation environment in a few clicks. Flightmare and FlightGoggles are certainly UAV-oriented, while Chrono is not. On the other hand, however, Chrono is the only simulator among those that has satisfactory documentation available and an active community, and its applicability to the control of multi-rotor UAVs has been proven [30–32].

Considering these pros and cons, Chrono has been chosen as the simulator to be employed in this research, despite not being UAV-oriented. This implies though that we would pay the price of having to set up an environment inside Chrono developed specifically for simulating UAV. Nevertheless, for its versatility Chrono let us simulate a vast range of experiments that would have been otherwise very difficult to simulate with any other tool.

VII. Conclusion

This paper introduced a freeware, open-source software ecosystem to design, test, and tune control systems for autonomous VTOL UAVs. The components of this ecosystem are a flight stack for multi-rotor UAVs [1], a flight stack for fixed-wing UAVs [2], a high-fidelity simulator for MIL and HIL texts [3], Matlab® codes for data visualization [4, 5], CAD models of UAVs [4, 5], and a website [6].

Future work directions include improving each of the components of this ecosystem. For instance, each of the control system designed and tested by the authors will be uploaded on [1, 2]. The documentation on [6] will be updated

and expanded periodically. Additional CAD models will be provided at [4, 5]. Furthermore, the aerospace control community is strongly encouraged to contributing to the development of this shared software platform.

Acknowledgments

This work has been partly supported funded by the National Science Foundation under Grant no. 2137159, and the US Army Research Lab under Grant no. W911QX2320001. This publication is partially funded from the MUR – DM 118/2023, as part of the project PNRR-NGEU.

Simulator	Open source	Rendering	Dynamics	Multiple vehicles	FEM	FSI	Contact	Collision	Sling payload	CAD import	UAV oriented	Documentation available	Active community
Chrono	Yes	Irrlicht/ POVray	Chrono:: engine	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Flightmare	Yes	Unity	Gazebo-based/ Real-world dynamics	Yes	No	No	No	Yes	No	No	Yes	No	No
FlightGoggles	Yes	Unity	Classical dynamics/ Real-world dynamics	Yes	No	No	No	Yes	No	No	Yes	No	No

Table 3 Comparison between Chrono, Flightmare, and FlightGoggles. From the comparison, it emerges the versatility of Chrono, and even though not UAV oriented it still comprises crucial features for our work that we could not find in any other simulator

References

[1] Gramuglia, M., Kumar, G. M., and L'Afflitto, A., "A freeware, open-source flight stack for multi-rotor UAVs," https://github.com/andrealaffly/ACSL-flightstack, 2024. Accessed: 2024-12-02.

[2] Kumar, G. M., Gramuglia, M., and L'Afflitto, A., "A freeware, open-source flight stack for fixed-wing UAVs," https://github.com/andrealaffly/ACSL-flightstack-winged, 2024. Accessed: 2024-12-02.

[3] Kumar, G. M., Gramuglia, M., and L'Afflitto, A., "A freeware, open-source Chrono-based simulator for UAVs," https://github.com/andrealaffly/acsl-chrono-simulator, 2024. Accessed: 2024-12-02.

[4] Gramuglia, M., Kumar, G. M., and L'Afflitto, A., "Supporting material for "A freeware, open-source flight stack for multi-rotor UAVs"," https://github.com/andrealaffly/ACSL-flightstack-accessories, 2024. Accessed: 2024-12-02.

[5] Kumar, G. M., Gramuglia, M., and L'Afflitto, A., "Supporting material for "A freeware, open-source flight stack for fixed-wing UAVs"," https://github.com/andrealaffly/ACSL-flightstack-winged-accessories, 2024. Accessed: 2024-12-02.

[6] Melendez, E., Gramuglia, M., Kumar, G. M., and L'Afflitto, A., "Website for An Open-Source Framework to Design, Tune, and Fly Nonlinear Control Systems for Autonomous UAVs," https://acslstacks.com, 2024. Accessed: 2024-12-02.

[7] Tasora, A., Serban, R., Mazhar, H., Pazouki, A., Melanz, D., Fleischmann, J., Taylor, M., Sugiyama, H., and Negrut, D., "Chrono: An open source multi-physics dynamics engine," *High Performance Computing in Science and Engineering – Lecture Notes in Computer Science*, edited by T. Kozubek, Springer, 2016, pp. 19–49.

[8] Project Chrono, "Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems," https://projectchrono.org/, 2014. Accessed: 2023-10-14.

[9] Brendt, J., Peden, T., Culp, D., De Marco, A., Duke, L., Froehlich, M., Megginson, D., Hofman, E., and Gidenstam, A., "JSBSim: An open source, platform-independent, flight dynamics & control software library in C++," https://jsbsim.sourceforge.net/documentation.html, 2024. Accessed: 2024-12-02.

[10] Scheiber, M., Fornasier, A., Jung, R., Böhm, C., Dhakate, R., Stewart, C., Steinbrener, J., Weiss, S., and Brommer, C., "CNS Flight Stack for Reproducible, Customizable, and Fully Autonomous Applications," *IEEE Robotics and Automation Letters*, Vol. 7, No. 4, 2022, pp. 11283–11290. https://doi.org/10.1109/LRA.2022.3196117.

[11] Control of Networked Systems Group at University of Klagenfurt, "CNS Flight Stack," , 2022. URL https://github.com/aau-cns/flight_stack.

[12] dotX Automation s.r.l., "PX4-based software stack for autonomous flight," , 2023. URL https://github.com/dotX-Automation/flight_stack.

[13] D'Angelo, S., Pagano, F., Longobardi, F., Ruggiero, F., and Lippiello, V., "Efficient Development of Model-Based Controllers in PX4 Firmware: A Template-Based Customization Approach," *International Conference on Unmanned Aircraft Systems*, 2024, pp. 1155–1162. https://doi.org/10.1109/ICUAS60882.2024.10556938.

[14] PRISMA Lab at University of Naples Federico II, "Px4_Tilting_Custom_Control," , 2024. URL https://github.com/prisma-lab/Px4_Tilting_Custom_Control.

[15] Baca, T., Petrlik, M., Vrba, M., Spurny, V., Penicka, R., Hert, D., and Saska, M., "The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles," *Journal of Intelligent & Robotic Systems*, Vol. 102, No. 1, 2021, p. 26. https://doi.org/10.1007/s10846-021-01383-5.

[16] Multi-robot Systems (MRS) group at Czech Technical University in Prague, "Multi-robot Systems Group UAV system," , 2021. URL https://github.com/ctu-mrs/mrs_uav_system.

[17] Multi-robot Systems (MRS) group at Czech Technical University in Prague, "Multi-robot Systems Group UAV system, Documentation website," , 2021. URL https://ctu-mrs.github.io/.

[18] Sanchez-Lopez, J. L., Suárez Fernández, R. A., Bavle, H., Sampedro, C., Molina, M., Pestana, J., and Campoy, P., "AEROSTACK: An architecture and open-source software framework for aerial robotics," *International Conference on Unmanned Aircraft Systems*, 2016, pp. 332–341. https://doi.org/10.1109/ICUAS.2016.7502591.

[19] Computer Vision and Aerial Robotics (CVAR) group at Universidad Politecnica de Madrid, "Aerostack, a software framework for aerial robotics," , 2016. URL https://github.com/cvar-upm/aerostack.

[20] Computer Vision and Aerial Robotics (CVAR) group at Universidad Politecnica de Madrid, "Aerostack Documentation," , 2016. URL https://github.com/aerostack/install/wiki.

[21] OFERA Consortium, "Real-time Executor Software Release (Year 3)," Tech. rep., OFERA, 2021. URL https://www.ofera.eu/storage/deliverables/M36/OFERA_42_D46_Real-time_executor_software_release_Y3.pdf, accessed: 2024-12-02.

[22] Yang, Y., and Azumi, T., "Exploring Real-Time Executor on ROS 2," *IEEE International Conference on Embedded Software and Systems*, 2020, pp. 1–8. https://doi.org/10.1109/ICESS49830.2020.9301530.

[23] Lange, R., "Callback-group-level Executor," https://www.apex.ai/_files/ugd/984e93_f3791ae0711042a883bfc40f827d6268.pdf, 2021. Accessed: 2024-12-02.

[24] Sheldahl, R. E., and Klimas, P. C., "Aerodynamic characteristics of seven symmetrical airfoil sections through 180-degree angle of attack for use in aerodynamic analysis of vertical axis wind turbines," 1981. https://doi.org/10.2172/6548367.

[25] Shokry, H., and Hinchey, M., "Model-Based Verification of Embedded Software," *Computer*, Vol. 42, No. 4, 2009, pp. 53–59. https://doi.org/10.1109/MC.2009.125.

[26] Kiesbye, J., Messmann, D., Preisinger, M., Reina, G., Nagy, D., Schummer, F., Mostad, M., Kale, T., and Langer, M., "Hardware-In-The-Loop and Software-In-The-Loop Testing of the MOVE-II CubeSat," *Aerospace*, Vol. 6, No. 12, 2019. https://doi.org/10.3390/aerospace6120130.

[27] Tavakoli, A., Faghihinia, A., and Kalhor, A., "An innovative test bed for verification of attitude control system," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 32, No. 6, 2017, pp. 16–22. https://doi.org/10.1109/MAES.2017.150198.

[28] "Blender Official Website," https://www.blender.org/, 2024. Accessed: 2024-12-02.

[29] Liu, C. K., and Negrut, D., "The Role of Physics-Based Simulators in Robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, Vol. 4, No. 1, 2021, pp. 35–58. https://doi.org/10.1146/annurev-control-072220-093055.

[30] Gramuglia, M., Kumar, G. M., and L'Afflitto, A., "A Hybrid Model Reference Adaptive Control System for Multi-Rotor Unmanned Aerial Vehicles," *SciTech*, AIAA, Orlando, FL, 2024, pp. 1–21. https://doi.org/10.2514/6.2024-0755.

[31] Gramuglia, M., Kumar, G. M., and L'Afflitto, A., "Adaptive control for hybrid dynamical systems with user-defined rate of convergence," *Journal of the Franklin Institute*, Vol. 361, No. 9, 2024, p. 106854. https://doi.org/https://doi.org/10.1016/j.jfranklin.2024.106854.

[32] Gramuglia, M., Kumar, G. M., and L'Afflitto, A., "Two-Layer Adaptive Funnel MRAC with Applications to the Control of Multi-Rotor UAVs," *International Workshop on Robot Motion and Control*, 2024, pp. 31–36. https://doi.org/10.1109/RoMoCo60539.2024.10604361.

[33] Project Chrono, "LittleHexy copter model," https://api.projectchrono.org/group__robot__models__copter.html, 2014. Accessed: 2023-10-14.

[34] Project Chrono, "Vehicle fording," https://projectchrono.org/assets/Images/humvee_fording.png, 2014. Accessed: 2023-10-14.

[35] Benatti, S., Young, A., Elmquist, A., Taves, J., Serban, R., Mangoni, D., Tasora, A., and Negrut, D., *PyChrono and gym-chrono: A Deep Reinforcement Learning Framework Leveraging Multibody Dynamics to Control Autonomous Vehicles and Robots*, Springer, 2022, pp. 573–584. https://doi.org/10.1007/978-3-030-81166-2_50.

[36] Elmquist, A., Young, A., Mahajan, I., Fahey, K., Dashora, A., Ashokkumar, S., Caldararu, S., Freire, V., Xu, X., Serban, R., and Negrut, D., "A software toolkit and hardware platform for investigating and comparing robot autonomy algorithms in simulation and reality," 2022. https://doi.org/10.48550/arXiv.2206.06537.

[37] Taves, J., Elmquist, A., Young, A., Serban, R., and Negrut, D., "SynChrono: A Scalable, Physics-Based Simulation Platform For Testing Groups of Autonomous Vehicles and/or Robots," *International Conference on Intelligent Robots and Systems*, IEEE, 2020, pp. 2251–2256. https://doi.org/10.1109/IROS45743.2020.9341585.

[38] Capasso, S., Tagliafierro, B., Martínez-Estévez, I., Domínguez, J. M., Crespo, A. J. C., and Viccione, G., "A DEM approach for simulating flexible beam elements with the Project Chrono core module in DualSPHysics," *Computational Particle Mechanics*, Vol. 9, No. 5, 2022, pp. 969–985. https://doi.org/10.1007/s40571-021-00451-9.

[39] Domínguez, J. M., Fourtakas, G., Altomare, C., Canelas, R. B., Tafuni, A., García-Feal, O., Martínez-Estévez, I., Mokos, A., Vacondio, R., Crespo, A. J. C., Rogers, B. D., Stansby, P. K., and Gómez-Gesteira, M., "DualSPHysics: from fluid dynamics to multiphysics problems," *Computational Particle Mechanics*, Vol. 9, No. 5, 2022, pp. 867–895. https://doi.org/10.1007/s40571-021-00404-2.

[40] Uzcátegui, L. U. M., Pereda, C. M.-K., and Medina, E. C., "The Virtual Rotor Kit Project: A virtual rotor test rig for balancing experiments," *International Journal of Mechanical Engineering Education*, Vol. 0, No. 0, 2023. https://doi.org/10.1177/03064190231197119.

[41] Benatti, S., Tasora, A., and Mangoni, D., "Training a Four Legged Robot via Deep Reinforcement Learning and Multibody Simulation," *Multibody Dynamics*, edited by A. Kecskeméthy and F. Geu Flores, Springer, Cham, 2020, pp. 391–398.

[42] Bhardwaj, G., Dasgupta, S., Sukavanam, N., and Balasubramanian, R., "Soft Soil Gait Planning and Control for Biped Robot using Deep Deterministic Policy Gradient Approach," *ArXiv*, Vol. abs/2306.08063, 2023.

[43] Young, A., Taves, J., Elmquist, A., Benatti, S., Tasora, A., Serban, R., and Negrut, D., "Enabling Artificial Intelligence Studies in Off-Road Mobility Through Physics-Based Simulation of Multiagent Scenarios," *Journal of Computational and Nonlinear Dynamics*, Vol. 17, No. 5, 2022, p. 051001. https://doi.org/10.1115/1.4053321.

[44] Shah, S., Dey, D., Lovett, C., and Kapoor, A., "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," *Field and Service Robotics*, edited by M. Hutter and R. Siegwart, Springer, Cham, 2018, pp. 621–635.

[45] NVIDIA, "PhysX," https://nvidia-omniverse.github.io/PhysX/physx/5.2.1/index.html, 2023. Accessed: 2023-09-24.

[46] Epic Games, "Unreal Engine," https://www.unrealengine.com, 2019. Accessed: 2023-09-24.

[47] Erez, T., Tassa, Y., and Todorov, E., "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX," *IEEE International Conference on Robotics and Automation*, 2015, pp. 4397–4404. https://doi.org/10.1109/ICRA.2015.7139807.

[48] FlightGear developers & contributors, "FlightGear," https://www.flightgear.org/, 2021. Accessed: 2023-09-24.

[49] Berndt, J., "JSBSim: An Open Source Flight Dynamics Model," 2004. https://doi.org/10.2514/6.2004-4923.

[50] Ross, A., "YASim," https://wiki.flightgear.org/YASim, 2021. Accessed: 2023-09-24.

[51] NASA, "LaRCsim," https://ntrs.nasa.gov/citations/19950023906, 1995. Accessed: 2023-09-24.

[52] Furrer, F., Burri, M., Achtelik, M., and Siegwart, R., *RotorS—A Modular Gazebo MAV Simulator Framework*, Springer International Publishing, Cham, 2016, pp. 595–625. https://doi.org/10.1007/978-3-319-26054-9_23.

[53] Nisar, B., Foehn, P., Falanga, D., and Scaramuzza, D., "VIMO: Simultaneous Visual Inertial Model-Based Odometry and Force Estimation," *IEEE Robotics and Automation Letters*, Vol. 4, No. 3, 2019, pp. 2785–2792. https://doi.org/10.1109/LRA.2019.2918689.

[54] Loquercio, A., Kaufmann, E., Ranftl, R., Dosovitskiy, A., Koltun, V., and Scaramuzza, D., "Deep Drone Racing: From Simulation to Reality With Domain Randomization," *IEEE Transactions on Robotics*, Vol. 36, No. 1, 2020, pp. 1–14. https://doi.org/10.1109/TRO.2019.2942989.

[55] Koenig, N., and Howard, A., "Design and use paradigms for Gazebo, an open-source multi-robot simulator," *International Conference on Intelligent Robots and Systems*, Vol. 3, 2004, pp. 2149–2154. https://doi.org/10.1109/IROS.2004.1389727.

[56] Song, Y., Naji, S., Kaufmann, E., Loquercio, A., and Scaramuzza, D., "Flightmare: A Flexible Quadrotor Simulator," *Conference on Robot Learning*, 2021, pp. 1147–1157.

[57] Guerra, W., Tal, E., Murali, V., Ryou, G., and Karaman, S., "FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality," *International Conference on Intelligent Robots and Systems*, 2019, pp. 6941–6948. https://doi.org/10.1109/IROS40897.2019.8968116.

[58] Sayre-McCord, T., Guerra, W., Antonini, A., Arneberg, J., Brown, A., Cavalheiro, G., Fang, Y., Gorodetsky, A., McCoy, D., Quilter, S., Riether, F., Tal, E., Terzioglu, Y., Carlone, L., and Karaman, S., "Visual-Inertial Navigation Algorithm Development Using Photorealistic Camera Simulation in the Loop," *International Conference on Robotics and Automation*, 2018, pp. 2566–2573. https://doi.org/10.1109/ICRA.2018.8460692.

[59] Loquercio, A., Kaufmann, E., Ranftl, R., Müller, M., Koltun, V., and Scaramuzza, D., "Learning high-speed flight in the wild," *Science Robotics*, Vol. 6, No. 59, 2021, p. eabg5810. https://doi.org/10.1126/scirobotics.abg5810.