Kurdish Studies
October 2024
Volume: 12, No: 5, pp 1377 - 1387
ISSN: 2051-4883 (Print) | ISSN 2051-4891 (Online)
www.KurdishStudies.net

DOI: 10.53555/ks.v12i5.3482

Real Time Flight Simulator Data Communication For Co-Simulation

Muhammad Ibrar[1]*, Saif Ur Rehman[2], Hamid Gulzar[3], Tahir Zaidi[4], Tuba Ahsan[5], Zahid Gulzar[6] Muhammad Umar Nazir[7]

*[1]Department of Electrical Engineering, The Superior University, Lahore, Pakistan. su92-mseew-f22-005@superior.edu.pk
[2]Department of Electrical Engineering, The Superior University, Lahore, Pakistan.
[3] Company # Arab Technical Co Ltd., Babader Plaza, King Fahd Street, Jeddah, KSA.
[4]Department of Avionics Engineering, The Superior University, Lahore, Pakistan.
[5]Department of Electrical Engineering, The Superior University, Lahore, Pakistan.
[6]Company # Arab Technical Co Ltd., Babader Plaza, King Fahd Street, Jeddah, KSA.
[7]Department of Avionics Engineering, The Superior University, Lahore, Pakistan.

Abstract: Testing airplane models in a flight environment and their realistic virtual testing of control systems in 3D is gaining popularity In a way that prioritizes safety and cost savings approach for evaluation and analysis. The initial part of the article presents a broad perspective and comparison of the leading flight simulators for PCs that have been successfully linked with MathWorks software, examining their strengths and weaknesses. Co-simulation enables the integration of flight simulation software's advanced simulation and visualization capabilities with MATLAB toolboxes' expertise in navigation, control, and sensor modeling, creating a robust and comprehensive simulation environment. Following this approach, one can utilize flight data to identify and validate aircraft models, assess the performance of control algorithms, and evaluate the handling qualities of aircraft, enabling a more comprehensive understanding of flight dynamics. There is a significant absence of comprehensive, comprehensive manual for flying co-simulation, and surprisingly, no one has yet attempted to conduct a thorough comparison of various flight co-simulation methods in a single, unified effort. As a result, we have developed and share our own extensive Simulink co-simulation lessons, showcasing integrations Featuring three widely used virtual flight trainers: FlightGear, Xplane, and Alphalink's Virtual Flight Test Environment (VFTE), providing a valuable resource for those looking to implement co-simulation in their work. The three co-simulation setups, tailored to specific aircraft types, leverage the Connectionless data transfer protocol in real-time for effective data transport, making interaction as easy as possible between the simulation components. Co-simulation between Simulink and Xplane produced effective testing of virtual flights of a Cessna-172 aircraft, precisely tracking height and speed variations while preserving stability in a variety of wind circumstances, a significant accomplishment considering the aircraft's single-propeller design. Furthermore, Simulink's integration with FlightGear and QGroundControl via MAVLink enabled realistic simulation of a Rascal-110 UAV's flight, highlighting its potential for medium-endurance UAV applications. This setup enables precise tracking of the UAV's lateral path on a map, facilitating the evaluation of MATLAB-based 6-DOF UAV models. Moreover, Simulink's integration using software-in-the-loop (SIL) simulation, QGroundControl makes it possible to test sophisticated H-infinity observer-based autopilots and achieve reliable low-altitude flying in windy circumstances. Furthermore, Simulink is integrated with the Virtual Flight Test Environment (VFTE) to replicate the operation of a ZOHD Nano Talon MAV, taking use of the VFTE's customized architecture for this more compact UAV system.

Keywords: Autonomous control of flight; Autopilot systems; MATLAB/Simulink integration; QGroundControl interface; FlightGear and Xplane simulations; Virtual flight testing; Software-in-the-loop (SIL) and Hardware-in-the-loop (HIL) testing

1. Introduction

In 1984, MathWorks, Inc. released MATLAB and Simulink, its graphical user interface, as a software package for commercial use, marking a significant milestone in the history of computational tools., a variety of engineering and science fields have taken advantage of and enhanced MATLAB's model creation and simulation capabilities. Currently, MATLAB/Simulink is utilized for many different real-time engineering applications as well as engineering research, development, and teaching. This study focuses on the ability of researchers, engineers, and hobbyists in aerospace to simulate flight. The Aerospace Toolbox in MATLAB provides a gateway to specialized toolboxes, including the Open Source Flight Control using UAV Toolbox and the UAV Toolbox, a rapidly growing autopilot system in the small UAV community, as well as integration with the free flight simulation software FlightGear, offering a comprehensive suite of tools for UAV development and simulation. Since its introduction in 1984, MATLAB and Simulink have evolved into a versatile platform for modeling and simulation, embraced by a broad spectrum of engineering and scientific communities, who have not only utilized but also enriched its capabilities, driving innovation and advancements in their respective fields. Currently, MATLAB/Simulink is utilized for many different real-time engineering applications as well as engineering research, development, and teaching. Following its introduction in 1984, MATLAB's modeling and simulation capabilities have been embraced and expanded by diverse engineering and scientific communities. Now, MATLAB/Simulink is an indispensable tool in real-time engineering applications, research, development, and education, with its impact felt across numerous fields and industries. The ability of aerospace engineers,

academics, and enthusiasts to replicate flight is the main emphasis of this study. The Aerospace Toolbox in MATLAB makes it easier to connect with more specialized toolboxes, like FlightGear, a free flight simulation application, and the UAV toolbox and UAV toolbox for OPEN SOURCE FLIGHT CONTROL, an autopilot that is gaining popularity in the small UAV field. In [1], the User Datagram Protocol (UDP) enabled the integration of MATLAB/Simulink with Xplane to evaluate the features of a reconnaissance UAV's longitudinal flying mode. Additionally, UDP facilitated the interface between MATLAB/Simulink and Xplane in [2-4] for autopilot testing, where a comparative analysis was performed via co-simulation, contrasting classical PID control with advanced H∞ ideal, reliable control. The method of co-simulation was similarly employed in [5] to construct a cockpit design interface that is reasonably priced. Furthermore, [6] leveraged the integration of Xplane and MATLAB's system identification toolkit to analyze pilot responses during flight, utilizing measured data to inform the analysis. This co-simulation method is applied in [7] to give neural autopilot training a platform. Nonetheless, there aren't enough comprehensive analyses or contrasts of the main options' advantages and disadvantages. Emerging trends are indicating a growing adoption of co-simulation technology in the aerospace industry, particularly for specific classes of unmanned aerial vehicles (UAVs) and manned aircraft, shaping the future of simulation and analysis in these fields. The versatile co-simulation technique has been utilized in a variety of aircraft simulations, from simple models in [8] to creative designs like Ornithopter-type UAVs in [9]. Additionally, as demonstrated in [10], MATLAB and Flight Gear's co-simulation capabilities are being increasingly utilized, particularly identifying specific fixed-wing aircraft models, leveraging UDP interface for data analysis from virtual flight simulations. As an approach, co-simulation is further extended to evaluate the performance of various ideal control methods during erratic weather, including LQR, LQG, and MPC. Additionally, in [11], determining the efficiency of autopilot technology for the Raptor 90, with a pseudo-spectral optimal controller operating in open-loop, is evaluated using a MATLAB-Real Flight G3 simulator interface, demonstrating the versatility of the integrated simulation methodology in performance benchmarking. The integrated simulation methodology was employed in [12], embedding the NASA TLX into Gazebo's simulation framework and drone flight simulator with real-world accuracy enabling the assessment of pilot workload. Moreover, the method was utilized in [13] to simulate and visualize the motion of reusable rockets for different types of aerospace vehicles, showcasing its versatility in aerospace research and development.

Co-simulation is becoming a crucial step in the design and testing process, with more real-flight tests being preceded by simulation studies. As seen in [14], MATLAB and Xplane were used to simulate the aerobatics of a small fixed-wing UAV, enabling the refinement of the design before moving to flight testing. The use of Software-in-the-loop (SIL) simulation, leveraging programming languages like MATLAB, is becoming increasingly popular for exploring UAV formation flying dynamics. This approach enables researchers to investigate and analyze the behavior of UAVs in formation flying scenarios, paving the way for advancements in autonomous aerial coordination. The study of UAV formation flight is benefiting from the growing adoption of co-simulation and software-in-the-loop (SIL) methods. A notable example is [15], where Xplane was integrated with JAVA-based modules, utilizing the NASA WorldWind API, to simulate the formation of multiple UAVs from the ground. This approach is gaining traction, as demonstrated in [14], which employed MATLAB and Xplane for simulating fixed-wing UAV aerobatics before actual flight testing. By leveraging simulation tools and programming languages, researchers are able to extensively test and optimize their designs, leading to an increase in successful flight tests and paving the way for advancements in autonomous aerial coordination.

In [19], a simulation using software-in-the-loop (SIL) was deployed for simulated UAV flight operations and risk minimization, leveraging JSON-based integration of ArduPilot SITL with MATLAB and X-Plane simulation tools. This approach enabled the simulation of UAV flight dynamics and the evaluation of potential risks in a virtual environment, further demonstrating the value of SIL simulations in ensuring safe and effective UAV operation.

In [20], Gazebo was integrated with Ardupilot SITL to simulate the flight of a quadcopter, allowing for a test flight. Meanwhile, [21] leveraged an interface between Labview and Xplane, investigating small UAV failure mechanisms and effects through the lens of STPA. Although various studies have explored co-simulation in aerospace engineering, a comprehensive comparison of the advantages and disadvantages of the main approaches is still lacking. However, emerging trends suggest that co-simulation will play a important role for the advancement of UAV and manned aircraft technology in future designs.

The availability of a UAV toolbox for open-source flight control has enabled the development of advanced simulation and testing capabilities for UAVs. By leveraging MATLAB/Simulink and VFTE, users can perform HIL co-simulation and SIL simulation, integrating with popular autopilots like Pixhawk and ground control software like QGroundControl. This approach enables the verification of flight capabilities, navigation, and control algorithms in a virtual environment, providing an additional layer of safety before actual flights. The toolbox's compatibility with small UAVs like the Nano Talon, used in flying lab kits, further expands its potential applications in UAV research and development.

This research aims to demonstrate the potential of advanced co-simulation techniques in assessing navigation precision in demanding flight scenarios. This co-simulation showcases the capability to command and visualize aircraft motion in 3D space under diverse wind regimes, a critical aspect for general aviation aircraft like the Cessna, whose single propeller can lead to challenging yaw dynamics. Meanwhile, the MATLAB-Flight Gear integrated simulation seeks to highlight the method's usefulness in conducting in-depth route following analysis for new uses for aircraft, such medium-range unmanned aircraft, where precise trajectory tracking is crucial.

The purpose of this study was to show how simple it is to use the Virtual Flight Test Environment (VFTE) to evaluate sophisticated optimum robust flight controllers, including observer-based robust H-infinity control to achieve optimal trade-offs between trajectory tracking precision and disturbance rejection. The research also explores the application of co-simulation methodologies for Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) validation. paper is as follows: Section 2 delves into the real-time communication approaches used in co-simulation with MATLAB/Simulink flight simulation; Section 3 compares the advantages and disadvantages of co-simulation with popular flight simulators and MATLAB/Simulink; Section 4 covers MAVLink communication protocols and ground station software like Mission Planner

and QGroundControl; Section 5 presents a comprehensive implementation of flight simulation with FlightGear and Xplane; Section 6 explains the interface between MATLAB/Simulink and VFTE for both HIL and SIL scenarios; The limits of co-simulation approaches are covered in Section 7, and the study is concluded in Section 8.

2. Protocols for Integrated-Simulation Communications in Real-Time

For co-simulation scenarios involving MATLAB/Simulink and flight simulators like Xplane and FlightGear, UDP has emerged as the dominant protocol, delivering faster communication speeds and lower overhead compared to TCP/IP, while sharing the same IP foundation. Unlike TCP/IP, UDP does not require error recovery or handshaking, resulting in a smaller preamble for the data transfer protocol. UDP supplies an optional integrity check for message confirmation, furthermore it does not guarantee error correction. In practice, the data payload typically ranges from 0 to 512 bytes per frame, although the theoretically with a 16-bit message length field limits the maximum message size to 65,527 bytes, the UDP protocol message format includes source and destination port numbers (2 bytes each), message length (2 bytes), and an optional checksum (2 bytes). Nevertheless, TCP/IP is still a key consideration for applications where data integrity is essential.

UDP transmit and receive blocks are easily accessible in MATLAB/Simulink, where they function as two separate, unidirectional broadcasts utilizing distinct IP addresses. This is also the case with Xplane, which permits certain IP addresses to be used in conjunction with MATLAB/Simulink to broadcast or receive specific data at a predetermined frequency. The configuration process depending on the installation location of the flight simulator, either on the user's PC or a different machine. MATLAB/Simulink allows users to generate a customized batch file (with a .bat extension) tailored to the specific aircraft for UDP communication with FlightGear. Execute this batch file from the MS-DOS command prompt to launch FlightGear and initiate a 3D flight simulation with the selected aircraft. Additionally, the procedure can be nuanced, requiring manual adjustments to the batch file lines with a specific syntax as specified in the FlightGear command line documentation.

3. An In-Depth Review of Integrated Simulation Software Tools for Flight Simulation

So far, several prominent aviation simulation tools have been expertly merged with MATLAB/Simulink. The majority of Illustrations of co-simulation in published works utilize FlightGear and Xplane, with Xplane being particularly popular due to its realistic 3D display capabilities and accurate flight dynamics simulation based on blade element analysis. Many applications have combined Xplane's strengths with MATLAB's advanced toolbox features and GNC functionalities for navigation, guidance, and control. Xplane's professional version, endorsed by the FAA, offers a comprehensive pilot training experience when paired with compatible controls. Meanwhile, FlightGear is attracting attention for its open-source and free access, boasting advanced graphics capabilities, precise geometric accuracy, and a dedicated simulator interface block within the MATLAB/Simulink aerospace blockset toolbox's animation tools, making it an appealing choice for simulation enthusiasts. MATLAB/Simulink has also demonstrated its versatility by successfully linking with FSX and FS2020: Two generations of Microsoft Flight Simulator, allowing for real-time flight simulation capabilities, made possible by the Simconnect toolbox accessible on Github. This toolbox utilizes an Simulink S-function blockset, allowing users to define the data to be sent or received, facilitating seamless connectivity with the flight simulator. Lockheed bought Microsoft Flight Simulator X's commercial version, ESP, due to its advantageous features. The flight simulator versions that were interfaced with MATLAB are no longer supported, so they are not included in Table 1's comparison. The addition of RealFlight [11, 12] to the flight simulation toolkit enables the replication of a broader range of aircraft types, including small unmanned aerial vehicles (UAVs) like Quad-wing aircraft, Quad-rotor aircraft, and various Autonomous Flying Machine combinations. By integrating leveraging Ardupilot's SITL technology, is a vital tool for autonomous small UAV systems, users can now simulate a wider variety of aircraft behaviors and scenarios with greater flexibility and realism. A comparison of Xplane, FlightGear, and RealFlight is shown in Table 1, with particular attention paid to the kinds of aircraft that are being considered, extra implementation concerns, and the mathematical models that are utilized to describe the dynamics of the aircraft. The analysis presented earlier allowed for a thorough comparison and assessment of the current techniques for simulating flight in a collaborative environment. The main outcomes from this comparison are that all three methods (Xplane, FlightGear, and RealFlight) are viable options when it comes to flight co-simulation, MATLAB-Simulink toolboxes offer a comprehensive solution. However, when high-fidelity flight dynamics are crucial, Xplane stands out as the top choice. Meanwhile, FlightGear excels in terms of seamless real-time software interfacing and straightforward implementation. On the other hand, RealFlight provides unparalleled flexibility for smaller UAV systems, making it an attractive option for specific use cases.. Additionally, an alternative tool, VFTE, specifically designed for small UAV systems, will be discussed in Section 6.2.

Table 1: Comparison of Flight Simulator Features

	X-plane [1,2,5,8,9,20]	FlightGear [10,18,20]	RealFlight [11,20]
Design tools for Aircraft	Planemaker enables the creation of detailed geometric definitions for various aircraft components, ranging from the fuselage to the entire body.	Openscenegraph supports a variety of file layouts, including AC3D, VRML1, and DXF, and also features a built-in Model Airplane Designer tool for creating and editing aircraft models.	The AccuModel™ aircraft editor allows users to input essential aircraft parameters, including mass and geometric characteristics, for accurate modeling and simulation
Realistic 3D visualization	Global scenery of exceptional quality, paired with the free OpenSceneryX library, providing an vast array of 3D objects for enhanced realism.	High-resolution global scenery with exceptional detail and accuracy, notably in Europe and the USA. [18].	High-performance 3D graphics, optimized for immersive experiences with Oculus Rift and HTC Vive.
Products Accessibility	Commercial (USD 60)	Free software with source code included	Commercial (USD 100)
Environments	Compatible with Windows, macOS, and Linux operating systems	Compatible with Windows, macOS, and Linux operating systems	Exclusive to Windows, not compatible with Linux operating systems.

Simplified simulation and program execution workflow	Easy-to-use interface for integrating with MATLAB/Simulink and SITL, with clear documentation and simple execution.	Easy to execute, but co-simulation requires precise configuration and batch file management. [9].	Our co-simulation solution utilizes Ardupilot SITL, which communicates with MATLAB, but unfortunately, direct Simulink integration is an uncommon practice, with only a few instances documented, according to [11]

4. Ground-Based Communication Software and Protocols

The two most well-known UAV ground support software are QGroundControl and MissionPlanner, and the two mentioned software are open-source. These software packages offer a range of features, including the ability to configure flight navigation paths for both real-world and artificial flights, select various flight modes as specified in the Ardupilot documentation, and upload default flight codes for various UAV configurations, such as fixed-wing, helicopter, multirotor, hybrid, and tail-aft fuselage designs. While QGroundControl is generally more compatible with multiple platforms, MissionPlanner is primarily designed for PC use.

The MAVLink protocol, widely used in miniature aerial vehicles (UAVs), enables data exchange between Simulink in MATLAB and QGroundControl. Optimized for critical data transmission, this protocol enables Simulink to send key information to QGroundControl, facilitating the display of the UAV's flight path on a Google map. Additionally, autopilots like the well-known Pixhawk line of ARM-cortex-based autopilots communicate with ground control software via MAVLink. You can find more information about the MAVLink protocol in [21, 23]. Missions needing more secure communication have led to the development of enhanced security protocols like MAVsec [24].

MAVLink blocks (refer to Figure 1) in the MATLAB/Simulink UAV toolbox facilitate connection with QGroundControl. Figure 1 illustrates the library that provided the initial MAVLink heartbeat block, used for determining payload type and data rate, and the MAVLink serialization block, which reformatted the virtual bus message into an unsigned 8-bit integer data stream. In our implementation, we utilized MAVLink to transmit data to QGroundControl, but we didn't require receiving data in return. This approach could be beneficial in scenarios where QGroundControl directly defines the flight path. Additionally, the MAVLink deserialization block can be employed to decode MAVLink message data when necessary, allowing for flexible communication and data exchange.

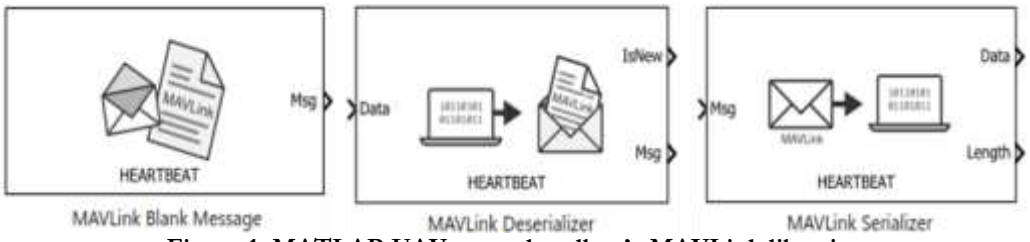

Figure 1: MATLAB UAV control toolbox's MAVLink libraries

The 2nd figure shows how to connect MATLAB Simulink's attitude and altitude signals to QGroundControl using the MAVLink protocol. Orientation and rotational speed data are transmitted as a data bundle using the bus assignment block.

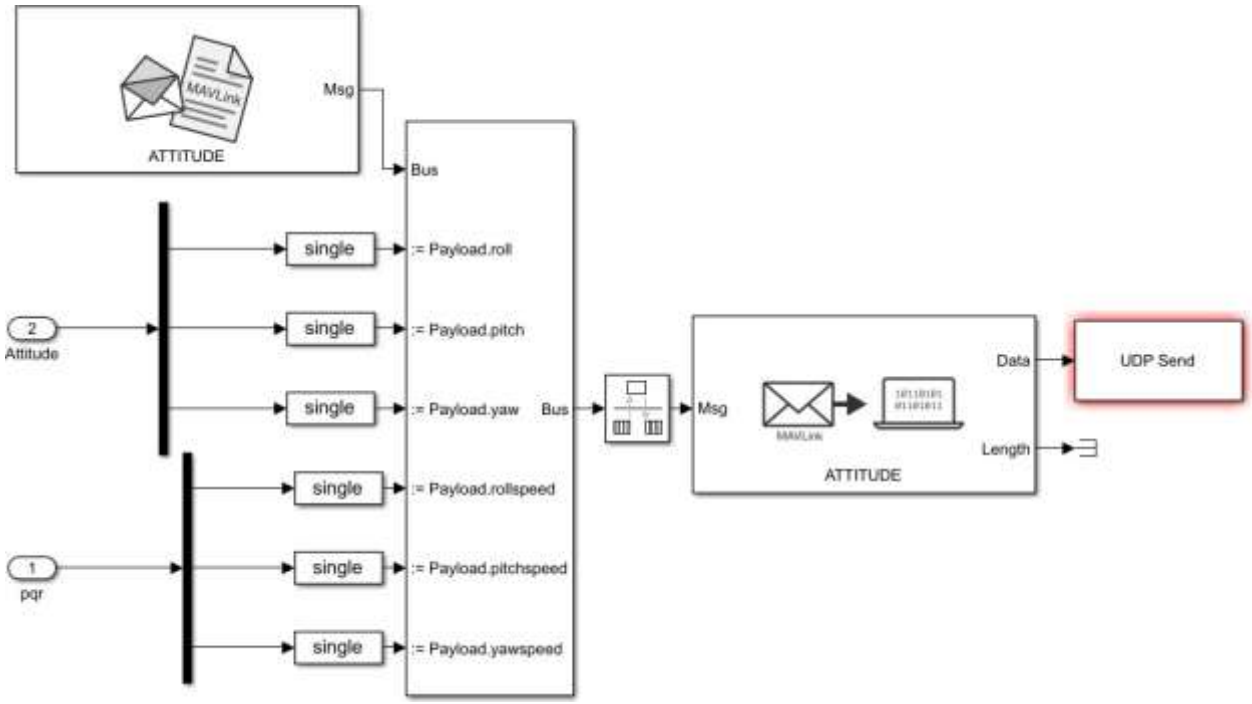

Figure 2: MAVLink-enabled data transfer link between Simulink and QGroundControl

The Analyze Tools menu in QGroundControl additionally features an MAVLink Inspector tool. The inspector displays a comprehensive list of incoming commands for the vehicle, providing detailed information on variable types, values, and message fields, including update frequency, count, and component ID. It's notable thatthe system receives the heartbeat message at a frequency of approximately 1 Hz, while data from IMU sensors operate at significantly higher frequencies, and GPS, which typically runs at frequencies between 5 Hz to 10 Hz.

5. MATLAB/Simulink integrations with Xplane and FlightGear

This segment details how we at Coventry University used Xplane and FlightGear to build MATLAB-Simulink co-simulation. It is demonstrated that this tool makes it possible to assess certain important aspects of aircraft flight control as well as autopilot designs.

5.1 Co-simulation interface: MATLAB/Simulink with Xplane

A co-simulation environment was set up, assuming a traditional Cessna-172 aircraft model available in Xplane-11. The MATLAB/Simulink model was designed to implement autopilot control systems for pitch, altitude, speed, and heading. The co-simulation with Xplane enabled the validation and assessment of these designs. Initial conditions were synchronized using the MAP feature, matching MATLAB's initial state with Xplane's initial altitude and location. A rate of 20 packets per second was chosen for the frame rate, with UDP communication established between Xplane and MATLAB using the IP address 1270.0.0.1 and ports 49,000 and 49,004 for data exchange in both directions. Figures 3 and 4 illustrate the UDP send setup and input/output selection with UDP configuration on the MATLAB side, respectively.

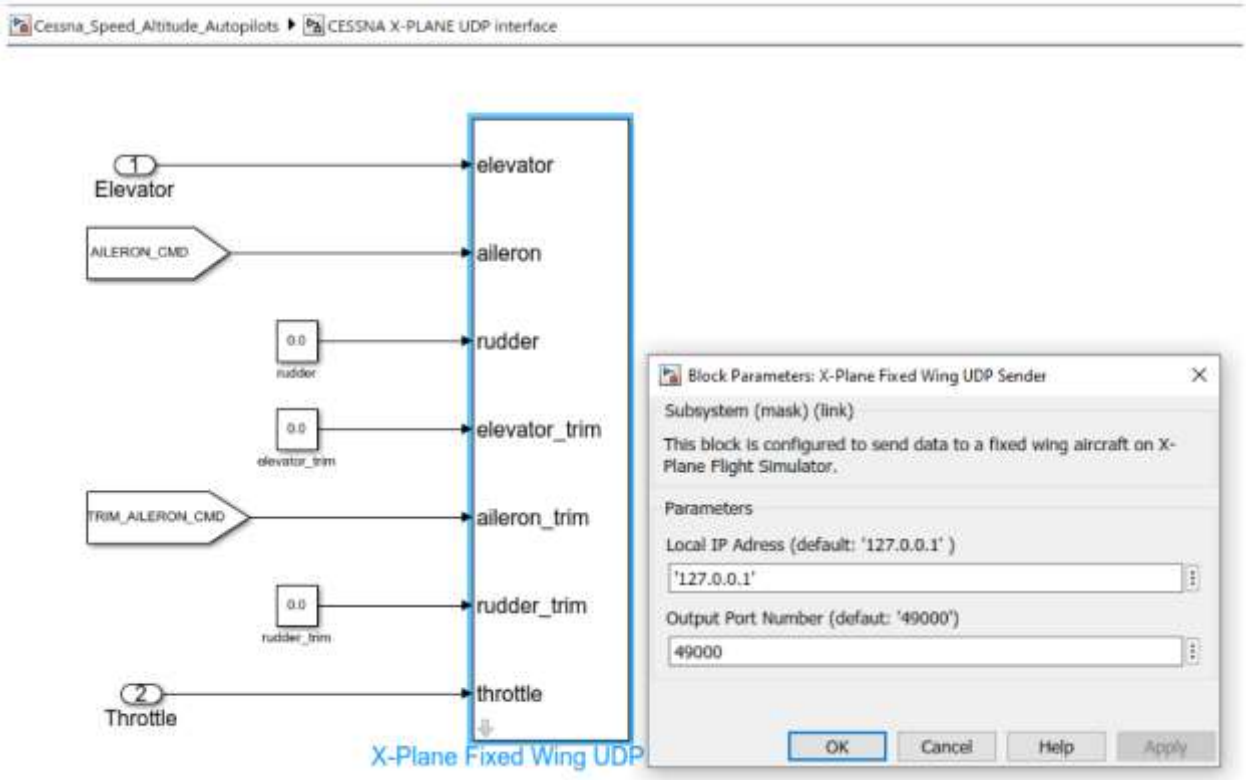

Figure 3: MATLAB/Simulink UDP data transmission block

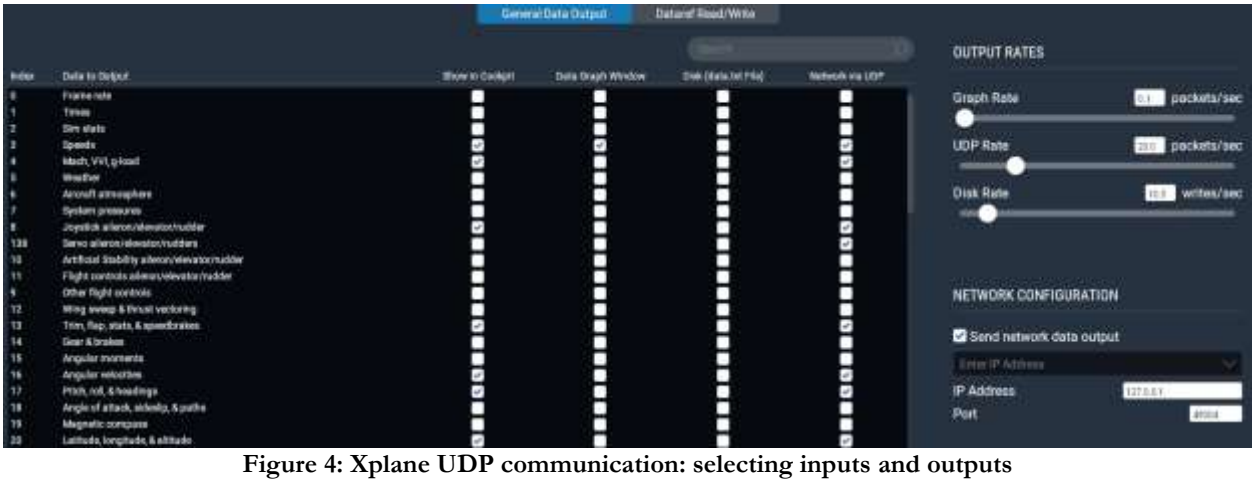

Figure 4: Xplane UDP communication: selecting inputs and outputs

Xplane can simulate various weather conditions, including clear, windy, and stormy environments, at different times of day. Alternatively, MATLAB-Simulink can input custom weather conditions. Additionally, MATLAB offers a formal model of Dryden wind gusts, derived from aircraft autopilot scenarios. Figure 5 shows the Simulink autopilot system, integrated with the Xplane interface, controlling speed and altitude. This system employs a traditional sequential loop closure technique, utilizing a pitch inner loop, as described in [25], to regulate altitude. To allow for simultaneous control of speed and altitude, we installed a speed autopilot.

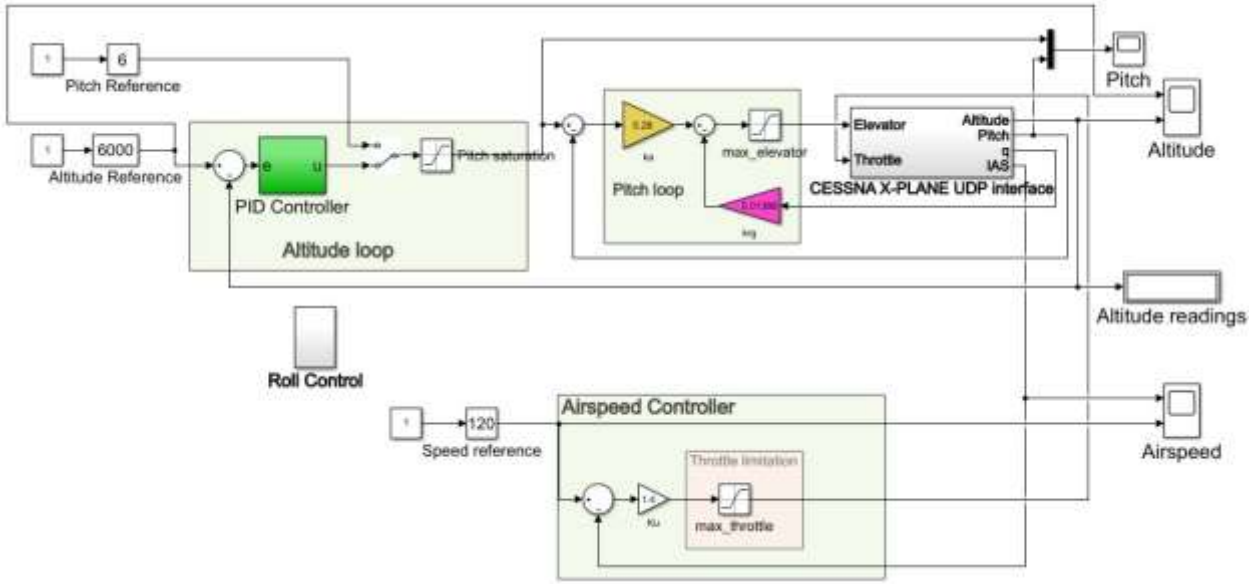

Figure 5: Simulink autopilots design for speed and altitude control, linked to Xplane via UDP protocol

The following simplified speed and altitude control rules are applied:

$$\delta_e = k_\theta(\theta_c - \theta) - k_q q \quad (1)$$

The variables are defined as follows: elevator deflection (δe), pitch rate (q), commanded pitch (θc), actual pitch angle (θ), and controller gains (kθ and kq). The commanded pitch (θc) is used to track the target altitude and is calculated as:

$$\theta_c = k_h(h_c - h) \quad (2)$$

where k_h is an altitude control gain and h, h_c, and k indicate the current and intended elevations, respectively. Similarly, proportional control in conjunction with autothrottle is used to regulate speed.

$$\delta_\tau = k_\tau(u_c - u) \quad (3)$$

The desired axial speed is represented by u, while uc represents the actual axial speed, with kτ being a positive gain coefficient. The throttle lever angle is denoted by δτ. To ensure control inputs remain within acceptable limits, saturation is applied. Each loop's proportional control is enough to keep setpoints constant, we note that replacing the gains with proportional-integral (PI) control can enhance tuning flexibility.

As seen in Figure 6, the pursuit view may be chosen from the Xplane menu.

After that, you pick the Xplane map and center it on the airplane by using the N key. You may then enter the aircraft's initial speed and altitude after opening the map, as seen in Figure 7. The starting altitude and speed must match the initial values set in Simulink.

Figure 8 illustrates the flying co-simulation, where the intention was to always maintain a steady control over altitude and speed references while maintaining roll stability using throttle and elevator inputs. Real-time data on speed and altitude is displayed on Simulink scopes and Xplane. The simulation effectively regulated speed within the range of 100 to 120 knots and altitude from 8000m to 6000m. Disregard the initial zero speed in the graphs, as it's simply a artifact of data collection before the aircraft's starting position was established in Xplane. The Cessna's single axial propeller creates a yawing force, making it challenging for inexperienced pilots to manage. To isolate longitudinal control from roll motion, the system also activated a roll control loop.

Figure 6: Choosing a view in Xplane

Figure 7: Specifying starting conditions for altitude and speed in Xplane.

Figure 8:I ntegrated flight simulation platform leveraging MATLAB/Simulink and X-Plane.

5.2 Co-simulation platform: MATLAB/Simulink and FlightGear

A co-simulation was conducted using a Rascal 110 UAV model from Thunderfly Aerospace (available on GitHub), demonstrating custom aircraft integration. The aircraft design file was deployed to FlightGear 2020.3 by copying it to the \Aircraft folder, setting up the simulation framework. Note that the "-master" prefix, automatically generated by GitHub, must be removed from the filename. Before creating a batch script, it's essential to verify the installed aircraft's filename and version by launching FlightGear independently and checking the command window output.

The aircraft's name is verified through the command window, the parameters that are now selected, and other instructions that the batch file can send to FlightGear. The.bat file itself makes use of the same commands.

A special FlightGear bat file generator block in MATLAB is used to create the batch script. The blue block in Figure 9 is this one; it lets us choose the aircraft shape, beginning characteristics, and initial position. To ensure seamless simulation, we enabled the generation of scenery on-the-fly, as no sceneries were pre-downloaded. It is advised to disable shader settings because doing so significantly boosts the simulation's performance on computers without expensive graphics cards or CPUs. Table 2 displays the settings of the Simulink run script configuration.

The batch file automates the launch of FlightGear with the Rascal UAV, setting the software up to receive data from Simulink. As the Simulink simulation begins, the UAV initializes with the predefined conditions in MATLAB, creating a unified and synchronized simulation environment.

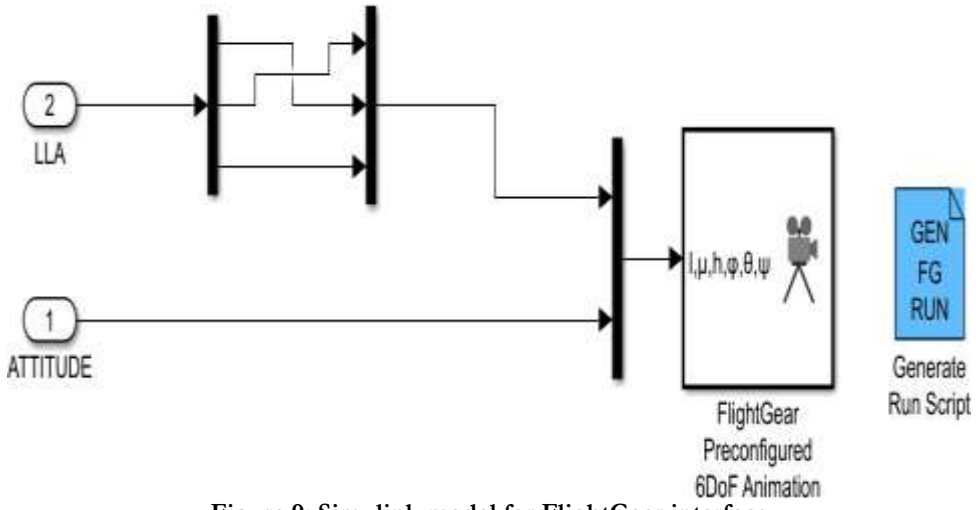

Figure 9: Simulink model for FlightGear interface

Table 2: Configuring parameters for FlightGear script generator.

Parameter	Values
FlightGear geometry model name:	c172p
Airport ID:	KSFO
Runway ID:	10L
Initial altitude(ft)*:	7224
Initial heading (deg)*:	113
Offset distance (miles)*:	4.72
Offset azimuth(deg)*:	0

A nonlinear six-degree-of-freedom (6DoF) mathematical model of the Rascal 110 Unmanned Aerial Vehicle (UAV) was developed and implemented in MATLAB/Simulink. The model's parameters were optimized utilizing the fminsearch algorithm, as outlined in [26], to decrease the probability of errors when addressing nonlinear equations. Subsequently, linmod tool was employed to linearize the model, facilitating the analysis of the UAV's dynamics. The FlightGear block in the simulator necessitates input data, including the aircraft's Euler angles (roll, pitch, and yaw) and geodetic coordinates (longitude, latitude, and altitude), to accurately simulate the UAV's behavior and trajectory.
The nonlinear dynamics are expressed in vector form as follows:

$$m(\dot{\mathbf{V}}_b + \boldsymbol{\omega}_b \times \mathbf{V}_b) = \mathbf{F}_b \tag{4}$$

$$\mathbf{J}\,\dot{\boldsymbol{\omega}}_b + \boldsymbol{\omega}_b \times \mathbf{J}\boldsymbol{\omega}_b = \mathbf{m}_b \tag{5}$$

The resultant external force (Fb) in the body frame, angular velocity (ωb) of the body frame with respect to the inertial frame, and moment vector (mb) acting on the UAV are used to formulate the equations of motion, the mass (m), and the moment of inertia matrix (J). A velocity vector (Vb) representation in body coordinates is utilized. Forces and moments models, originally developed by Christopher Lum for the RCAM aircraft (drawing on Garteur group's work [27]), were modified and implemented for the Rascal-110 UAV, characterized by a single central axial propeller design. These models establish a relationship between the forces (Fb) and moments (mb) and the control inputs (elevator, aileron, rudder, throttle) and system states, enabling the simulation of the UAV's dynamics. In Figure 10, the Simulink model is displayed. Please take note that the GPS sensor should not be used with MATLAB versions 2021A or below. The simulation was run at an altitude of 1000 meters and a trim speed of 85 meters per second. To validate the aircraft's responses, flight testing was conducted with manual control inputs covering a range of values: elevator (-10° to 25°), throttle (0.5° to 10°), aileron (-25° to 25°), and rudder (-30° to 30°). This testing aimed to verify the model's accuracy in predicting the aircraft's behavior across nine state variables: 3 speeds (axial, vertical, lateral), 3 attitudes (pitch, roll, yaw), and 3 angular rates (roll, pitch, yaw).

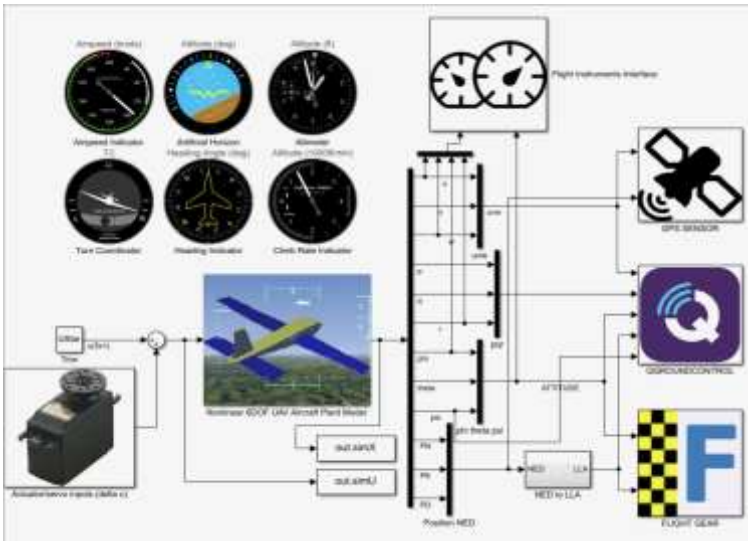

Figure 10: 6DoF FlightGear and QGroundControl co-simulation model in Simulink

The 11th figure illustrates a 3D open-loop maneuver simulation, resulting from the successful interfacing of a 6DoF nonlinear UAV dynamics model in Simulink with FlightGear's simulation framework using UDP communication. This integration enables a 3D visualization of the UAV's behavioral dynamics, allowing for a more comprehensive understanding of its performance.

Figure 11: Rascal 110 UAV simulation with FlightGear and QGroundControl interfaces

6. Examining the Limitations of Co-Simulation Approaches

The study utilizes a combination of commercial software tools, comprising X-Plane, MATLAB/Simulink from The MathWorks Inc., and Alphalink's VFTE, to facilitate its research activities and data analysis, for co-simulation purposes. Please note that certain features in the Simulink toolbox, such as the UAV toolbox, necessitate the latest MATLAB versions (for instance, MATLAB 2021a or later for the GPS block functionality). When upgrading to a newer MATLAB version, modifications to the models may be required to ensure compatibility and optimal performance. Fortunately, the MATLAB script used in this study was compatible with all recent MATLAB versions tested. The professional edition of Xplane offers advanced features, and additional plugins may be required for complex simulations like formation flying. Currently, Alphalink software is exclusive to the NanoTalon UAV. However, this setup enables remote access to realistic flight simulations via the cloud, reducing onboard PC resource requirements without compromising real-time simulation performance. The experimental outcomes demonstrated that a single personal computer with standard hardware configurations (8 GB RAM, 1.8 GHz Intel quad-core CPU, and sufficient storage capacity) can support seamless co-simulations of X-Plane-MATLAB, VFTE-MATLAB, or FlightGear-MATLAB, enabling real-time flight simulation and analysis capabilities.

7. Conclusions

This research investigated the application of MATLAB/Simulink to various flight simulation programs, highlighting differences in their 3D simulation capabilities. Xplane, a commercial software primarily designed for manned aircraft, demonstrated a straightforward method for a realistic flight dynamics model based on blade element theory enables real-time co-simulation via UDP, accommodating various aircraft configurations. Co-simulation using Simulink and X-Plane effectively replicated the flight dynamics of a Cessna 172 aircraft, showcasing its capability for simultaneous altitude and speed control under various flight conditions, making it particularly suitable for general aviation aircraft. Furthermore, lateral stability

enhancementwas implemented to mitigate the yawing moment induced by the single propeller. In contrast, FlightGear, an open-source software, demonstrated capabilities in simulating medium- to high-end UAVs, offering flexibility in selecting flight dynamics models. The Rascal-110 UAV was chosen as a case study for FlightGear, showcasing a seamless interface with QGroundControl, a ground control station program ideal for path following. A comprehensive tutorial on the co-simulation procedure for FlightGear and X-Plane is provided. While Real Flight is suitable for small UAV simulation, the lack of direct interface information with MATLAB/Simulink led to the evaluation of VFTE software for co-simulation in small UAV scenarios.

REFERENCES

[1] Silva, W.R.; Da Silva, A.L.; Grundling, H. Modelling, Simulation and Control of a Fixed-Wing Unmanned Aerial Vehicle (UAV). In Proceedings of the 24th ABCM International Congress of Mechanical Engineering, Curitiba, Brazil, 3–8 December 2017.

[2] Ribeiro, L.R.; Oliveira, N.M.F. UAV Autopilot Controllers Test Platform Using MATLAB/Simulink and X-Plane. In Proceedings of the Frontiers in Education Conference, Arlington, VA, USA, 27–30 October 2010.

[3] Yuceol, O.E.; Akbulut, A. Modeling and Simulation of a Small Unmanned Aerial Vehicle. Adv. Intell. Syst. Comput. 2013, 210, 245–255.

[4] Nugroho, L. Comparison of Classical and Modern Landing Control System for a Small Unmanned Aerial Vehicle. In Proceedings of the 2014 International Conference on Computer, Control, Informatics and Its Applications: "New Challenges and Opportunities in Big Data", IC3INA 2014, Bandung, Indonesia, 21–23 October 2014.

[5] Cameron, B. Development and Implementation of a Cost-Effective Cockpit Interface Architecture for Flight Simulation. Master's Thesis, The Carleton University, Ottawa, ON, Canada, 2016.

[6] Jalovecky, R.; Boril, J. Analyis of Measured Pilot Resonses during the Flight. Mosatt 2013, 2013, 6.

[7] Pinguet, J.; Feyel, P.; Sandou, G. A Neural Autopilot Training Platform based on a MATLAB and X-Plane co-simulation. In Proceedings of the 2021 International Conference on Unmanned Aircraft Systems (ICUAS), Athens, Greece, 15–18 June 2021;pp. 1200–1209.

[8] Arif, A.A.; Stepen; Sasongko, R.A. Numerical Simulation Platform for a Generic Aircraft Flight Dynamic Simulation. Int. J. Eng. Technol. 2018, 7.

[9] Kaviyarasu, A.; Kumar, K.S. Simulation of Flapping-Wing Unmanned Aerial Vehicle Using x-Plane and MATLAB/Simulink. Def. Sci. J. 2014, 64, 327–331.

[10] Aschauer, G.; Schirrer, A.; Kozek, M. Co-Simulation of MATLAB and FlightGear for Identification and Control of Aircraft. IFAC- Papers 2015, 48, 67–72.

[11] Grady, N.B.; Frye, M.T.; Qian, C. The Instrumentation and Flight Testing of a Rotorcraft Vehicle for Undergraduate Flight Control Research. Available online: https://arc.aiaa.org/doi/10.2514/6.2006-6739 (accessed on 30 July 2022).

[12] Abioye, A.O.; Prior, S.D.; Saddington, P.; Ramchurn, S.D. The Performance and Cognitive Workload Analysis of a Multimodal Speech and Visual Gesture (MSVG) UAV Control Interface. Robot. Auton. Syst. 2022, 147, 103915.

[13] Sagliano, M. Open-Source Visualisation of Reusable Rockets Motion: Approaching Simulink–FlightGear Co-simulation. In Proceedings of the 2021 AIAA Scitech Forum, AIAA 2021-0410, Virtual, 11–15 & 19–21 January 2021. [CrossRef]

[14] Bulka, E.; Nahon, M. Autonomous Fixed-Wing Aerobatics: From Theory to Flight. In Proceedings of the IEEE International Conference on Robotics and Automation, Brisbane, Australia, 21–25 May 2018.

[15] Angonese, A.T.; Rosa, P.F.F. Ground Control Station for Multiple UAVs Flight Simulation. In Proceedings of the 2013 IEEE Latin American Robotics Symposium, LARS 2013, Washington, DC, USA, 21–27 October 2013.

[16] Madridano, Á.; Al-Kaff, A.; Martín, D.; de la Escalera, A. 3d Trajectory Planning Method for UAVs Swarm in Building Emergencies. Sensors 2020, 20, 642.

[17] Baidya, S.; Shaikh, Z.; Levorato, M. FlynetSim: An Open Source Synchronized UAV Network Simulator Based on Ns-3 and Ardupilot. In Proceedings of the MSWiM 2018—The 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Montreal, QC, Canada, 28 October–2 November 2018.

[18] Sun, Y.; Zhen, Z.; Ou, C.; Pu, H. 3D Scene Simulation of UAVs Formation Flight Based on FlightGear Simulator. In Proceedings of the 2014 IEEE Chinese Guidance, Navigation and Control Conference, CGNCC 2014, Yantai, China, 8–10 August 2014.

[19] Johnson, C. Development of a Software in The Loop Simulation Approach for Risk Mitigation in Unmanned Aerial System Development. Master's Thesis, Oklahoma State University, Stillwater, OK, USA, 2020.

[20] Millan, S. Realistic VTOL Simulator. Master's Thesis, Catalonia Polytechnic University, Barcelona, Spain, 2020.

[21] Mattei, A.L.P.; Orbital, E.S.A.; Toledo, C.F.M.; Arantes, J.D.S.; Trindade, O., Jr. Unmanned Aerial Vehicles Flight Safety Improve- ment Using In-Flight Fd. Intell. Inf. Manag. 2021, 13, 97–123.

[22] Guther, A. Simconnect Toolbox for MATLAB/Simulink. Available online: https://github.com/aguther/simconnect-toolbox (accessed on 30 July 2022).

[23] Koubaa, A.; Allouch, A.; Alajlan, M.; Javed, Y.; Belghith, A.; Khalgui, M. Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey. IEEE Access 2019, 7, 87658–87680.

[24] Allouch, A.; Cheikhrouhou, O.; Koubaa, A.; Khalgui, M.; Abbes, T. MAVSec: Securing the MAVLink Protocol for Ardupilot/PX4 Unmanned Aerial Systems. In Proceedings of the 2019 15th International Wireless Communications and Mobile Computing Conference, IWCMC 2019, Tangier, Morocco, 24–28 June 2019.Bittar, A. X-P

[25] lane Library.zip, MATLAB Central File Exchange. Available online: https://www.mathworks.com/MATLABcentral/

fileexchange/47516-x-plane-library-zip (accessed on 30 July 2022).

[26] Christopher Lum Solving Systems of Equations Using the Optimization Penalty Method. Available online: https://www.youtube. com/watch?v=rx2vUzjuDc0 (accessed on 30 July 2022).

[27] Garteur. Robust Flight Control Design Challenge Problem Formulation and Manual: The Research Civil Aircraft Model (RCAM). Available online: https://www.scienceopen.com/document?vid=127ac6ee-15a3-438d-b3c8-f023dfbb83dd (accessed on 30 July 2022).