# An Environment for Indoor Testing and Diagnosis of Drones using Co-simulation

Renato R. de Abreu, Thyago Oliveira, Leydson Silva, Tiago P. Nascimento and Alisson V. Brito
Universidade Federal da Paraiba (UFPB)
Joao Pessoa, Brazil
Email: renatodeabreu@gmail.com, tiagopn@ci.ufpb.br, alisson@ci.ufpb.br,
thyago.oliveira@eng.ci.ufpb.br, leydson.sk@gmail.com

***Abstract*—The objective of this work is to present a testing tool, which analyzes and evaluates drones during the flight in indoor environments. For this purpose, the framework Ptolemy II was extended for communication with real drones using the High-Level Architecture (HLA) for data exchanging and synchronization. The presented testing environment is extendable for other testing routines and is ready for integration with other simulation and analysis tools. In this paper, two failure detection experiments were performed, with a total of 20 flights for each one, which 80% were used to train a decision tree algorithm, and the other 20% flights to test the algorithm in which one of the propellers had an anomaly. The failure rate or detection rate was 70% for the first experiment and 90% for the second one.**

**Key-words: Unmanned Aerial Vehicle, Testing, Hardware-in-the-loop Simulation, High-Level Architecture**

## I. Introduction

The Unmanned Aerial Vehicles (UAVs), popularly known as drones, had acquired more importance in recent years. Due to its high aerodynamic capacity, drones are used in vigilance and rescue missions [1]. They can be remotely piloted or execute pre-programmed tasks, and since they are not manned, its size and weight are reduced compared with conventional aircraft. Nowadays, it is not uncommon to observe drones flying over our heads.

Every drone is susceptible to failure and accidents caused by drones are being occasioned due to malfunctions, which is not easy to foresee [2], [3]. This unpredictability brings risks to humans and may cause severe material and environmental damages.

One alternative is to monitor and verify the drone behavior in an indoor and controlled scenario. The monitoring might increase safety before tests in open spaces. This article proposes an environment for the elaboration and execution of tests using real drones to diagnose faults and abnormal behaviors. The system allows the configuration of various types of tests and analyzes drone behavior using a drag-and-drop approach. The concatenation of actors defines the path to be performed by the drone. Specific actors were implemented and integrated into the environment to perform diagnostics by analyzing telemetry data collected from the Drone during the flights.

For this, the Ptolemy II simulation environment was integrated into the Drone using the standard for interoperation of simulators, IEEE 1516 (HLA) [4]. This allows not only the integration of Ptolemy with the drone, but also opens the opportunity to integrate other simulators, like network simulator [5], ROS [6], embedded systems [7], SystemC [8], or even a group of diverse simulator [9].

In this work, the behavior of a physical drone in a controlled environment is analyzed. For this, two scenarios were configured: in the first one, the drone performs a free flight, only taking off, stopping, and landing. In the second one, a weight of 0.1 gram was added to one of the propellers. Thus, a diagnosis approach was implemented to detect the unbalancing problem.

Several groups of researchers around the world are seeking for new strategies to test or to evaluate drones, in this context, various works can be cited: [10], [11], [12], [13], [14], [15] and [16]. However, the integration of the drone with Ptolemy via HLA is not yet present in the literature. This results in the two main contributions of this work: 1) the development of a testing and diagnosis tool for drones using Ptolemy; and 2) the integration of drones via HLA to allow the developments of other tools in the future.

## II. Related Work

It is possible to find works, such as [17], which studies the influence in failures of the drones physical components and assess a change of performance after an occurrence of some damage.

The work of [17], as well in this research, analyzes the drone behavior during the flight. Nevertheless, the authors focus on analyzing vehicle behavior after the occurrence of failures in the aileron. Then, it verifies the flight performance of the drone damaged by the failure. Data collected from the sensors can be obtained and used to reconfigure the vehicle orientation system, while the present work focuses on the implementation of a test environment to analyze the drone behavior in a simulated environment for monitoring and analysis during the flight. The works of [18] and [19] also seek to monitor and evaluate the performance of drones flights, in both works are studied vehicles powered by solar energy seeking to realize long flights.

Many of the works that monitor the drone performance during flights seek to achieve results that help improve flight safety, such as in [20], which proposes an in-flight safety assessment, to upgrade safety by a statistical method or simulated tests in soil using HIL. In that study, it proposes a simulation method in which it seeks to test the drone response in front of input stimulus that simulates extreme adverse conditions and from this to evaluate the vehicle behavior checking the reactions to the stimulus received. Though our work evaluates drones behavior, we focus on the construction of a test environment in which can be used for any indoor test, as new actors can be developed and integrated to increase the set of possible tests and diagnosis.

The work of [21] brings a different approach. That study seeks to build a structure for quadcopters capable of communicating with a ground station, and through wireless communication, it is possible to monitor the vehicle positioning during the flight. That work is similar to the present study, as both aims at the construction of infrastructures to monitor the drones state in real-time, however, that work aims to present the data in a visual interface, while this work aims to perform analyzes using Hardware-in-the-loop (HIL) simulation, integrating the drone in real-time not only with our analysis tool but also with other tools in the future, using HLA.

## III. Architecture of the Environment for UAV Testing

In simulations following the High-Level Architecture (HLA) specification [4], each participant is a Federate, which sends and receives messages, and the set of all Federates in the same domain creates a Federation. The RTIG is the gateway that centralizes communication and is responsible for ensuring the delivery of messages in the correct order according to global time, called here, HLA time.

The proposed environment (Figure 1) is divided into three parts: Drone Federate, Ptolemy Federate, and RTIG/HLA. The Drone Federate is the embedded software in the drone itself and is responsible for sending commands to move the drone and also collect and publish data from the drone telemetry to the Federation. The Ptolemy Federate contains two components, one responsible for creating the test sequence and another one responsible for analyzing the telemetry data published by Drone Federate. The RTIG is responsible for communication and synchronization between the Federate Drone and Ptolemy Federate.

The Drone Federate uses PyHLA [22], and the Ptolemy Federate uses JCerti [23]. In Ptolemy, communication with HLA is performed by the actors: HLAManager, HLAPublisher, and HLASubscriber (Figure 1). For integration of Ptolemy with HLA, actors from Ptolemy distribution were used [24]. The actor HLAManager (Figure 1) is a special actor that does not process messages but ensures that the model time is consistent with the HLA time. This actor has no inputs or outputs ports. The role of the HLASubscriber actor is to bring the published values from the Federation and sending it to the model using its output ports. The HLAPublisher actor has the task of publishing updated values from the model to the Federation. A Federate can only send and receive updates of a particular object if it is registered for that. This requires that each Federate registers the publish and subscribe policies in advance [25].

The implementation of a test begins by setting the mission to be performed by the drone. The concatenation of actors (developed and added to the library of Ptolemy II) creates a mission (Table I). For example, for a drone perform a square path, the drone must take off, move forward for 2 seconds at an acceleration of 0.25; move to left for 2 seconds at an acceleration of 0.25; move backwards for 2 seconds at an acceleration of 0.25, move to left for 2 seconds at an acceleration of 0.25 and then land.

TABLE I
Actors Developed for UAV Testing

| Actor | Description | Parameters |
|---|---|---|
| DroneType | specifies the type of the drone | DroneType, NumberOfRepetions |
| TakeOff | performs takeoff procedure | Height |
| Land | performs landing procedure | none |
| Rover | keeps the drone in a fixed position | Time of rover |
| Forward | Move the drone forward | Speed, TimeInSeconds |
| Backward | Move the drone back | Speed, TimeInSeconds |
| Left | Move the drone left | Speed, TimeInSeconds |
| Rigth | Move the drone to right | Speed, TimeInSeconds |
| TurnLeft | Turns the drone 90 degrees to the left | Speed, TimeInSeconds |
| TurnRight | Turns the drone 90 degrees to the right | Speed to right, TimeInSeconds |
| MoveUp | increases the altitude of the drone | Speed, TimeInSeconds |
| MoveDown | decreases the altitude of the drone | Speed, TimeInSeconds |

After the drone mission definition, the model is executed in Ptolemy. So a string containing the drone mission is published in the Federation to be read by Drone Federate. The Drone Federate reads the mission, starts the drone, and performs the defined mission. At the same time, it starts the collection and publication of drone telemetry data read from the sensors of the drone. The data published by Drone Federate is read in Ptolemy Federate and sent to the actor Drone Telemetry, which has the function of persistence, preparing data to generate charts and to be analyzed by specific actors to perform diagnosis.

In this work, we used the AR.Drone 2.0. The proposed environment can be adapted to any drone, as long as a

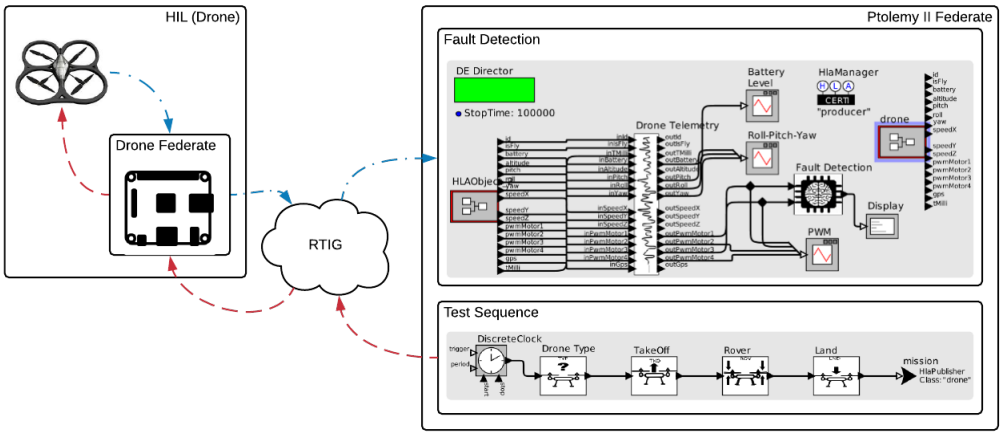


Fig. 1. Architecture of the Environment for UAV Testing

Drone Federate is developed specifically for the drone to be tested. In previous work, it was developed a system that can be embedded in any drone to command it and send telemetry data to a central computer [26]. In previous work, a Federate similar to that proposed here was developed, but for the monitoring of ground robots [6], [9].

Figure 1 shows an example of a concatenation of actors to define a mission to be performed by the drone. In this mission, the drone must: take off, rover, and land. Each movement actor has two parameters: speed, which defines how fast the action is performed, and time-in-seconds, which defines the duration of the movement in seconds. When starting the model, the actor Discrete Clock sends a signal to the first actor, the Drone Type, making the actor to transfer to the next actor a string containing an action to be carried out by drone (see Table I). The next actor in sequence upon receiving the action from the previous actor concatenates with its action and sends it to the next player in the sequence. In the end, a string containing a sequence of actions with all actors is forwarded to HLAPublisher actor, which publishes it through RTIG.

The Drone Federate (Figure 1) is implemented in a Raspberry Pi board (running the Raspbian operating system), and wi-fi adapter. The Raspberry can be replaced by a computer or another board model, which supports Python 2.0 language and the hardware needed to control the drone (if necessary). The board may be embedded in the drone, or not. This depends on the type of drone. For example, the ARDrone can be controlled externally via wifi and provides an API to control and data acquisition. The Drone Federate is implemented in DroneFederate.py (Figure 2 application written in Python 2.0. The DroneFederate.py application is responsible for receiving the test sequence published by Ptolemy Federate, control the drone for the execution of the defined mission and simultaneously collect and publish the telemetry data in the Federation.

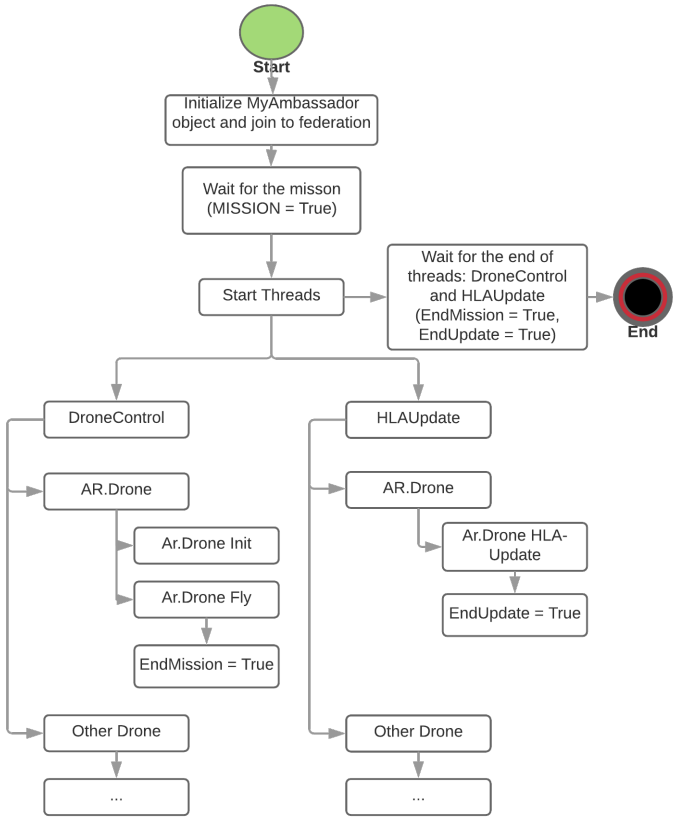


Fig. 2. Algorithm executed in the Drone Federate by the DroneFederate.py

The application uses two threads to perform these functions simultaneously. When running DroneFederate.py (Figure 2), initially three global variables are declared: 1) Mission, which receives the mission; 2) EndControl, which signals the end of the DroneControl; and 3) EndUpdate to signal the end of HLAUpdate thread. Following MyAm-

bassador object is created (class containing methods for connection with RTIG/HLA) and then the Drone Federate is included in the Federation. The program then waits for the publication (by Ptolemy Federate) of the string containing the mission of the drone. Upon receiving the mission, the Mission variable changes its value to the right, and the program execution continues. Then, two threads are initialized: Drone Control and HLAUpdate. Drone Control executes the movement of the drone, and HLAUpdate collects the telemetry data during the flight of the drone and publishes them to the Federation (to be read by Ptolemy Federate).

In Fault Detection module (Figure 1), the HLAObject Subscribe actor get the telemetry data published by Drone Federate and forwards them to the Drone Telemetry actor. The actor Drone Telemetry has the treatment function and persistence of the received data. The actor is composed of input and output ports corresponding to the attributes used by the HLA Drone class (Listing 1).

The actor receives the telemetry data from the input ports, saves them in CSV files and forwards them through the respective output ports. The received data is used to generate graphics, and for failure detection by an actor, which implements an algorithm based on machine learning.

Listing 1. Class Drone defined to be used by HLA for message exchanging.

```
1  (class drone
2    (attribute id reliable timestamp)
3    (attribute isFly reliable timestamp)
4    (attribute tMilli reliable timestamp)
5    (attribute battery reliable timestamp)
6    (attribute altitude reliable timestamp)
7    (attribute pitch reliable timestamp)
8    (attribute roll reliable timestamp)
9    (attribute yaw reliable timestamp)
10   (attribute speedX reliable timestamp)
11   (attribute speedY reliable timestamp)
12   (attribute speedZ reliable timestamp)
13   (attribute pwmMotor1 reliable timestamp)
14   (attribute pwmMotor2 reliable timestamp)
15   (attribute pwmMotor3 reliable timestamp)
16   (attribute pwmMotor4 reliable timestamp)
17   (attribute gps reliable timestamp)
18   (attribute mission reliable timestamp)
19 )
```

## IV. Experiments

To verify the relevance of the proposed environment, two full testing procedure, and diagnosis of a drone AR.Drone 2.0 was performed. For the first one, a simple mission was configured to be carried out by the drone. The mission consists of taking the drone off, staying in rover state for 2 minutes and land. For the second, the mission consists of taking the drone off, staying in rover state for 10 seconds, move the drone forward, staying in rover state for 2 seconds, move drone backward, staying in rover state for 2 seconds and land.

During the execution of the missions, the drone telemetry data is collected and published in the Federation. Actor Drone Telemetry has read these data in Ptolemy Federate. The actor stored the Drone Telemetry data and sent them to be analyzed by an actor named Fault Detection using an algorithm based on artificial intelligence, explicitly implemented for these experiments. In the end, this actor presents a diagnosis of the drone, based on the analyzed data. An image with the interface of the simulator in action can be seen in Figure 3. A complete video can be watched on http://tiny.cc/bmz99y.

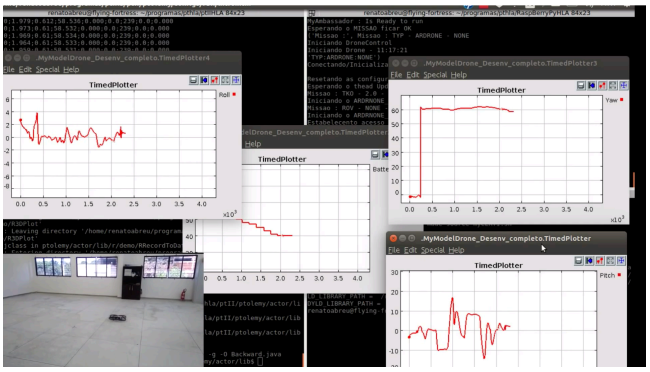


Fig. 3. Experiment in action with visualization of the drone, the system messages and graphics generated during the simulation.

The main objective of the experiments was to confirm whether it is possible, using the proposed environment, diagnose an anomalous behavior of the tested drone during the flight. For this, for each experiment, twenty flights were conducted without any abnormality inserted, and other twenty flights with an inserted an unbalanced propeller in motor 4. The anomaly was inserted with adhesive tape with 5 cm by 2 cm, on the edge of the propeller. This tape weights approximately 0.2 gram and introduces a subtle unbalance in the propeller. The flights were conducted in a closed environment without the interference of external factors such as wind. These data were used to Fault Detection actor training.

The Fault Detection actor implements a known machine learning algorithm called Decision Tree using scikit-learn library [27]. The main objective of the Fault Detection actor is to imediatly detect anomalies inserted to the drone, or not. The actor classifies the flight as stable (without fault) or unstable (with some anomaly). The algorithm has been trained with saved data from previous flights.

For each experiment, the data was divided as follows: 80% of the stable flights (no anomaly) and 80% of the bad flights (with the anomaly) for training, and 20% of stable flights and 20% of unstable flights for testing. To choose which algorithm would be used, a program was implemented to test the accuracy of the following algorithms: Decision Tree, Naive Bayes, and Support Vector Machine. This program was executed 1000 times. The best performance was achieved by the Decision Tree approach, with 85.3% hit rate. A comparison between the algorithms

can be seen in Table II.

TABLE II
COMPARISON OF MACHINE LEARNING TECHNIQUES FOR ANOMALY DETECTION

| Algorithm | Number of tests | Accurate Rate |
|---|---|---|
| Decision tree | 1000 | 85.3% |
| Naive Bayes | 1000 | 60.4% |
| Suport Vector Machines | 1000 | 70.7% |

After collecting data and training the Fault Detection actor, six flights were performed. The data collected during these flights were passed to the actor Fault Detection for the diagnosis. A single parameter was selected from telemetry to be analyzed (see Table III). After a sequence of graphical analysis, the Yaw parameter was chosen. This parameter gave a clear distinction between a flight with and without the inserted anomaly (Figure 4). Improve the hit rate of the diagnosis algorithm is not within the scope of this work, but offer the possibility to do that.

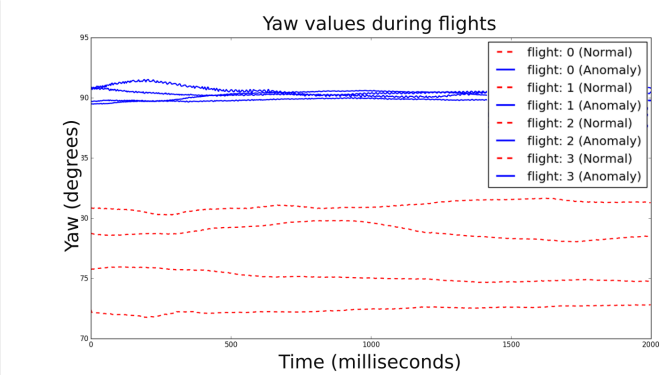


Fig. 4. Yaw values during flights

TABLE III
PARAMETERS OF THE DRONE UNDER ANALYSIS.

| # | Parameter | Method in AR.Drone | Description |
|---|---|---|---|
| 1 | Battery | getBattery()[0] | Returns the battery level |
| 2 | Pitch | NavData["demo"][2][0] | Get the pitch angle $\theta$ in mili-degrees |
| 3 | Roll | NavData["demo"][2][1] | Get the roll angle $\varphi$ in mili-degrees |
| 4 | Yaw | NavData["demo"][2][2] | Get the yaw angle $\psi$ in mili-degrees |
| 5 | Altitude | NavData["altitude"][3] | Returns the Drone Altitude |
| 6 | Linear Speed in X | NavData["demo"][4][0] | Get the estimated speed in X in m/s |
| 7 | Linear Speed in Y | NavData["demo"][4][1] | Get the estimated speed in Y in m/s |
| 8 | Estimated Linear Speed in Z | NavData["demo"][4][2] | Get the estimated speed in Z in m/s |

## V. RESULTS

As expected, the drone executed the sequence of movements defined in the test environment, and the data that was collected from drone was sent to the actor Drone Telemetry through HLA. For the first experiment, twenty flights of 60 seconds each were performed. During each flight, about 11,000 samples were collected containing all parameters listed in Table III, which means 183 samples per second. Half of these flights had no anomaly inserted and the other half with an anomaly. For the second experiment, twenty flights of 28 seconds each were performed. During each flight, about 3,500 samples were collected containing all parameters, which means 58 samples per second. Half of these flights had no anomaly inserted and the other half with an anomaly.

For the first experiment (Table IV), with flights without abnormality an accuracy rate of 60% was achieved, i.e., for 10 flights without inserted anomaly, 6 were correctly identified as stable, and 4 were incorrectly identified as unstable. For flights with abnormality the accuracy rate was 80%. For the second experiment, flights without abnormality achieved a 90% accuracy rate. For flights with abnormality was achieved an accuracy rate of 90%. The results show that on more complex missions, the rate of correct answers was higher. For a more accurate result, more flights would be necessary to train the algorithm, as well as adjustments in the algorithm configuration parameters (Decision Tree). We remind that the Fault Detection is not one of the main objectives of this work. We also observed that the battery cost for flights with the anomaly is higher than the flights without the anomaly, as can be observed in Figure 5.

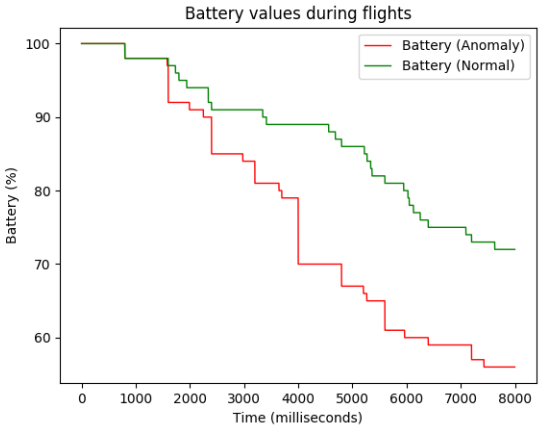


Fig. 5. Battery values during testing flight.

## VI. CONCLUSION

Due to the characteristics of the HLA, the environment could be integrated with any other drones or other simulation and analysis tools. The experiments satisfactorily

TABLE IV
Experiment 1 - Accuracy of Fault Detection actor

| Type | #flights | Hit Rate | Miss Rate |
|---|---|---|---|
| Stable | 10 | 60% | 40% |
| Unstable | 10 | 80% | 20% |
| Total | 20 | 70% | 30% |

demonstrated the operation and functionality of the environment. An accuracy rate of 70% was obtained even with a small number of training flights and without fine adjustments in the Decision Tree algorithm, neither any signal filtering.

As future work, we intend to focus on the use of the environment to implement tests that perform accurate diagnoses of drones. For this we intend to increase the historical database of drones, in order to train the actors for detection of failures; create new fault-detection actors based on other algorithms and other techniques; test new parameters, as well as combining them to increase the accurate rate; insert control modules for new drones, for example those controlled by ArduPilot; and increase the number of actions and include sensors, like ultrasonic and image capture.

## Acknowledgment

This work is supported by The Brazilian National Council for Scientific and Technological Development (CNPq) and the Higher Education Improvement Coordination (CAPES).

## References

[1] B. Michini, J. Redding, N. K. Ure, M. Cutler, and J. P. How, "Design and flight testing of an autonomous variable-pitch quadrotor," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 2978–2979.

[2] A. Patelli and L. Mottola, "Model-based real-time testing of drone autopilots," in *Proceedings of the 2Nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, ser. DroNet '16. New York, NY, USA: ACM, 2016, pp. 11–16. [Online]. Available: http://doi.acm.org/10.1145/2935620.2935630

[3] S. American, "5 epic drone flying failures—and what the faa is doing to prevent future mishaps," 2015, http://goo.gl/tIXfHH - Accessed June 2016.

[4] IEEE, "Std. 1516.2-2000. IEEE Standard for Modeling and Simulation (M & S) High Level Architecture (HLA) – Framework and Rules," 2000.

[5] A. Brito and T. Oliveira, "Simulation and test of communication in multi-robot systems using co-simulation," in *New Advances in Information Systems and Technologies*. Springer, 2016, pp. 911–917.

[6] J. C. V. Junior, A. V. Brito, L. F. S. Costa, T. P. Nascimento, and E. U. K. Melcher, "Testing real-time embedded systems using high level architecture," *Design Automation for Embedded Systems*, vol. 20, no. 4, pp. 289–309, 2016.

[7] J. C. VS, A. V. Brito, and T. P. Nascimento, "Verification of embedded system designs through hardware-software co-simulation," *International Journal of Information and Electronics Engineering*, vol. 5, no. 1, p. 68, 2015.

[8] T. W. Silva, D. C. Morais, H. G. Andrade, A. M. Lima, E. U. Melcher, and A. V. Brito, "Environment for integration of distributed heterogeneous computing systems," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 4, 2018.

[9] A. V. Brito, H. Bucher, H. Oliveira, L. F. S. Costa, O. Sander, E. U. Melcher, and J. Becker, "A distributed simulation platform using hla for complex embedded systems design," in *Distributed Simulation and Real Time Applications (DS-RT), 2015 IEEE/ACM 19th International Symposium on*. IEEE, 2015, pp. 195–202.

[10] R. Loh, Y. Bian, and T. Roe, "Uavs in civil airspace: Safety requirements," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 24, no. 1, pp. 5–17, Jan 2009.

[11] D. Stojcsics and A. Molnar, "Fixed-wing small-size uav navigation methods with hil simulation for aerobot autopilot," in *Intelligent Systems and Informatics (SISY), 2011 IEEE 9th International Symposium on*, Sept 2011, pp. 241–245.

[12] L. Yanjun, L. Yang, and Y. Shenglin, "Research on the algorithm of information fusion for height of uav," in *Intelligent Systems Design and Engineering Applications, 2013 Fourth International Conference on*, Nov 2013, pp. 523–526.

[13] L. Jaw, D. Homan, V. Crum, W. Chou, K. Keller, K. Swearingen, and T. Smith, "Model-based approach to validation and verification of flight critical software," in *Aerospace Conference, 2008 IEEE*, March 2008, pp. 1–8.

[14] Z. Wang, K. Akiyama, K. Nonaka, and K. Sekiguchi, "Experimental verification of the model predictive control with disturbance rejection for quadrotors," in *Society of Instrument and Control Engineers of Japan (SICE), 2015 54th Annual Conference of the*, July 2015, pp. 778–783.

[15] M. Ahsan, H. Rafique, and W. Ahmed, "Verification of equilibrium point stability for linearization of an aircraft model," in *Multi Topic Conference (INMIC), 2013 16th International*, Dec 2013, pp. 1–6.

[16] C. Yoo, Y. Kang, and B. Park, "Hardware-in-the-loop test for fault diagnosis system of tilt rotor uav," in *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, Oct 2008, pp. 320–323.

[17] G. Ducard, K. C. Kulling, and H. P. Geering, "Evaluation of reduction in the performance of a small uav after an aileron failure for an adaptive guidance system," in *2007 American Control Conference*, July 2007, pp. 1793–1798.

[18] H. B. Park, J. S. Lee, and K. H. Yu, "Flight evaluation of solar powered unmanned flying vehicle using ground testbed," in *Control, Automation and Systems (ICCAS), 2015 15th International Conference on*, Oct 2015, pp. 871–874.

[19] J.-S. Lee, H.-B. Park, G.-Y. Jung, and K.-H. Yu, "Design of virtual flight system for evaluation of solar powered uav," in *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*, Nov 2013, pp. 3463–3467.

[20] D. Deng and H. Yuan, "Uav flight safety ground test and evaluation," in *IEEE AUTOTESTCON, 2015*, Nov 2015, pp. 422–427.

[21] R. M. Vázquez, M. Romero, O. Portillo, J. C. Ávila, and A. H. Vilchis, "Experimental platform of a physical model for a quadrotor helicopter," in *Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2012 IEEE Ninth*, Nov 2012, pp. 311–314.

[22] "Using m&s hla in python," www.nongnu.org/certi/PyHLA, 2016, access: 2018-03-27.

[23] "Certi - summary," http://savannah.nongnu.org/projects/certi, 2016, access: 2018-03-27.

[24] "User manual for hla-ptii federates," https://download-mirror.savannah.gnu.org/releases/certi, 2015, access: 2018-03-27.

[25] D. Come, "Improving the hla-certi framework," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-202, Sep 2015. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-202.html

[26] V. S. Medeiros, R. E. Vale, Y. C. Gouveia, W. T. Souza, and A. V. Brito, "An independent control system for testing and analysis of uavs in indoor environments," in *Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR), 2016 XIII Latin American*. IEEE, 2016, pp. 55–60.

[27] (2018) scikit-learn machine learning in python. [Online]. Available: http://scikit-learn.org/stable/