Research Article

A Novel Software Simulator Model Based on Active Hybrid Architecture

Amr AbdelHamid¹ and Peng Zong²

¹College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics (NUAA),
29 Yudao Street, Nanjing 210016, China

²Astronautics College, NUAA, 29 Yudao Street, Nanjing 210016, China

Correspondence should be addressed to Amr AbdelHamid; amr@nuaa.edu.cn

Received 2 February 2015; Accepted 18 February 2015

Academic Editor: Ronald M. Barrett

Copyright © 2015 A. AbdelHamid and P. Zong. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The simulated training is an important issue for any type of missions such as aerial, ground, sea, or even space missions. In this paper, a new flexible aerial simulator based on active hybrid architecture is introduced. The simulator infrastructure is applicable to any type of training missions and research activities. This software-based simulator is tested on aerial missions to prove its applicability within time critical systems. The proposed active hybrid architecture is introduced via using the VB.NET and MATLAB in the same simulation loop. It exploits the remarkable computational power of MATLAB as a backbone aircraft model, and such mathematical model provides realistic dynamics to the trainee. Meanwhile, the Human-Machine Interface (HMI), the mission planning, the hardware interfacing, data logging, and MATLAB interfacing are developed using VB.NET. The proposed simulator is flexible enough to perform navigation and obstacle avoidance training missions. The active hybrid architecture is used during the simulated training, and also through postmission activities (like the generation of signals playback reports for evaluation purposes). The results show the ability of the proposed architecture to fulfill the aerial simulator demands and to provide a flexible infrastructure for different simulated mission requirements. Finally, a comparison with some existing simulators is introduced.

1. Introduction

Recently, enormous interest arises towards remotely controlled vehicles such as Unmanned Aerial Vehicles (UAVs), Unmanned Surface Vehicles (USV), Unmanned Ground Vehicles (UGV), Unmanned Underwater Vehicle (UUV), or even Unmanned Aerospace Vehicles. Many researches on unmanned vehicles have gained much attention worldwide [1–5]. This paper focuses on UAVs due to their tight time constraints compared to other unmanned vehicles. UAVs have been referred to in many ways: pilotless aircraft, drones, RPVs, and robot planes are few such names. UAVs are defined by the American Department of Defense (DOD) as “powered aerial vehicles that do not carry a human operator, use aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or non-lethal payload” [6]. Also, DOD states that “UAVs protect lives of pilots by performing the dull, dirty, or dangerous (3D) missions that

do not require a cockpit pilot” [6, 7]. Since good training leads to high mission success ratio, simulated training becomes necessary due to the high cost and impracticability of the actual vehicles training missions. Thus, the majority of the pilot training is directed to the simulated training. Many simulators are developed to fulfill the training and evaluation requirements. Therefore, the software design and engineering play great roles in the life cycle of training simulators.

The main objective of this paper is to design, implement, and evaluate a new software simulator based on simultaneous active hybrid architecture, via a minimized set of portable and convenient equipment. The new active hybrid architecture integrates the power of VB.NET and MATLAB in a unique manner in the field of aerial simulators. This integration is performed via utilizing the power of the MATLAB Component Object Model (COM) automation server technology. Different aspects are considered while building the proposed simulator which are as follows. (1) Implement the Model In-the-Loop Simulation (MILS) in a new hybrid manner.

(2) Exploit a comprehensive well defined flexible aerial mathematical model [8]. (3) Use simple equipment representation for validating the new hybrid architecture and to minimize the required budget [9]. (4) Achieve realistic innovative functional aerial simulator without AutoCode Generation as in [10]. (5) Achieve a robust design and infrastructure that could be implemented to any other types of unmanned vehicles, this could be done by changing the unmanned vehicle mathematical model to (USV, UGV, etc.). (6) Achieve a flexible design that can accept any future required modifications and it can be expandable to any level to help in many research fields (validation/evaluation testing). (7) Fulfill the portability requirement. (8) Minimize the maintenance procedures. (9) Develop a powerful high resolution data logging module for different playback purposes (on-screen or signals reporting).

Towards fulfilling this objective, the proposed simulator utilizes the Discrete Event Simulation (DES) [11], MILS, and active hybrid architecture. Due to high computational processing nowadays, it is feasible to run a powerful mathematical airframe engine with other developed software on a single workstation. Thus, the hybrid architecture is utilized to shorten the development process and gain maximum benefits from different environments. Moreover, the active hybrid architecture permits more than one environment to work simultaneously in the same simulation loop. In this paper, the De Havilland DHC-2 “Beaver” aircraft is selected as a backbone aerial mathematical model. The Beaver is a single-engine, high-wing, propeller-driven, versatile, short takeoff and landing (STOL) aircraft developed by De Havilland Canada, primarily known as a bush plane. Via solving rigid body Equation of Motion (EoM), and twelve Ordinary Differential Equations (ODEs), the Flight Dynamics and Control (FDC) toolbox can provide accurate simulation of the “Beaver” motion. The “Beaver” dynamics are obtained via exploiting a powerful MATLAB/Simulink open-loop mathematical model. This model is called “oloo1,” which allows the user to achieve complete flight parameters due to certain control inputs. Moreover, the VB.NET closes the control loop via a complete integration with the MATLAB engine. A standard Remote Control (RC) guidance device is interfaced with the proposed simulator package. The RC device implements a Hardware In-the-Loop Simulation (HILS) for a realistic imitation of real guided mission devices. Furthermore, the proposed simulator real time constraints are introduced for the important modules. The proposed hybrid simulator utilizes many advanced programming techniques like Application Programming Interface (API), threads, COM technology, datasets, stopwatches, and many other components.

The test procedures for the proposed simulator cover many characteristics like the following. (a) Analyze the new active hybrid architecture processing delay versus the simulator predefined time constraints. (b) Illustrate the HMI capabilities and simulated mission signals playback. (c) Test the flexibility of the proposed software package via an uprising modification in the mission requirements. The results show that the proposed active hybrid simulator is validated; the system real time constraints are achieved; the real time constraints safety margin is maintained; the trainee detailed activities are reported via the on-screen and signals

playback module; and the flexibility of the simulator design and architecture against future modifications is obtained.

Many difficulties and challenges have been conquered toward fulfilling the mentioned aspects of the simulator: (1) the handling of MATLAB engine interface should be accurate enough to close the simulation control loop. (2) The proposed simulator should manage all software/hardware resources efficiently to be able to satisfy the real time constraints. (3) The accurate evaluation requires high resolution logging module which is in the order of 10^{-7} seconds. (4) The data structure containers should be thread-safe. In other words, data structures are manipulated in a manner that guarantees safe execution of simultaneous threads.

Unlike [8–10, 12], the proposed simulator (1) focuses on increasing the functional fidelities and ease of development of the complete MATLAB/Simulink simulators through the integration of VB.NET. (2) It achieves the capability to use a variety of map types. (3) It gains an analysis capability via hybrid charting functionalities. (4) It is hosted on a single workstation for portability. (5) It utilizes the minimum required set of equipment to reduce the budget. (6) It does not require any migration to other packages. A comparison between the proposed simulator and these different simulators is illustrated in Section 4.4.

The rest of this paper is organized as follows: Section 2 presents the related work to different simulators. Section 3 details the proposed hybrid simulator model (the MATLAB mathematical model, the proposed VB.NET package, and the proposed active hybrid architecture integration). Sections 4 and 5 comprehensively present the simulator results and conclusions.

2. Related Work

Software Engineering plays a great role in the UAV development sector. For example, the software and hardware architectures are used in the design and implementation of a small semiautonomous fixed-wing UAV [13], while in [14] the development of a real time embedded on-board computer and Ground Control Station (GCS) software system for a UAV helicopter is introduced. The guidance, navigation, and control system of a small vertical-takeoff-and-landing (VTOL) unmanned tripropeller air vehicle based on a six-degrees-of-freedom nonlinear dynamic model is designed and developed in [15]. On the other hand, authors in [16] developed a systematic approach to Microaerial Vehicle (MAV) trajectory generation, and the characteristic issues of MAV flights in winds are addressed. A web access to real time flight data for a modern GCS is achieved, plus possessing the standard functions, such as data receiving in real time, storing the data, and displaying the data [17]. Moreover, the mission planning problem is solved to select the path which maximizes the collected information [18].

Reference [12] stated that “MATLAB is the tool of choice for many simulation applications, however it simply does not provide the real time high fidelity visualization or physical simulation necessary.” Additionally, it classifies MATLAB/Simulink simulators as they have low physical fidelity, medium functional fidelity, and medium ease of

development, where “Physical fidelity is defined as the degree to which the physical simulation looks, sounds, and feels like the operational environment,” while functional fidelity is defined as “the degree to which the simulation acts like the operational equipment in reacting to the tasks executed by the trainee.” Thus, to improve the functional fidelity of MATLAB/Simulink simulators, the integration of MATLAB with other development packages is required.

Different researchers utilize various types of hybrid architecture by mixing the capabilities of VB and MATLAB in many research fields like a lifetime prediction system for Vacuum Fluorescent Display (VFD), creation is done using the interface development of VB and merits of MATLAB in both calculation and display [19], and a variable-spray control system based on machine vision utilizes a VB application due to its high executive efficiency to collect real time information and control signals; meanwhile, MATLAB software is adopted for its excellent image processing function [20]. Moreover, VB and MATLAB as a hybrid programming method is utilized to develop a friendly man-machine interface for the heated oil pipeline shutdown and restart simulation software [21].

Since good training leads to a high mission success ratio, simulated training becomes necessary due to the high cost and impracticability of the actual vehicles training missions. The majority of the pilot training is directed to the simulated training. Therefore, a ground control station simulator with path planning utilizes the MATLAB/Simulink with FlightGear for better physical simulation and visualization [22]. A low cost simulator using three workstations, Head-Mount-Display (HMO), pilot control pedals set, and data gloves are built to test the viability of the use of virtual reality tools in flight simulators [9], while a fixed based flight simulator utilizes six workstations for visual display, instruments control, audio, real time simulation, control loading, and human-machine interface via the MATLAB AutoCode Generation to generate the design modules code to be exported to the RT-LAB which is integrated aperture with MATLAB [10].

In this paper, a new active hybrid simulator is proposed. Such simulator utilizes single workstation for running a MATLAB/Simulink backbone aircraft mathematical model, innovative HMI developed in VB.NET, real RC interfacing, and precise data logging module for recording all simulated mission inputs and outputs. The data recordings are not only used for on-screen playback, but also used for the testing and evaluation process through the signals playback.

3. Proposed Hybrid Software Simulator Model

Software applications have a great potential among the users and researchers, due to their simple installation procedures, low maintenance effort, and quit long mean time between failures (MTBF). The software applications have replaced many hardware modules/systems. Nowadays, the proposed simulator not only relies deeply on software, but also uses Hardware In-the-Loop Simulation (HILS) to provide realistic response to the mission training. The proposed simulator

FIGURE 1: Hybrid simulator block diagram.

uses a powerful “Beaver” mathematical model as a background computational engine and utilizes an innovative HMI console for user interaction.

The hybrid simulator block diagram is illustrated in Figure 1. It is clearly noticed that the vehicle mathematical model is considered as one of the main components of the proposed simulator model. This mathematical model provides the dynamics of the vehicle to the trainee. It works as an Attitude Heading Reference System (AHRS) plus a Global Positioning System (GPS) to let the trainee know the position and orientation of the aircraft. The mathematical model calculates the new position and orientation according to the trainee guidance commands. In addition, the HMI plays a great role of connecting all the components together and merges the calculated results in a single visual interface. The time constraints of the hybrid simulator modules are important, as they would be compared with the results. This comparison will check if the proposed simulator satisfies the real time constraints or not.

The proposed simulator consists of an innovative HMI to view the stationary and movable objects; therefore, the graphics refresh rate will be 30 frames/sec for smooth motion [23]. This means that the HMI will have an update cycle of 33 ms. The RC interface module will capture the trainee actions 30 times per second for accurate acquisition (30 guidance commands data frames every second). Therefore, the RC interface will have an update cycle of 33 ms. The maps resources will be updated according to the individual trainee actions outside the computation/control simulation loop (in a separate thread). This is considered as one of the conquered challenges in this paper. From all of the above mentioned constraints, it is clear that the real time constraint is 33 ms. In other words, all the simulator tasks processing time (RC acquisition, RC data extraction, open-loop model initialization, hybrid simulation, data gathering

and processing, graphical representation, and data recording) should be accomplished in 33 ms or less.

As shown in Figure 1, the proposed simulator model is divided into two main environments (MATLAB and VB.NET). The following subsection is concerned with the aircraft mathematical model, while, later, comprehensive details about the proposed software simulator package (VB.NET) will be presented.

3.1. MATLAB Mathematical Model Computation. This subsection provides a brief mathematical aircraft modelling knowledge and its set of equations. Initially, it is worthily noted that the core of the aerial mathematical model is a nonlinear model of the aircraft dynamics, consisting of twelve ODEs or state equations and a large number of output equations, as shown in the next subsection.

3.1.1. Nonlinear Airframe Model. The aircraft equations of motion are derived from basic Newtonian mechanics [24–27]. The rigid body force and the moment equations are stated as follows:

$$\begin{aligned}\mathbf{F} &= m \left(\frac{\partial \mathbf{V}}{\partial t} + \boldsymbol{\Omega} \times \mathbf{V} \right), \\ \mathbf{M} &= \frac{\partial (\mathbf{I} \cdot \boldsymbol{\Omega})}{\partial t} + \boldsymbol{\Omega} \times (\mathbf{I} \cdot \boldsymbol{\Omega}),\end{aligned}\quad (1)$$

where m is the mass of the aircraft and $\mathbf{V} = [u \ v \ w]^T$ is the velocity vector at the center of gravity (c.g.) where u , v , and w are the velocity component along the X_{Body} -axis, Y_{Body} -axis, and Z_{Body} -axis, respectively. $\boldsymbol{\Omega} = [p \ q \ r]^T$ is the angular velocity vector about the c.g. where p is the angular rate of roll, q is the angular rate of pitch, and r is the angular rate of yaw. $\mathbf{F} = [F_x \ F_y \ F_z]^T$ is the total external force vector where F_x , F_y , and F_z are the total external force along X_{Body} -axis, Y_{Body} -axis, and Z_{Body} -axis, respectively. $\mathbf{M} = [L \ M \ N]^T$ is the total external moment vector where L is total rolling moment, M is the total pitching moment, and N is the total yawing moment, and $\mathbf{I}$ is the inertia tensor of the rigid body and can be written as

$$\mathbf{I} = \begin{bmatrix} I_{xx} & -J_{xy} & -J_{xz} \\ -J_{yx} & I_{yy} & -J_{yz} \\ -J_{zx} & -J_{zy} & I_{zz} \end{bmatrix}. \quad (2)$$

In (2), the tensor coefficients are the moments and products of inertia of the rigid body. If the reference frame is fixed to the airframe, these values are constant and independent of the attitude of the vehicle. In order to use equations (1) and (2) for control system design and analysis, simulation purposes, and system identification, they need to be rewritten in nonlinear state-space format.

These dynamic equations form a state-space system which is valid for any rigid body (road vehicles, airframes, water surface vehicles, or spacecraft). The body-axes components of linear and rotational velocities could be described as the state variables, while the body-axes components of the

external forces and moments are the input variables of these equations. Typically, some sort of coupling between state variables themselves and the force and moment equations should take place. Although this makes the equations more complex, it is still possible to combine these equations in a nonlinear state-space system. The derivation continues till the complete output state vector $\mathbf{x}$ is reached. $\mathbf{x}$ consists of twelve elements: three linear velocities, three angular velocities (p, q, r), three Euler angles which define the attitude of the aircraft relative to the Earth (ψ, θ, ϕ), and two coordinates and the altitude which define the position of the aircraft (x_e, y_e, H) relative to the Earth. Typically, the true airspeed (V), angle of attack (α), and sideslip angle (β) can be used instead of the linear velocity components along the body-axes of the aircraft, yielding the following state vector:

$$\mathbf{x} = [V, \alpha, \beta, p, q, r, \psi, \theta, \phi, x_e, y_e, H]^T. \quad (3)$$

Thereafter, the gravitational, propulsive, and aerodynamic effects and the influence of nonsteady atmosphere should take place [25, 28]; this is done by continuing derivation towards the aerodynamic forces and moments (which are dependent on the flight conditions) and the external aerodynamic control inputs, defined by the input vector $\mathbf{u}_{\text{aero}}$. Consider

$$\mathbf{u}_{\text{aero}} = [\delta_e \ \delta_a \ \delta_r \ \delta_f]^T, \quad (4)$$

where δ_e , δ_a , δ_r , and δ_f are elevator deflection, aileron deflection, rudder deflection, and flaps deflection, respectively. These inputs are the deflections of the aerodynamic control surfaces. All the values of the stability and control coefficients for the Beaver aircraft are available in addition to the propulsive forces and moments derivations and coefficients [28].

3.1.2. Flight Dynamics and Control (FDC) Toolbox. The FDC toolbox implemented in MATLAB/Simulink makes it possible to analyze aircraft dynamics and flight control systems within one software environment (MATLAB) on one PC or workstation. This toolbox has been set up around a general nonlinear aircraft model, developed and constructed in a modular way to provide maximum flexibility for the user [8]. The FDC has three built-in open-loop simulation models. The proposed simulator is interested in the first open-loop model which is so-called “olooop1.” The original “olooop1” simulation model is used to obtain nonlinear aircraft responses due to certain control input signals. The control surfaces notation conventions (which are the inputs control signal to the olooop1 simulation model) are described in FDC documentation [8].

3.2. Proposed Software Simulator Package. Microsoft Visual Studio .Net is a perfect solution for building modern and fancy Human-Machine Interfaces (HMI). It is a complete set of development tools for building the following: ASP web applications, XML web services, desktop applications, and mobile applications. All of the supported programming languages (Visual Basic .NET, Visual C++, Visual C#.NET,

F#, and Visual J#) use the same Integrated Development Environment (IDE), which allows them to share tools and facilitates the creation of mixed-language solutions. Typically, VB.NET is a multiparadigm high level programming language; it provides the ability for developers to produce rapid and convenient software applications. It has become one of the famous visual integrated developing tools that run under Windows operating system [19]. Thus, VB.NET is selected to be the development language for the proposed software package. The proposed hybrid simulator is based on a modular concept which is not only used to reduce the cost [29], but also used to provide more flexibility for any design changes or any uprising modifications [30]. Since the proposed simulator extremely relies on software modules and components, the main development components for implementing the proposed simulator are demonstrated in the following subsections.

3.2.1. Human-Machine Interface (HMI) and Reference Maps Module. The main simulator HMI console uses intuitive display that is divided into three parts: the graphical area, the tabular area, and the status area. The graphical area contains all static/dynamic types of data (maps, airframe, waypoints, and mission paths) in a graphical form. The tabular area contains numerical data, controls, and gauges. The status area contains all important data and those must be in front of the trainee. The use of tab control and expanded group box extremely increases the reserved area for controls in tabular area.

The geographic reference map is one of the most important modules in the HMI console, which could be deduced from the word “reference.” It allocates all main important objects in the simulator (home coordinates, aircraft coordinates, waypoints, etc.). A unique decision must be made—during the design phase—to determine the following criteria: (i) the utilized map category (raster or digital maps) and (ii) the utilized map type (standard or custom map).

For globalization purpose, the geographic reference map should be in a standard format and style, for more compatibility with other commercial packages. A standard mapping module is developed to facilitate the use of maps, routing, and geocoding from free map providers. The maps module achieves a mouse move accuracy for the HMI console up to 10^{-6} degrees. Via the utilization of the map provider APIs, this module can achieve accessibility to the global map resources. Hereby, many functions are added to the HMI console like selecting the access mode (whether it is via server, cache, or both), enabling/disabling grid tile mode, showing/hiding marker points, allocating the exact target (known latitude and longitude) for calibration and verification process, finding any geocoding (city, area, or street), and finally adding any custom map as an overlay. All map setting actions are event driven, which means that they need action from the user to be initiated.

3.2.2. Proposed Mission Planning. Although the geographic map reference module is settled and established, the HMI standalone could not achieve a mission success. There is

no successful mission training without a powerful and accurate mission planning module. Therefore, whatever the type of the operation is (real/simulated or remotely piloted/autonomous), mission planning is an essential module. By using the developed map module capability, the proposed simulator gained two more important functions: the routing and the overlays. The routing supports navigation through map features by defining start, middle, and end points, while the overlays provide the capability of having extra graphical layers with transparent background over the main map view area. The overlays could use standard items or custom items. The mission planning of the simulator is divided into two categories: “mission designer” and “mission viewer.” Hereby, an uprising need for a database appears; therefore, the mission planning module utilizes the built-in visual studio datasets as a local in-memory database [31]. The dataset is used to save the training mission waypoints in memory and prepare them in an appropriate format for file saving.

The mission designer is used to allocate the mission start point coordinates, the ordered middle points, and the end point and finally save the designed mission data. Figure 2 illustrates the mission designer flowchart. The mission designer module accepts the waypoint coordinates as a click event on the graphical area, copies these coordinates to editable controls to allow user modifications (if any), adds the way point according to the user request to the data grid (which is bound to the dataset), and, if mission is completed, saves all data grid waypoints to a standard Keyhole Markup Language (KML) file [32].

On the other hand, the mission viewer is used to load the designed mission and add it as a transparent layer (overlay) on the geographic map area. This layer shows the mission trajectory in front of the trainee. As shown in Figure 3, the mission viewer flowchart is an event driven module. In other words, the mission viewer loads all the KML files (designed missions), waits for the trainee to select one of the missions, overlays the selected mission over the graphical area, and finally manipulates the zoom level to center the selected mission in the screen. The mission viewer is capable of overlaying any designed KML mission: not only the missions that are designed by the proposed mission designer, but also the missions that are designed by other geographical packages (like Google Earth) as long as these missions are in KML format. In other words, the utilization of KML files provides a bidirectional importing/exporting of designed mission with standard packages like Google Earth.

As shown in Figure 4, the mission designer accepts the user defined coordinate and adds it to tabular and graphical area, and meanwhile calculates the total route distance. The start point is illustrated in a big green balloon, the middle points are shown as small green balloons, and the end point is allocated as a big red balloon. The mission illustrated in Figure 4 is designed to test the trainee to fly over mainland of China. This mission starts at the home location (at Nanjing), through Hanggang, Guilin, and finally ends at Guangzhou.

3.2.3. Remote Control Guidance Device Interfacing. Adding hardware equipment to the simulation process provides more

FIGURE 2: Proposed mission designer flowchart.

trusted results on different scales. As most of the modern RC devices are interfaced via USB, an acquisition module is required to capture this USB data. Connecting such RC devices to the USB port adds a new Human Interface Device (HID) in the windows device manager.

Therefore, a complete module is developed to capture the RC data from the HID interface. This module is capable

FIGURE 3: Proposed mission viewer flowchart.

of interfacing the Walkera DEVO F7 remote control device with the proposed simulator efficiently. This interface is

FIGURE 4: Mission designer interface snapshot.

implemented in a separate thread to be independent of the simulator activities. This device is used as a real ground pilot guidance device. The RC data are captured every 33 ms for accurate ground pilot guidance commands sensitivity. Writing and reading the captured data are done via thread-safe containers, which is considered a defeated challenge during this paper. The complete flowchart of such interface thread is illustrated in Figure 5. It is clear that the RC interface thread is an endless one: it starts when the HMI application starts, detects the event which indicates that HID is connected, gets handle of the RC HID object, reads the HID device parameters to adjust the receive buffers, reads RC data message, extracts the trainee input commands from the message, saves the extracted commands to global memory, and repeats the loop for continuous capture of trainee actions. If the RC device is disconnected at any time, the thread will wait until the trainee connects the RC guidance device again.

3.2.4. Data Logging, Signals Playback, and On-Screen Playback Modules. Data logging module is an important and critical component as it is responsible for recording all incoming and outgoing traffic to/from the proposed simulator. The recording is performed in an accurate time stamping for all events. The resolution unit is measured in ticks (where 1 tick = 10^{-7} seconds). All simulator interfaces and flight parameters (MATLAB COM automation server, RC guidance device, and aircraft mathematical model state variables) are recorded. This recording is implemented via a separate thread to avoid any effect taking place against the simulator

FIGURE 5: RC guidance device interface flowchart.

activities. Therefore, this is considered one of the conquered challenges during the development of the hybrid simulator. Through the proposed hybrid architecture, MATLAB features are exploited via the utilization of not only the MATLAB computational capability, but also the charting and statistics capability. A new class (called *MfileCoder.vb*) is developed to write all the captured data as a column array. Once the mission is completed, this class writes the complete body of the M-file statistics code. Through this technique, the VB.NET application writes a complete MATLAB code for achieving a full mission signals playback. The utilization of this M-file will be mentioned in Section 3.3. Moreover, the recorded M-file is not only used for the signals playback, but also used for the on-screen playback (display replays for the accomplished simulated missions). This is done through a separate off-line module via the utilization of dynamic timers. Table 1 contains a sample of the recorded data for some inputs and outputs of the aircraft model and its processing delay.

3.3. Active Hybrid Architecture Integration. As VB.Net has some weaknesses such as the limited calculating function and finite drawing ones, it is not convenient to implement complex algorithm using VB.Net. Conversely, MATLAB was designed to perform mathematical computation and effective matrix calculation, to analyze and visualize data, and to facilitate the development of new software programs [33, 34]. However, it has less capability to generate human-computer interactive graphic interface that cannot be independent of MATLAB environment. Typically, the integration between these two packages creates extreme advantages with tiny minor weaknesses. As the proposed simulator software package is developed using VB.NET and the Beaver mathematical model is a MATLAB/Simulink toolbox, it is clear that a MATLAB interfacing should take place to establish the communication between these different environments [35, 36]. Dynamic Data Exchange (DDE), ActiveX Automation Server, Dynamic Link Library (DLL), and Component Object Module (COM) are a sample for some MATLAB interfaces [21, 37]. The appropriate technique selection depends on the hybrid communication scenario (which application is the master and which one is the slave). In other words, it depends on which platform is a server (slave) and which one is the client (master). In this situation, HMI console is the master (client), while the MATLAB engine/Simulink is the slave (server). Due to the nature of the simulator, the MATLAB automation server interface is used to create the communication link between the HMI console and its backbone mathematical computation engine [35, 36]. Thus, the original Beaver mathematical model should be modified as shown in Figure 6. This modification could be concluded as follows. (1) Inputs are changed to constants. (2) The output state variables are changed to workspace output ("ToWorkspace" Simulink component). These modifications are considered the first step towards the hybrid environment integration as each block has an identified name, and this name is used to control any block property through the COM interface connection.

The MATLAB COM automation server interface could be achieved by utilizing the "CreateObject" command, which

creates and returns a reference to a COM object. As in the illustrated example

```
MatLabInstance = CreateObject("Matlab.Application").
```

This command will activate a new instance from MATLAB command window. This instance is capable of sending and receiving the data from/to the workspace and also executing MATLAB commands, and receiving the command results. The set of functions to read and write data to any workspace of a MATLAB engine is provided by MATLAB interfaces and is assembled in Table 2 [36].

Figure 7 illustrates the active hybrid architecture overview of the proposed simulator, while the labeled numbers show the data flow of the hybrid integration process. Step (1) indicates the RC guidance device transmitted data, which are processed via the mentioned RC interface module. Step (2) shows the RC guidance device processed data (guidance commands) that is transferred to the HMI console. Step (3) presents the guidance commands that are transferred to the COM client to be ready to cross the environment barrier (transfer the data across the processes). Step (4) indicates that the guidance commands are transferred to the COM server at the MATLAB environment. Step (5) shows the guidance commands that are directed inside the MATLAB environment to the inputs of the Simulink (Beaver aircraft mathematical model). After the simulation of the mathematical model is executed, Steps ((6), (7), and (8)) feed the aircraft flight parameters back to the COM server and across the environment barrier to the HMI console again, respectively. Then at Step (9), the data logging module captures the received flight parameters, encapsulates them with the previously transmitted guidance commands (Step (3)) and saves them in a recording file structured in M-file format. Steps (1) to (10) are repeated until the simulated mission is completed. Once the mission is completed, Steps ((11), (12), and (13)) are triggered to request an execution of the recorded M-file. This command is transferred through the COM client to the COM server. Finally, the COM server executes the command and runs the M-file directly to view the signals playback. This accurate data flow management is considered one of the most important challenges that have been conquered during the development of the proposed hybrid simulator.

Actually, through this active hybrid architecture, the VB.NET acts as a master application, while the MATLAB engine acts as a slave engine. This master-slave relation is achieved via a complete integration between the two environments. This integration is implemented using the MATLAB COM automation server interface.

It is worthily noted that the HMI console (VB.NET environment) closes the open-loop simulation model (MATLAB environment) by capturing the outputs from the workspace, updates the aircraft position in front of the trainee, modifies the open-loop MATLAB mathematical model initial conditions, provides new control inputs, and orders a new simulation run (Steps ((3), (4), (5), (6), (7), and (8)) represent the closed loop simulation).

TABLE 1: Data logging module recording sample.

Start time (ticks)	End time (ticks)	Processing delay (ticks)	δ_a (rad)	δ_e (rad)	δ_r (rad)	N (rpm)	V (m/s)	α (rad)	β (rad)	X (m)	Y (m)	Z (m)
635551513677046257	635551513677099822	53565	-0.01792	-0.0016	0	1800	44.911	0.1469	-0.02238	145097.15	90800.57	1829.034
635551513677116261	635551513677187423	71162	-0.02103	-0.00351	0	1800	44.786	0.1499	-0.00207	145114.49	90758.09	1829.546
635551513677206266	635551513677265025	58759	-0.02249	-0.00495	0	1800	44.604	0.1525	0.015887	145131.24	90715.55	1830.279
635551513677286270	635551513677348200	61930	-0.02513	-0.00651	0	1800	44.359	0.1531	0.022863	145147.42	90673.02	1831.295
635551513677376276	635551513677432243	55967	-0.02734	-0.00736	0	1800	44.073	0.1531	0.023362	145163.05	90630.59	1832.531
635551513677446280	635551513677512156	65876	-0.02981	-0.00752	0	1800	43.775	0.1534	0.023468	145178.24	90588.32	1833.877
635551513677526284	635551513677591498	65214	-0.03112	-0.008	0	1800	43.482	0.1544	0.025542	145193.10	90546.26	1835.233
635551513677606289	635551513677666243	59954	-0.03194	-0.00864	0	1800	43.207	0.1554	0.028946	145207.72	90504.42	1836.530
635551513677686293	635551513677740500	54207	-0.032	-0.0096	0	1800	42.963	0.1567	0.032337	145222.20	90462.81	1837.713

FIGURE 6: Modified “oloop1” simulation model for hybrid environment integration.

TABLE 2: MATLAB automation server set of functions.

Function	Description
1 Execute	Execute MATLAB command in server
2 Feval	Evaluate MATLAB command in server
3 GetCharArray	Get character array from server
4 GetFullMatrix	Get matrix from server
5 GetWorkspaceData	Get any type of data from server
6 PutCharArray	Store character array in server
7 PutFullMatrix	Store matrix in server
8 PutWorkspaceData	Store any type of data in server
9 MaximizeCommandWindow	Display server window on Windows desktop
10 MinimizeCommandWindow	Minimize size of server window
11 Quit	Quit the MATLAB session

Figure 8 demonstrates the complete flowchart of the proposed hybrid simulator. It can be observed that most procedures are event driven (depends on the trainee activities). For example, if the instructor uses the proposed simulator to design a new training mission the proposed simulator processes this through reference connector number (1) to activate the flowchart in Figure 2. But if the trainee utilizes the simulator capabilities in a simulated mission, the lower part is activated to perform the active hybrid simulation with

the MATLAB mathematical model (represented as Steps (3) to (8) in Figure 7).

It is worthily noted that the flowchart connector references numbering (1, 2, 3, and 100) has no relation to the sequence numbering of Figure 7; these connector references' numbering refers to the same numbering in Figures 2, 3, and 5. Moreover, connector reference number 100 is assigned to a dedicated return point in the flowchart, while assigned to 100 to allow future expansion for the HMI modules.

Additionally, the hybrid architecture (VB.NET and MATLAB) is not only used in the simulated mission training, but also extended to the off-line functionalities like “signals playback.” As mentioned in Section 3.2.4, once the simulated mission is ended, the data logging module creates a recording file written in M-file syntax. Through the mentioned COM object instance, the HMI console is able to command the MATLAB engine to run the recorded file. The next sample command illustrates this action: result = MatLabInstance.Execute (“run(‘D:\RecordingMission_2015_02_14.m’)”).

This command shows how the HMI console (VB.NET) commands the MATLAB engine to execute the MATLAB command “run,” with attribute “D:\RecordingMission_2015_02_14,” which refers to the VB.NET file.

4. Results

As the Software Development Life Cycle (SDLC) procedures state that checkpoints and testing procedures are mandatory after module development [38–40], a performance analysis checkpoint is implemented to test the integration of all developed modules. This checkpoint contains many test procedures to evaluate the following: (a) the active hybrid architecture from real time point of view, (b) HMI capabilities

4.1. Hybrid Architecture Real Time Constraints Analysis. Owing the advent of high processing capabilities, it is

worthily noted that the workstation which is used to run this hybrid simulator has the following configuration: AMD FX-8350 Black Edition 8-Core 4.0 GHz/8 MB L2 cache and 8 MB L3 cache, 8 GB RAM clocked at 1866 MHz, 2 GB video card, and equipped with Windows 7 operating system, MATLAB, and the proposed simulator software. The technical specification of the utilized AMD CPU indicates

FIGURE 8: Proposed hybrid simulator complete flowchart.

that it can run eight simultaneous threads. Therefore, it is suitable to capture the RC guidance device, manage the MATLAB COM automation server, access the map provider's resources, visualize the HMI features, and store the data logging recordings simultaneously.

This subsection is interested in measuring the processing delay of the proposed active hybrid architecture with respect to the previously mentioned time constraints in Section 3. Via the illustrated data logging module, the VB.NET can measure the accurate consumed time of different MATLAB commands. As shown in Figure 9, 20000 sample measurements are plotted; these measurements represent MATLAB COM automation server commands execution times

(processing delay). For example, importing or exporting a variable from/to the workspace requires much less processing time than solving twelve ODEs (for simulating the flight parameters). The actual processing delay variations are shown in blue, while average MATLAB COM automation server processing delay is shown in red color.

It can be deduced that, using the active hybrid architecture technique, the average delay is in order of 7 ms, which means that the active hybrid architecture satisfies the simulator time constraints which is 33 ms. This means that the ratio of the average delay to the simulator time constraints is almost 1:5. Therefore, a smooth motion of the aircraft positions is achieved. Moreover, the dynamics of the aircraft

FIGURE 9: MATLAB COM automation server processing delay.

FIGURE 10: Proposed hybrid simulator HMI console.

in the simulated missions during the maneuvers are sensed via the illustrated set of gauges. The resultant platform satisfies the ground pilot training requirements. As the hybrid architecture satisfies the tight time constraints of the aerial vehicles, it could be extended to any other type of slower vehicles (road, sea surface, or even underwater vehicles). As

a result, it is valid to implement the active hybrid architecture for the proposed aerial simulator.

Figure 10 presents the proposed simulator HMI, which shows a sample training trajectory in light blue color in the graphic area, while the trainee is committed to perform a manual altitude hold mission. The mission starts at the home

FIGURE 11: Simulated mission signals playback.

logo, passes through six more middle waypoints, and finally ends with the red balloon point. The tabular area contains detailed output state variables of the mathematical model and also contains the MATLAB commanded simulation time

which equals the graphical area refresh rate (33 ms). Moreover, the illustrated set of gauges show all flight parameters in front of the trainee to have a complete emulation of the real mission. The attitude control shows the roll and pitch

FIGURE 12: Proposed hybrid simulator for an obstacle avoidance mission.

angles, the altimeter indicates that the aircraft flies at 5466 feet (equal to 1666 meters as shown in the tabular area). The air speed gauge indicates a true air speed at 86 knots, while the tabular area presents the airspeed in 44.6 m/s. Finally, the status area presents the status of the RC guidance device and the current zoom level, and it also provides the current mouse move coordinates with high resolution. In addition, the background data logging module is collecting all inputs and outputs parameters to be saved as a mission record for future analysis and evaluation process. It is clear that the proposed hybrid simulator functional fidelity is higher than the original MATLAB/Simulink FDC toolbox.

4.2. HMI Capabilities and Simulated Mission Signals Playback. Through this training mission, the trainee is committed to follow the trajectory and maintain constant altitude at 6000 feet (1828 meter). Figure 11 shows the signals playback for the whole mission. Figures 11(a), 11(b), and 11(c) show the aileron, elevator, and rudder deflections according to trainee RC guidance commands. Figures 11(d) to 11(i) show the most important signals from the aircraft mathematical model state variables. Through deep exploration of Figure 11, the trainee utilizes a lot of compensation along the mission. However, Figure 11(i) shows that the trainee maintains constant altitude for a long period; however, at simulation sample 3500, the trainee starts to lose the altitude. Due to the excessive use of compensation signals (shown in Figures 11(a), 11(b), and 11(c)) the trainee reaches the final destination waypoint at 5466 feet. The trainee maneuver is not smooth as shown from the variation of the compensation signals. Also, these compensation signals affect the aircraft angle of attack, side

slip angle, and airspeed, as shown in the variation in Figures 11(d), 11(e), and 11(f).

4.3. Proposed Hybrid Simulator Flexibility Analysis (Obstacle Avoidance Simulated Missions). The proposed hybrid simulator architecture should satisfy a quite wide range of modifications (for both VB.Net and MATLAB environments). This subsection demonstrates a nontraditional simulated training mission. As shown in Figure 12, the trainee is commanded to reach the waypoints along a designed mission. The simulated mission starts at big green balloon (start waypoint) at the south, through three middle waypoints, and finishes at the end waypoint at the top left. The light blue line shows the Line of Sight (LoS) path between the mission waypoints. However, this simulated training is not a regular one. The instructor uploaded an overlay obstacle map. The obstacle map is based on colored irregular shapes to indicate different obstacles heights. This map is overlaid on the HMI map using the developed reference map module which allows the loading of custom maps. Moreover, an additional circle of acceptance is represented as a red circle around the waypoint. This circle is overlaid to check that the waypoint is reached. Finally, the actual simulated mission trajectory is shown in black color.

To be able to evaluate the performance of the trainee, two extra error signals are calculated to check that the waypoint is reached. As observed in Figure 13, the next waypoint range and next waypoint azimuth are calculated in the VB.NET environment and added to the recording file. The trainee tries to minimize the next waypoint range (which means that the next waypoint is approaching as shown in Figure 13(a)) and, at the same time, tries to minimize the next waypoint azimuth to zero (which means that the waypoint is ahead Figure 13(b)).

FIGURE 13: Signals playback for obstacle avoidance simulated mission.

Throughout the visual interaction between the trainee and the HMI, the trainee uses the RC guidance commands to generate appropriate aileron and elevator deflections to guide the aircraft to the targeted waypoints as shown in Figures 13(c) and 13(d). Moreover, the variation of the airspeed, the angle of attack, and the side slip angle according to the trainee maneuvers are shown in Figures 13(e), 13(f), and 13(g).

From Figures 12 and 13, the trainee reaches waypoint 1 after 275 seconds as shown in Figures 13(a) and 13(b) and then starts a maneuver to let waypoint 2 ahead (by minimizing the azimuth error signal in Figure 13(b) to zero). After the azimuth is minimized to zero, at time of 495 seconds starts a maneuver with the green obstacle which causes a reincrement in the next waypoint error signal again. After accomplishing the maneuver with the obstacle at time instance 610 seconds, the trainee starts to minimize the azimuth error signal to zero to let waypoint 2 ahead. The trainee repeats the same process with another two obstacles before waypoint 3 and before the final destination.

It is worthily noted that, from Figure 12, the trainee tries to reach the waypoint from the shortest path; this is done by making close distance maneuvers with the obstacles (which shows that this trainee level is quite good). Moreover, Figures 13(c) to 13(g) are used to evaluate the trainee maneuver level. These signals show that all trainee commands are within the acceptable range of the aircraft acceptable deflections (Figures 13(c) and 13(d)), almost constant speed is reserved, and smooth maneuvers are maintained (Figures 13(f) and 13(g)). By comparing the signals playback in Figures 11 and 12, the trainee of Figure 11 is a novice one, while the trainee of the obstacle avoidance mission is considered an expert.

4.4. Proposed Hybrid Simulator versus Different Existing Simulators. As presented in Table 3, this subsection lists a brief comparison between the functions and features of the proposed hybrid simulator versus other different existing simulators.

TABLE 3: Proposed hybrid simulator versus other different existing simulators.

Criteria	Proposed hybrid software simulator	FDC toolbox version (1.3) [8]	SCALAB virtual reality simulator [9]	IEST fixed based flight simulator [10]
Flight model	Fixed wing De Havilland DHC-2 "Beaver"	Fixed wing De Havilland DHC-2 "Beaver"	Helicopter Uh-1h	IEST framework Internal flight model
Required workstations	1	1	3	10
OS	Windows	Windows	Windows	Windows QNX (optional)
Network infrastructure	Not required	Not required	Required	Required
Extra equipment	RC guidance device 1 LCD	1 LCD	Head mount display Data gloves Pilot control stick Motion tracker Pedals 1 LCD	Physical cabin Electrical cabinet, visual displays, platform Boarding bridge and Instructor Operation Station (IOS) SoundBox Instrument panel 4 LCDs
Portability	Portable	Portable	Portable	Fixed
Cost	Higher than FDC by the RC guidance device cost	Low cost	15000\$	Not explicitly mentioned
Utilized languages and packages	VB.NET and MATLAB/Simulink	MATLAB/Simulink	FlightGear	MATLAB RT-Lab
Required automatic code generation	Not required	Not required	Not required	Required
Simulation environment	Active hybrid environment (VB.NET and MATLAB)	Single environment	Single environment	MATLAB model migration to RT-Lab
HMI	Available (VB.NET)	Not available	Available	Available
Maps resources	Available (support map resources from global providers through API)	Not available	Available	Available
Mission planning	Supported	Not available	Not explicitly mentioned	Not explicitly mentioned
HILS	Available	Not available	Available	Available
Atmospheric models	Can use FDC models	Gusts models, turbulence, and wind-shear	Not explicitly mentioned	Not explicitly mentioned
Full mission recording, signals playback and on-screen playback	Available (VB.NET development)	Not available (single run only)	Available (FlightGear)	Not explicitly mentioned
Core model modification requirements	Modify in MATLAB; then immediately run the whole simulator features	Modify in MATLAB	Not explicitly mentioned	Modify MATLAB/Simulink model, AutoCode Generation for RT-Lab compilation, load model, and execute the whole simulator features
Architecture flexibility	Can be extended to support aerial or other different vehicle types (ground, surface, etc.)	Aerial vehicles only	Not explicitly mentioned	Not explicitly mentioned
Functional fidelity	Above medium	Medium	Medium	Not explicitly mentioned

5. Conclusions

This paper illustrates a new active hybrid architecture software simulator. It utilizes a comprehensive nonlinear aircraft mathematical model (MATLAB/Simulink) and a VB.NET developed package for trainee activities. All utilized components are mentioned, detailed flowcharts for software modules are presented, and the dataflow integration sequence is explained. A standard RC guidance device is interfaced for better trainee interaction and sensation. The simulated training process is recorded with high resolution data logging module (for on-screen playback activities and signals playback requirements). The utilized mathematical model is just a sample and could be modified or replaced by any other vehicle model according to the requirement. As the hybrid architecture satisfies the tight time constraints of the aerial vehicles, it could be extended to any other type of vehicles (road, sea surface, or even underwater vehicles) with quite good flexibility. The integration of the VB.NET with the MATLAB/Simulink increases the functional fidelity of the original FDC toolbox. Finally, a comparison between the proposed simulator, the original FDC toolbox, and other different types of simulators is presented.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to acknowledge the anonymous reviewers for their insightful comments and suggestions that led to significant enhancement of the paper quality. In addition, the authors would like to thank the Chinese government "National 863 fund" (ref. 2014AA7010051) for their great support.

References

- [1] Z. Liu, Y. Chen, B. Liu, C. Cao, and X. Fu, "HAWK: an unmanned mini-helicopter-based aerial wireless kit for localization," *IEEE Transactions on Mobile Computing*, vol. 13, no. 2, pp. 287–298, 2014.
- [2] A. A. Doshi, A. J. Postula, S. P. N. Singh, and A. Fletcher, "Development of control platform for micro-UAVs with integrated motion planning in wind," in *Proceedings of the 3rd Mediterranean Conference on Embedded Computing (MECO '14)*, pp. 28–31, Budva, Montenegro, June 2014.
- [3] V. Artale, M. Collotta, C. Milazzo, G. Pau, and A. Ricciardello, "An adaptive trajectory control for UAV using a real-time architecture," in *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS '14)*, pp. 32–42, Orlando, Fla, USA, May 2014.
- [4] Y. Lin, J. Hyypä, T. Rosnell, A. Jaakkola, and E. Honkavaara, "Development of a UAV-MMS-collaborative aerial-to-ground remote sensing system—a preparatory field validation," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 6, no. 4, pp. 1893–1898, 2013.
- [5] S. Ragi and E. K. P. Chong, "UAV path planning in a dynamic environment via partially observable markov decision process," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 4, pp. 2397–2412, 2013.
- [6] E. Bone and C. Bolkcom, *Unmanned Aerial Vehicles: Background and Issues for Congress*, The Library of Congress (Congressional Research Service), 2003.
- [7] Office of the Secretary of Defense, *Unmanned Aircraft Systems Roadmap 2005*, Department of Defense USA, 2005.
- [8] M. Rauw, *FDC 1.2—A Simulink Toolbox for Flight Dynamics and Control Analysis*, 2nd edition, 2001.
- [9] I. Yavrucuk, E. Kubali, and O. Tarimci, "A low cost flight simulator using virtual reality tools," *IEEE Aerospace and Electronic Systems Magazine*, vol. 26, no. 4, pp. 10–14, 2011.
- [10] S. Zheng, Z. Ye, J. Jin, and J. Han, "Fixed-based flight simulator development for human-machine interaction," in *Proceedings of the International Conference on Computer and Communication Technologies in Agriculture Engineering (CCTAE '10)*, pp. 408–411, 2010.
- [11] A. M. Law and D. M. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill Higher Education, 3rd edition, 2000.
- [12] J. Craighead, R. Murphy, J. Burke, and B. Goldiez, "A survey of commercial & open source unmanned vehicle simulators," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '07)*, pp. 852–857, IEEE, Rome, Italy, April 2007.
- [13] R. W. Beard, D. Kingston, M. Quigley et al., "Autonomous vehicle technologies for small fixed-wing UAVs," *Journal of Aerospace Computing, Information and Communication*, vol. 2, pp. 92–108, 2005.
- [14] M. Dong, B. M. Chen, G. Cai, and K. Peng, "Development of a real-time onboard and ground station software system for a UAV helicopter," *Journal of Aerospace Computing, Information and Communication*, vol. 4, no. 8, pp. 933–955, 2007.
- [15] R. Huang, Y. Liu, and J. J. Zhu, "Guidance, navigation, and control system design for tripropeller vertical-takeoff-and-landing unmanned air vehicle," *Journal of Aircraft*, vol. 46, no. 6, pp. 1837–1856, 2009.
- [16] Y. Ketema and Y. J. Zhao, "Micro air vehicle trajectory planning in winds," *Journal of Aircraft*, vol. 47, no. 4, pp. 1460–1462, 2010.
- [17] Y. Liu, B. Xian, F. Wang, S. Liu, and X. Gu, "Development of the Ground Control Station for a quadrotor unmanned aerial vehicle based on Java programming," in *Proceedings of the 31st Chinese Control Conference (CCC '12)*, pp. 5002–5007, July 2012.
- [18] H. Ergezer and K. Leblebicioglu, "Path planning for UAVs for maximum information collection," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 1, pp. 502–520, 2013.
- [19] J. Zhang, R. Wang, and L. Pan, "The system of lifetime prediction for VFD based on VB and MATLAB," in *Proceedings of the International Conference on Artificial Intelligence and Computational Intelligence (AICI '09)*, pp. 59–62, November 2009.
- [20] L. Chen, X. Bai, and X. Zhao, "Software design of variable-spray control system based on VB and MATLAB," in *Proceedings of the 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC '11)*, pp. 2398–2401, August 2011.
- [21] C. Li, X. Wu, and W. Jia, "Research of heated oil pipeline shutdown and restart process based on VB and MATLAB," *International Journal of Modern Education and Computer Science*, vol. 2, no. 2, pp. 18–24, 2010.

- [22] A. Ajami, J.-F. Balmat, J.-P. Gauthier, and T. Maillot, "Path planning and Ground Control Station simulator for UAV," in *Proceedings of the IEEE Aerospace Conference (AERO '13)*, pp. 1–13, March 2013.
- [23] P. McFedries, *Computers Simplified*, vol. 2, John Wiley & Sons, 2007.
- [24] E. L. Duke, R. F. Antoniewicz, and A. K. D. Krambeer, "Derivation and definition of a linear aircraft model," NASA Reference Publication 1207, 1988.
- [25] B. Etkin and L. D. Reid, *Dynamics of Flight: Stability and Control*, John Wiley & Sons, New York, NY, USA, 1996.
- [26] D. McLean, *Automatic Flight Control Systems*, Prentice Hall, 1990.
- [27] D. T. McRuer, D. Graham, and I. Ashkenas, *Aircraft Dynamics and Automatic Control*, Princeton University Press, Princeton, NJ, USA, 1973.
- [28] R. T. H. Tjee and J. A. Mulder, "Stability and control derivatives of the De Havilland DHC-2 'Beaver' aircraft," Internal Report, Delft University of Technology, Faculty of Aerospace Engineering, Section Stability and Control, 1988.
- [29] W. S. Davis and R. H. Fisher, *COBOL: An Introduction to Structured Logic and Modular Program Design*, Addison-Wesley, 1979.
- [30] B. S. El-Haik and A. Shaout, *Software Design for Six Sigma: A Roadmap for Excellence*, Wiley, 2010.
- [31] Y. Bai, "Data selection query with visual basic.NET," in *Practical Database Programming with Visual Basic.NET*, pp. 241–416, John Wiley & Sons, 2012.
- [32] A. Chen, G. G. Leptoukh, and S. J. Kempler, "Using KML and virtual globes to access and visualize heterogeneous datasets and explore their relationships along the A-Train tracks," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 3, no. 3, pp. 352–358, 2010.
- [33] M. H. Trauth, *MATLAB Recipes for Earth Sciences*, Springer, Berlin, Germany, 3rd edition, 2010.
- [34] M. Trauth and E. Sillmann, *MATLAB and Design Recipes for Earth Sciences*, Springer, Berlin, Germany, 2013.
- [35] I. The MathWorks, *Application Program Interface Guide, Version 5*, The MathWorks, Inc., 1998.
- [36] The MathWorks, *Matlab External Interfaces*, The MathWorks, Natick, Mass, USA, 2013.
- [37] L. Hong and J. Cai, "The application guide of mixed programming between MATLAB and other programming languages," in *Proceedings of the 2nd International Conference on Computer and Automation Engineering (ICCAE '10)*, pp. 185–189, February 2010.
- [38] A. M. Davis, E. H. Bersoff, and E. R. Comer, "Strategy for comparing alternative software development life cycle models," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1453–1461, 1988.
- [39] H. Hashimi, A. Hafez, and M. Beraka, "A novel view of risk management in software development life cycle," in *Proceedings of the 12th International Symposium on Pervasive Systems, Algorithms and Networks (ISPAN '12)*, pp. 128–134, 2012.
- [40] D. Horie, T. Kasahara, Y. Goto, and J. Cheng, "A new model of software life cycle processes for consistent design, development, management, and maintenance of secure information systems," in *Proceedings of the 8th IEEE/ACIS International Conference on Computer and Information Science (ICIS '09)*, pp. 897–902, June 2009.