COBEM 2021 26th International Congress of Mechanical Engineering
ABCM Associação Brasileira de Engenharia e Ciências Mecânicas
November 22-26, 2021, Virtual Congress, Brazil

COB-2021-1335
HIL PLATFORM FOR FIXED-WING AUTOPILOT – A TUTORIAL

Waldenê de Melo Moura
Instituto Tecnológico de Aeronáutica, PPG-CTE, Área de Sistemas Espaciais, Ensaios e Lançamentos.
Praça Marechal Eduardo Gomes, 50, Vila das Acácias.
12.228-900 - São José dos Campos, SP – Brasil.
waldene@ita.br

Neusa Maria Franco de Oliveira
Instituto Tecnológico de Aeronáutica, DEE, Departamento Eletrônica Aplicada.
Praça Marechal Eduardo Gomes, 50, Vila das Acácias.
12.228-900 - São José dos Campos, SP – Brasil.
neusa@ita.br

Alison de Oliveira Moraes
Centro Técnico Aeroespacial, Instituto de Aeronáutica e Espaço.
Praça Marechal Eduardo Gomes, 50, Vila das Acácias.
12.228-900 - São José dos Campos, SP – Brasil.
aom@ita.br

Abstract. *Due to the growing interest in drones and Unmanned Aerial Vehicles (UAV), there was a need to have a tool that helped in the implementation of the basic aspects of an autopilot. Thus, this work aims to develop a tutorial that describes the steps for the implementation of a Hardware in the Loop (HIL) platform of autopilot embedded in a micro controller dedicated to control a simulated fixed-wing aircraft in a Matlab Simulink® programming environment. For this purpose, the control and guidance systems of an Autopilot implemented from the linearized equations of the mathematical model, in steady state (trimmed), of the fixed-wing aircraft model Piper J-3 Cub ¼ were loaded into the micro controller. The embedded system was made available for the Arduino platform and the communication between the HIL and the simulated aircraft takes place through the asynchronous serial communication protocols. All implementation steps are described and made available on the GitHub platform allowing the end user to be able to reproduce the experiment and adapt it according to their needs. The results obtained are presented with successful simulated flight missions, being established through predefined waypoints to be reached by the aircraft.*

Keywords: *HIL, Autopilot, UAVs, Embedded Systems, Arduino.*

1 INTRODUCTION

With the increasing advancement in the research and development of drones and UAV, HIL tests for autopilots embedded in microcontrollers have been widely used. (Cook, 2013; Cai *et al.*, 2009)

Lizarraga *et al.* (2009) presents the preliminary results of a rapidly reconfigurable autopilot for small unmanned aerial vehicles designed to be programmable by Matlab Simulink by transferring the models directly to the autopilot via the code generation feature. Schulz *et al.* (2020) presents a test bench designed for the development of flight control systems for small unmanned aerial vehicles with sensors in a real-time simulation environment.

Santos *et al.* (2011) describes a complete simulation environment developed for testing and validating UAV lateral and longitudinal control systems. The environment consists of an autopilot embedded in a microcontroller and a PC running the X-Plane flight simulator. Communication between the two environments is done remotely by radio frequency.

Santos (2018) made all the theoretical development of an HIL implementing an autopilot through the control loops related to the lateral-directional and longitudinal dynamics of the aircraft in question. This implementation was embedded in a Stellaris hardware controlling a simulated aircraft through Matlab Simulink® and, as an alternative, the X-Plane flight simulator.

Despite the references cited above provide all the necessary information required to reproduce the experiments, to effectivelly repeat them one has to invest a few hours in development and communication tests, even considering that the user has a good experience in programming languages. So, this work contributes as a tutorial detailing from the establishment of communication between the simulated aircraft and the microcontroller, to the steps necessary to board the code of the control loops and guidance algorithm in the microcontroller and, finally, running the full control loop, with the autopilot (guidance and control blocks) being responsible for making the aircraft (mathematical model running in Matlab) perform a mission defined by way-points. This tutorial is based in the complete theoretical development and practical implementation detailed by Santos (2018). The codes of the entire embedded system, implemented in microcontrollers of the Arduino family, and the modeled fixed-wing aircraft system are available on the GitHub platform (Moura 2021a, 2021b) allowing the end user to reproduce the experiment and adapt it according to their needs.

2 SYSTEM OF AUTOPILOT FOR UAV

The autopilot for an autonomous vehicle can be represented as in Figure 1. These blocks are implemented using electronic components embedded in the aircraft, such as sensors, actuators and microprocessors/microcontrollers, and the software running in them. In this work, the autopilot for a fixed-wing aircraft was considered, being composed of these generic blocks.

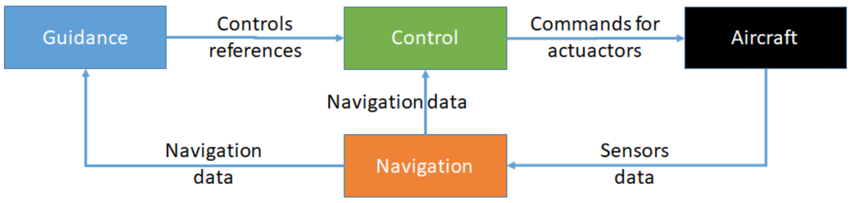


Figure 1. Autopilot blocks.

The Navigation block determines the aircraft's states (position, speed and acceleration) in relation to a given reference frame based on data collected by sensors. This information is treated and made available in navigation data for the Guidance and Control blocks.

The Guidance block determines the best trajectory and/or the reference signals to be reached by the vehicle to execute a desired maneuver. It can be responsible for planning and generating trajectory references to be used by the Control block for the aircraft to execute a desired mission or simple as to compute reference signals to reach predefined waypoints.

The Control block calculates the inputs for the actuators of the moving components (elevator, aileron, rudder and throttle) in order to stabilize the aircraft and effectively execute the desired maneuver.

In this work, the sensors data arrives directly at the Navigation block in an ideal way. With this, the autopilot includes, through programming, the functions of the Guidance and Control block. This implementation was embedded in low-cost microcontrollers from the Arduino platform (Arduino, 2021), being tested on microcontrollers ATmega328P (Arduino UNO) from Microchip (Atmega328p, 2020), Stm32f103c8 from STM Microeletronics (Stm32f103x8, 2015), Esp8266 and Esp32 from Expressif Systems (Esp8266, 2020; Esp32, 2021). For that, the embedded programs have been implemented, in their basic programming standards, for the Serial communication protocol meeting all the microcontrollers mentioned above.

The Aircraft block represents the fixed-wing aircraft to be simulated using Matlab Simulink®. In this work, the parameters of a Piper J-3 Cub ¼ whose measurements have a quarter of the scale of the original measurements of the Piper J-3 Cub produced by “Piper Aircraft” (Santos, 2018) were used.

The structure of an unmanned fixed-wing aircraft has two types of components: fixed and mobile. Fixed components include the fuselage, wings, propulsion engines, landing gear, and vertical and horizontal stabilizers. The moving components consist of the aircraft's control surfaces that suffer deflection: the ailerons (δ_a), the elevator (δ_e) and the rudder (δ_r), which, when deflected, causes variations in the lift forces on one or more control surfaces of the aircraft, allowing the change in the aircraft’s attitude (Stevens *et al.*, 2016). Another component is the actuator: the lever or throttle (Π), which controls the aircraft's speed in relation to the air. (Jaw and Mattingly, 2009).

In this work, the linearized mathematical model that describes the equations of motion of the fixed-wing aircraft Piper J-3 Cub ¼ developed by Santos (2018) was used. This linearization used the stead flight conditions (trimmed). In this way, all the aircraft's translation and rotation accelerations are set to zero. (Stevens *et al.*, 2016).

$$0=\dot{\vec{x}}=f(\vec{x}_0,\vec{u}_0) \quad (1)$$

The points of steady flight conditions were calculated and the values of the variables state is presented in Table 1.

Linearized models of conventional fixed-wing aircraft, in general, are divided into two systems that represent the longitudinal and lateral-directional dynamics of the aircraft. This work was developed by designing the control loops for the two systems.

Table 1. Steady flight points (Santos, 2018).

	u_0	w_0	θ_0	H_0
Values	0	0	0	400
Units	m/s	m/s	rad	m
	Π_0	δ_{e_0}	δ_{a_0}	δ_{r_0}
Values	4.16	-0.0703	0	0
Units	%	rad	rad	rad

2.1 Longitudinal control loop

In order to control longitudinal dynamics, the pitch stability increase system (θ) and the combination of speed autopilot and pitch angle (V_T, θ) with altitude autopilot (H) was used. Thus, the speed and altitude controllers act in parallel with each other, allowing the control system not to perform sudden movements and, after reaching the desired reference values, the aircraft returns to the state of steady flight. (Cook, 2013).

The block diagram in Figure 2 represents the implemented control of the longitudinal dynamics. More detailed information can be seen in Santos (2018, p. 53).

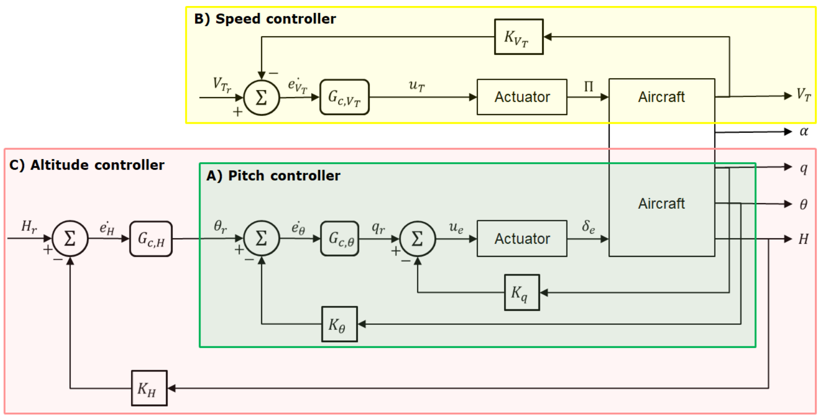


Figure 2. Longitudinal Dynamics: (a) pitch controller , (b) speed controller and (c) altitude controller.

The implementation of the pitch stability increase controller, shown in Figure 2A, is done through the continuous time transfer function of the dynamic compensator $G_{c,\theta}(s)$ given by Eq. (2).

$$G_{c,\theta}(s)=C_\theta K_p+C_\theta K_i\frac{1}{s}=\frac{-0.4532\,s+0.6662}{s} \tag{2}$$

The discrete transfer function, using a bilinear transformation by Tustin's method (Franklin *et al.*, 2013; Stevens *et al.*, 2016) with a sampling period of $T_s=0.01\ s$, is given by Eq. (3).

$$G_{c,\theta}(z)=\frac{Y_{c,\theta}(z)}{U_{c,\theta}(z)}=\frac{-0.4565\,z+0.4499}{z-1}=\frac{-0.4565+0.4499\,z^{-1}}{1-z^{-1}} \tag{3}$$

Therefore:

$$Y_{c,\theta}(z)=Y_{c,\theta}(z)\,z^{-1}-0.4565\,U_{c,\theta}(z)+0.4499\,U_{c,\theta}(z)\,z^{-1} \tag{4}$$

Converting Eq. (4) to a differences equation by defining z as a shift operator gives:

$$Y_{c,\theta}[z]=Y_{c,\theta}[z-1]-0.4565\,U_{c,\theta}[z]+0.4499\,U_{c,\theta}[z-1] \tag{5}$$

Which can be rewritten as:

$$Y_{c,\theta}[z+1]=Y_{c,\theta}[z]-0.4565\,U_{c,\theta}[z+1]+0.4499\,U_{c,\theta}[z] \tag{6}$$

Equation (6) is the Pitch Stability Increase Controller equation to be implemented in the microcontroller.

The implementation of the speed controller, shown in Figure 2B, is done through the continuous time transfer function of the dynamic compensator $G_{c,V_T}(s)$ given by:

$$G_{c,V_T}(s)=CV_t K_p+CV_t K_i+\frac{1}{s}=\frac{0.0303\,s+0.022}{s} \tag{7}$$

Developing Eq. (7) following the same steps described above, you have:

$$Y_{c,V_T}[z+1]=Y_{c,V_T}[z]+0.03041\,U_{c,V_T}[z+1]-0.03019\,U_{c,V_T}[z] \tag{8}$$

Equation (8) is the speed controller equation to be implemented in the microcontroller.

Finally, the implementation of the altitude controller, shown in Figure 2C, is done through the continuous time transfer functions of the dynamic compensator $G_{c,\theta}(s)$, described by Eq. (6), added in cascade with the dynamic compensator $G_{c,H}(s)$, whose equation is given by:

$$G_{c,H}(s)=C_h K_p+C_h K_i\frac{1}{s}=\frac{0.0172\,s+0.006}{s} \tag{9}$$

Developing Eq. (9) following the same steps described above, you have:

$$Y_{c,H}[z+1]=Y_{c,H}[z]+0.01723\,U_{c,H}[z+1]-0.01717\,U_{c,H}[z] \tag{10}$$

Equation (10) added to Eq. (6) form the altitude controller equation to be implemented in the microcontroller.

2.2 Lateral-directional control loop

For lateral-directional control, the yaw angle is used to keep the aircraft pointing to a desired direction (compass heading) through the feedback of the yaw angle (ψ) and the use of the roll angle (ϕ) thus keeping the aircraft's wings level, generating specific variations in the roll angle allowing coordinated turns with minimal slippage.

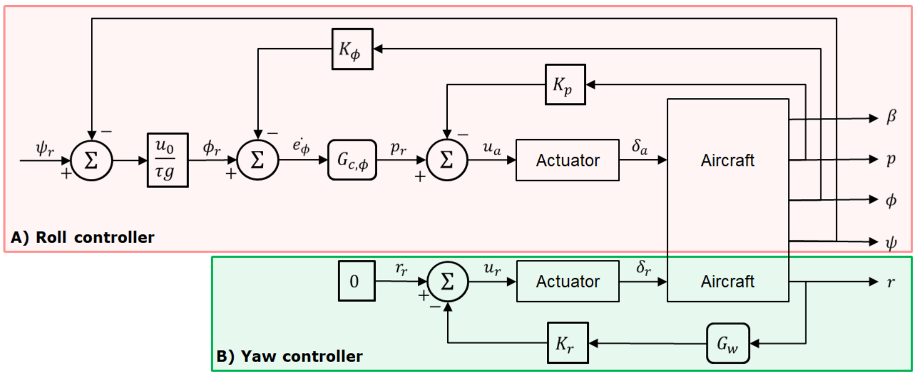


Figure 3. Lateral-directional diagram: (a) roll controller and (b) yaw controller.

The block diagram in Figure 3 represents the implemented control of the lateral-directional dynamics. More details can be seen in Santos (2018, p.65).

The implementation of the roll angle controller, shown in Figure 3A, is done through the continuous time transfer function of the dynamic compensator $G_{c,\phi}(s)$ given by:

$$G_{c,\phi}(s)=C_{phi}K_p+C_{phi}K_i\frac{1}{s}=-0.0907-\frac{0.02}{s} \tag{11}$$

Developing Eq. (11) following the same steps described above, you have:

$$Y_{c,\phi}[n+1]=Y_{c,\phi}[n]-0.0908\,U_{c,\phi}[n+1]+0.0906\,U_{c,\phi}[n] \tag{12}$$

Equation (12) is the roll angle controller equation to be implemented in the microcontroller.

The implementation of the yaw angle controller, shown in Figure 3B, is done through the continuous time transfer function of the dynamic compensator $G_w(s)$ given by:

$$G_w(s)=-K_r\frac{s}{s+1}=\frac{-0.105\,s}{s+1} \tag{13}$$

Developing Eq. (13) following the same steps described above, you have:

$$Y_w[n+1]=0.99\,Y_w[n]-0.1045\,U_w[n+1]+0.1045\,U_w[n] \tag{14}$$

Equation (14) is the yaw angle controller equation to be implemented in the microcontroller.

3 AUTOPILOT DEVELOPMENT FOR UAV

The development took place in two stages. In the first stage, it was executed communication testing and configuration of the serial communication protocol between the microcontroller and the simulated aircraft in Matlab Simulink®. Once configured and tested, the HIL platform's autopilot implementation followed.

3.1 Serial communication test

For the implementation using the serial communication protocol, any microcontroller that meets the Arduino platform can be chosen. For the communication test and configuration of the serial protocol, a system in C++ language was developed to be embedded in the microcontroller and a system in Matlab Simulink® to simulate the aircraft. Everything has been made available in a repository on GitHub (Moura, 2021a) which can be accessed publicly. To do this, download the “Serial_Communication” folder and load the “HIL_Serial_Test.ino” file for the chosen microcontroller. After that, the file “Hil_Serial_Test.slx”, Figure 4, is loaded in Matlab Simulink® environment.

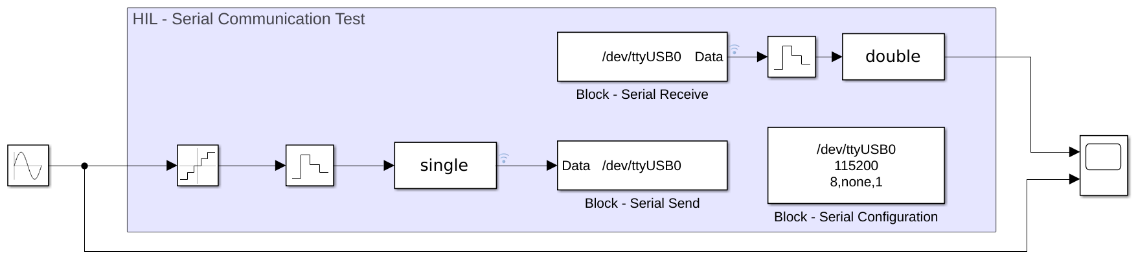


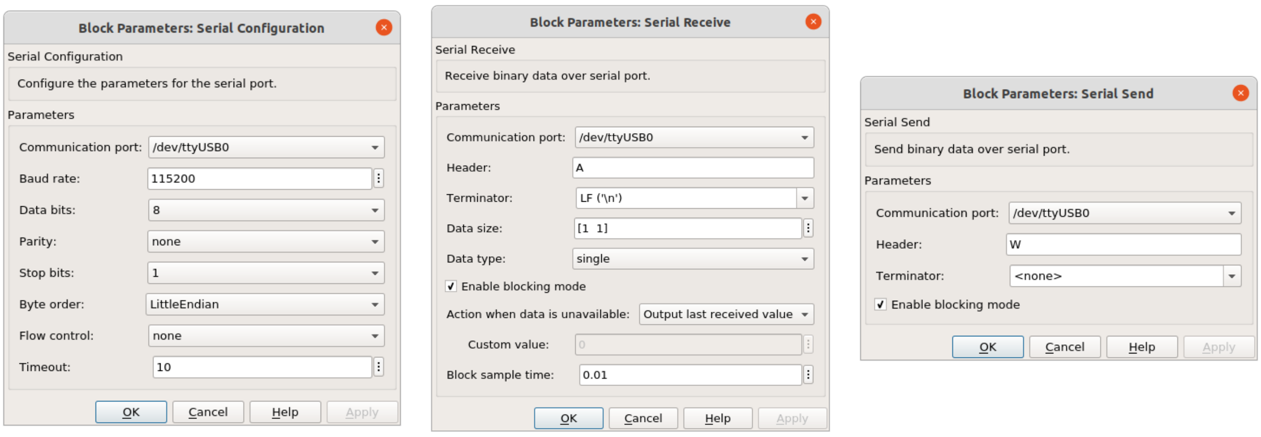
Figure 4. Simulink block diagram for the serial communication test (HIL_Serial_Test.slx).

At this point, it has the “HIL_Serial_Test.ino” file loaded in the microcontroller and the “HIL_Serial_Test.slx” in the Matlab Simulink® environment. To perform the communication between them, the “Serial Configuration”, “Serial Receive” and “Serial Send” blocks were used, which serve, respectively, to configure the serial port, to receive and to send data through the selected serial port. Figure 5 shows the settings made in these blocks.

Attention must be paid to the “Communication Port” parameter, which, depending on the operating system, will indicate the port to be used for communication. "COM**x**" for Windows or "/dev/ttyUSB**x**" for Linux. The letter **x** represents the port number provided by the operating system. Once the microcontroller is connected, these values can be selected directly in the “Communication Port” parameter. Other parameters to be observed are the “Baud Rate”, “Header” and “Terminator”, configured with their default options according to Figure 5.

If the communication between the microcontroller and Matlab Simulink® is working correctly, the graph in Figure 6 will be presented. That is, considering the Figure 4, the continuous-time input signal generated by Matlab Simulink® (Figure 6A), is converted to a digital signal and sent to the microcontroller. The microcontroller reads this signal and returns it to the Matlab Simulink®, which displays it on the oscilloscope (Figure 6B). Note that the returned signal (detailed in the enlarged image) is a quantized signal.

More information about the settings of the communication blocks and their parameters can be found in Mariga (2019).



Block Parameters: Serial Configuration

Serial Configuration

Configure the parameters for the serial port.

Parameters

Communication port:	/dev/ttyUSB0
Baud rate:	115200
Data bits:	8
Parity:	none
Stop bits:	1
Byte order:	LittleEndian
Flow control:	none
Timeout:	10

Block Parameters: Serial Receive

Serial Receive

Receive binary data over serial port.

Parameters

Communication port:	/dev/ttyUSB0
Header:	A
Terminator:	LF ('\n')
Data size:	[1 1]
Data type:	single
✓ Enable blocking mode	
Action when data is unavailable:	Output last received value
Custom value:	0
Block sample time:	0.01

Block Parameters: Serial Send

Serial Send

Send binary data over serial port.

Parameters

Communication port:	/dev/ttyUSB0
Header:	W
Terminator:	<none>
✓ Enable blocking mode	

Figure 5. Configuration parameters in Matlab Simulink®.

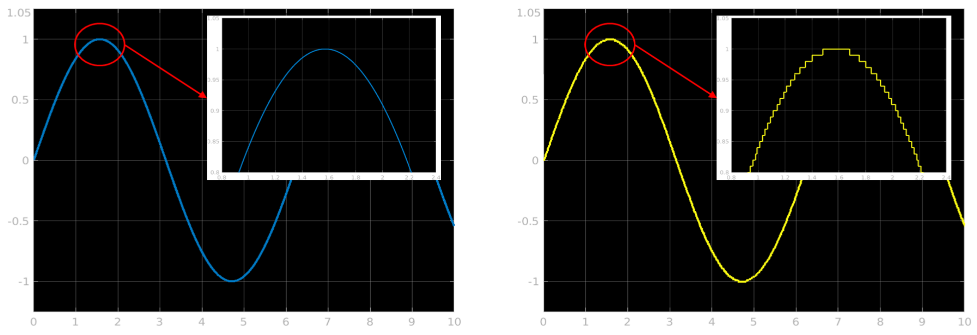


Figure 6. Results presented in the osciloscope: (a) input signal, (b) signal received from the microcontroller

3.2 HIL Test Platform Implementation

The HIL platform with the autopilot for controlling of a fixed-wing aircraft modeled in Matlab Simulink®, was developed and made available in a repository on GitHub (Moura, 2021b) in the "HIL" folder name. In this folder, there are two subfolders: “Arduino”, where a program was developed to be embedded in a microcontroller; and “Matlab”, where a program in Matlab Simulink® was developed to be used for the simulation of fixed-wing aircraft.

In the first folder there are three files: “HIL_Serial.ino”, which describes the implementation of a program in C++ language of the Control and Guidance block; the file “HIL_Waypoints.h”, which defines the waypoints to be reached by the autopilot; and the file “HIL_Variables.h”, which defines the variables used by the program.

The mapping of the desired waypoints, for each trajectory, must be provided in a matrix of points defined in the local navigation coordinate system (NED). The Table 2 shows the tested example.

These files must be compiled on the Arduino platform and embedded in the chosen microcontroller.

Table 2. Waypoints mapping (Santos, 2018).

	WP_1	WP_2	WP_3	WP_4	WP_5	WP_6	WP_7
$N(m)$	0	1000	-809.0170	309.0170	309.0170	-809.0170	1000
$E(m)$	0	0	587.7853	-951.0565	951.0565	-587.7853	0
$D(m)$	400	400	420	380	420	380	400
$V(m/s)$	20	20	20	20	20	20	20

In the second folder, there are the Matlab Simulink® files to be used in the simulation.

The “start.m” file initializes the parameters and file necessary to the execution of the simulate aircraft, in this case the Piper J-3 Cub ¼. It defines the input vectors and aircraft steady states determined according to Eq. 15; loads the file “Piper J3 1_4 VT0_20.mat” which contains the parameters of the linearized model of the fixed-wing aircraft used to be simulated; and opens the file “HIL_Serial.slx”.

$$\vec{u}=\begin{pmatrix}\Pi & \delta_e & \delta_a & \delta_r\end{pmatrix}^T \\ \vec{y}=\begin{pmatrix}V_t & \alpha & \beta & p & q & r & \phi & \theta & \psi & x_n & y_n & H\end{pmatrix}^T \tag{15}$$

This last file, “HIL_Serial.slx”, contains the Matlab Simulink® blocks, shown in Figure 7, and it is responsible for performing the simulation. It is composed of the “HIL (Serial) – Navigation / Guidance / Control” block, the “Aircraft” block and the state and input vectors (defined in Eq. 15) calculated on each interaction.

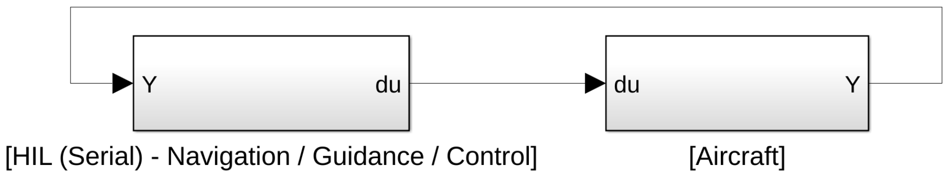


Figure 7. Matlab Simulink® blocks from the “HIL_Serial.slx” file.

The “HIL (Serial) - Navigation / Guidance / Control” block (Figure 8) is responsible for the communication between the microcontroller and the simulated aircraft. In the first step, the content of the input vector ($\vec{y}$) is sent to the microcontroller through the serial communication protocol. Once received, the microcontroller performs the calculations and returns, for the simulation, the content of the state vector ($\vec{du}$) – which is sent to the "Aircraft" block; a stop signal – triggered when the simulated aircraft reaches the final waypoint; and the altitude (H_e) and yaw angle (ψ_e) reference signals.

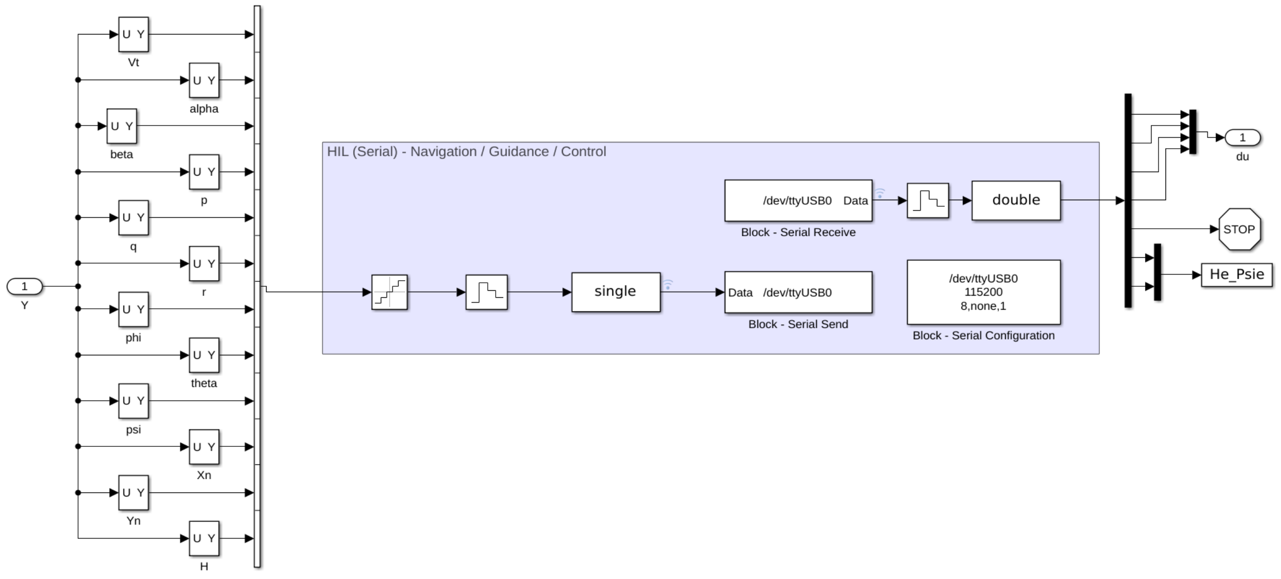
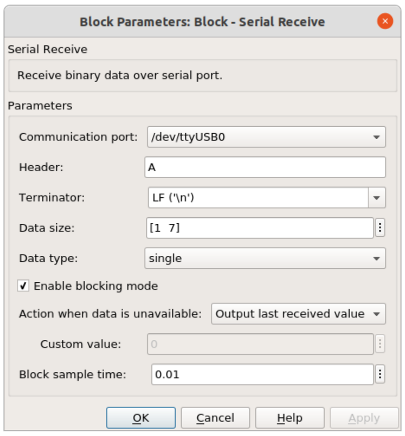


Figure 8. Subsystem “HIL (Serial) – Navigation / Guidance / Control” detail.

The configurations of the “Serial Configuration” and “Serial Send” blocks follow the same configuration previously reported in Figure 5. For the “Serial Receive” block, the “Data Size” parameter must be set to a seven-position vector, as shown in Figure 9.

Finally, the “Aircraft” block (see Figure 7) is implemented by the subsystem shown in Figure 10. This block receives the state vector ($\vec{du}$) coming from the “HIL (Serial) – Navigation / Guidance / Control” block and adds it to the steady state vector ($\vec{U}_e$), defined in “start.m”, generating the state vector to be used in the function “sfunction_piper.m”. This function performs the calculations of the rigid body dynamics of the fixed-wing aircraft generating the new values for the input vector, repeating all the cycle.

Once the simulation is finished, the graphics for the analysis of the output signals (trajectory, speed, altitude and yaw angle) can be generated through the file "plot_graphics.m".


Block Parameters: Block - Serial Receive

Serial Receive

Receive binary data over serial port.

Parameters

Communication port: /dev/ttyUSB0

Header: A

Terminator: LF ('\n')

Data size: [1 7]

Data type: single

[✓] Enable blocking mode

Action when data is unavailable: Output last received value

Custom value: 0

Block sample time: 0.01

Figure 9. Configuration of the “Serial Receive” block.

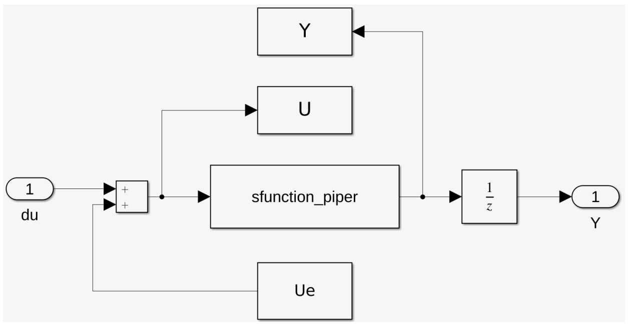


Figure 10. Detail of the Aircraft subsystem.

4 RESULTS AND DISCUSSIONS

Mission results consist of traversing a trajectory of predefined waypoints. After reaching the last desired point, the mission is considered complete and the simulation ends.

For the simulation, the star-shaped trajectory defined in Table 2 was selected, where the waypoints are defined in the NED coordinate system. This trajectory is the same as defined in Santos (2018, p. 105) and was chosen so that the graphic results can be compared. The microcontroller used was ESP32 of Expressif Systems and repository files on GitHub in Moura (2021b).

The simulation was considered to take place under ideal conditions and, during the simulations, the reference speed of the fixed-wing aircraft was kept in the steady region ($V_{T_r} = 20 m/s$). The duration of the mission was 562s.

Figure 11 shows the result of the trajectory obtained. Here it is seen that the aircraft passed through the preset waypoints starting at WP1 and ending at WP7, equaling the graphic reported in Santos (2018).

Figure 12 shows the responses for the airspeed. Comparing the results, we see that the HIL oscillated less in speed than what was reported by Santos (2018). This is due to the fact that the signals sent from Matlab Simulink® to the microcontroller are reduced from 8 bytes to 4 bytes (see "single" block in Figure 8), thus reducing the range of variation of the calculations performed by the microcontroller. Another point to note is that, due to quantization, the values sent to the microcontroller tend to be truncated into intervals band (see Figure 6b).

Figures 13 and 14 show the responses, respectively, for the altitude and yaw angle of the fixed-wing aircraft during the simulation. The fluctuation of results is due to the same reason described in the previous paragraph.

Observing the graphics we see that the waypoints were reached and the results are equal to those found by Santos (2018, p. 105) thus showing the success of the mission.

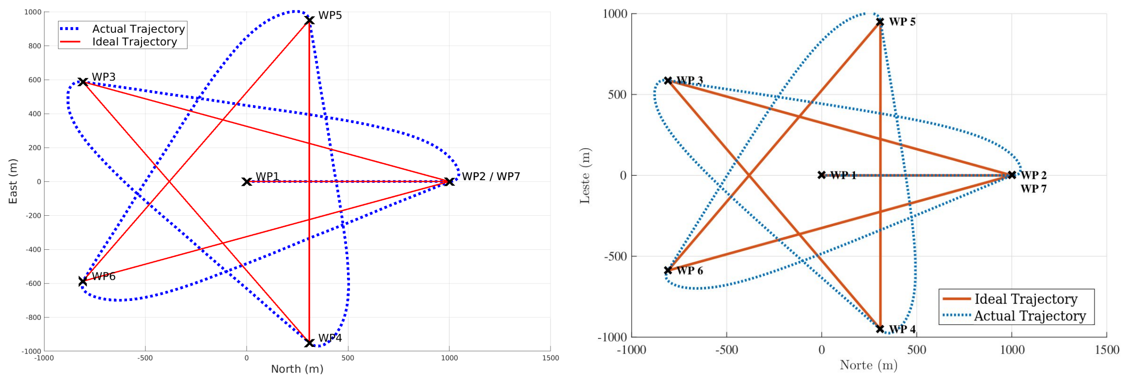


Figure 11. Trajectory performed by aircraft in simulation. (a) HIL in ESP32, (b) HIL adapted from Santos (2018, p. 106)

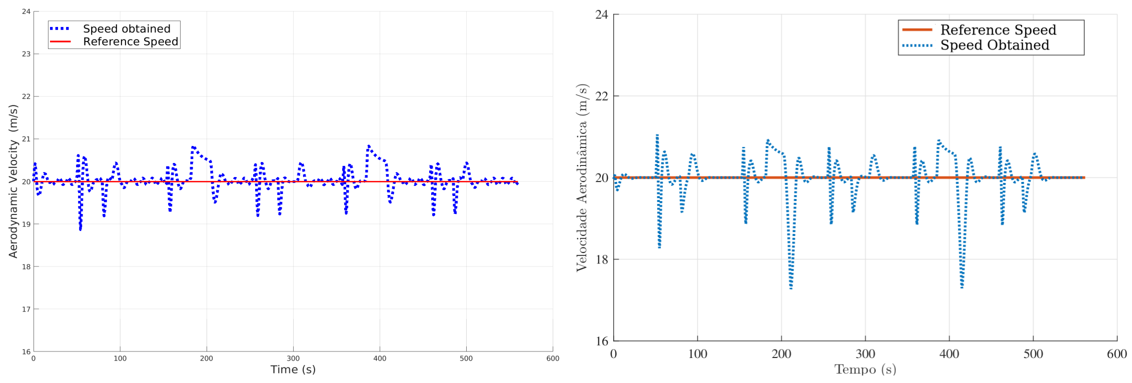


Figure 12. Speed performed by aircraft in simulation. (a) HIL in ESP32, (b) HIL adapted from Santos (2018, p. 106).

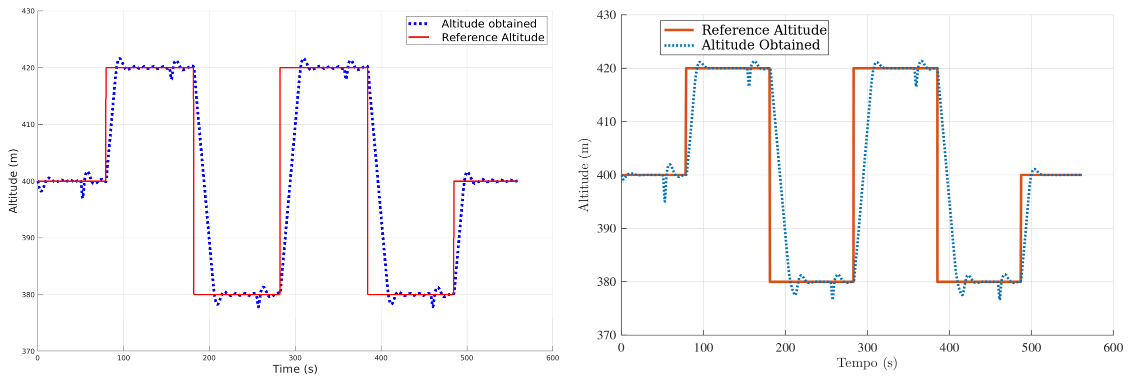


Figure 13. Altitude performed by aircraft in simulation. (a) HIL in ESP32, (b) HIL adapted from Santos (2018, p. 107)

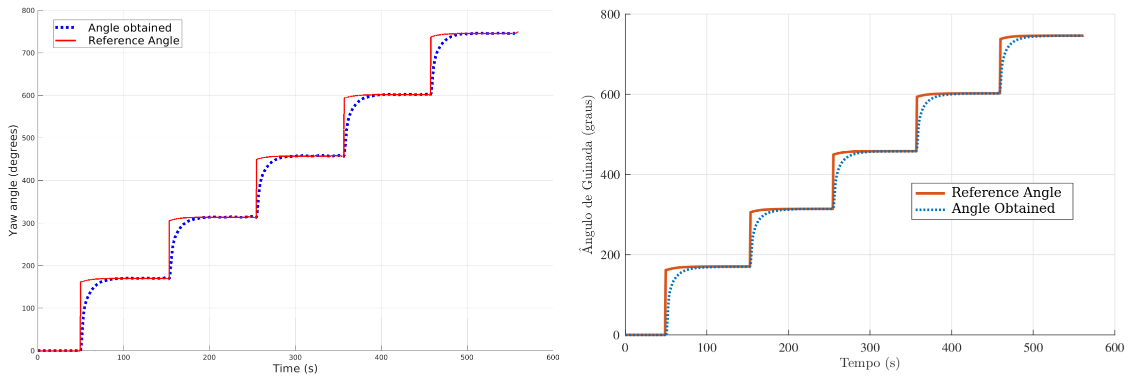


Figure 14. Angle performed by aircraft in simulation. (a) HIL in ESP32, (b) HIL adapted from Santos (2018, p. 107)

5 CONCLUSION

This work aimed to develop a tutorial that describes the steps for the implementation and use of an autopilot HIL platform embedded in a dedicated microcontroller to control a fixed-wing aircraft simulated in a Matlab Simulink® programming environment. It is described since the communication between the microcontroller and simulated aircraft until the necessary steps to board and execute the full control loop of an autopilot to carry out a mission defined by way-points. The development was made for microcontrollers that use an open platform that is low cost, easy to handle and easy to program. The discretization equations implemented in the microcontroller were detailed, as well as the tests and communication between microcontroller and simulator. The results are presented in graphs of successful flight mission simulations being established through pre-defined waypoints reached by the aircraft. Finally, all source codes used for tests and simulations are available on the GitHub platform (Moura 2021a, 2021b) allowing the end user to reproduce the experiment and adapt it according to their needs.

6 REFERENCES

Arduino., 2021. What is Arduino?, https://www.arduino.cc/en/Guide/Introduction. Accessed 2021 January 26.

Atmega328p., 2020. Data sheet. Microchip. DS40002061B. Rev 8271A. Release: 2020 Sep 12., https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf. Accessed 2021 January 12.

Cai, G.; Chen, B.; Lee, T.; Dong, M., 2009. Design and implementation of a hardware-in-the-loop simulation system for small-scale uav helicopters. Mechatronics, https://www.sciencedirect.com/science/article/abs/pii/S0957415809001123. Accessed 2021 February 25.

Cook., M. V., 2013. A Linear Systems Approach to Aircraft Stability and Control. Third Edition. Elsevier.

Esp32., 2021. Data sheet. Espressif Systems. Version 3.6. Rev 2021 Mar 19, https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Accessed 2021 January 12.

Esp8266., 2020. Data sheet. Expressif Systems. Version 6.6. Rev 2020 Sep 30, https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. Accessed 2021 January 12.

Franklin, G. F.; Powell, J. D.; Emami-Naeini, A., 2013. Control Systems for Engineering (in Portuguese). – 6. ed. – Porto Alegre: Bookman.

Jaw, L. C.; Mattingly, J. D., 2009. Aircraft engine controls: design, system analysis, and health monitoring. American Institute of Aeronautics and Astronautics, Inc., Reston, Virginia.

Lizarraga, M.; Elkaim, G.; Horn, G.; Curry, R.; Dobrokhodov, V.; Kaminer, I., 2009. Low Cost Rapidly Reconfigurable UAV Autopilot for Research and Development of Guidance, Navigation and Control Algorithms. ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.

Mariga, L., 2019. How to communicate Simulink and Arduino connection. GitHub Repository, https://github.com/leomariga/Simulink-Arduino-Serial. Accessed 2021 June 14.

Moura, W. M., 2021a. Communications Tests – Serial Communication. GitHub Repository, https://github.com/MouraWM/HIL-platform-fixed-wing-autopilot/tree/main/Communication_Tests/Serial_Communication. Accessed 2021 June 14.

Moura, W. M., 2021b. HIL – HIL Platform and Autopilot. GitHub Repository, https://github.com/MouraWM/HIL-platform-fixed-wing-autopilot/tree/main/HIL. Accessed 2021 June 14.

Santos, M. H., 2018. Design and bench test of an autopilot for unmanned aerial vehicles (in Portuguese). 150 p. Master’s Thesis, Graduate Program in Electronics and Computer Engineering, Aeronautics Technological Institute, São José dos Campos, Brazil.

Santos, S. R. B.; Givigi Junior, S. N.; Nascimento Júnior, C. L.; Bittar, A.; Oliveira, N. M. F., 2011. Modeling of a hardware-in-the-loop simulator for uav autopilot controllers. 21st Brazilian Congress of Mechanical Engineering. October 24-28, 2011, Natal, RN, Brazil.

Schulz, Stephan.; Hasberg, Hagen.; Buscher, René., 2020. Hardware-in-the-loop testbed for autopilot development using flight simulation and parallel kinematics. Faculty of Engineering and Computer Science, Hamburg University, https://zenodo.org/record/3757919/files/schulzhasbergbuescher_hiltestbed_2020.pdf. Acessed 2021 June 12.

Stevens, B. L.; Lewis., F. L.; Johnson., E. N., 2016. Aircraft control and simulation. Dynamics, controls design, and autonomous systems. Third Edition. Published by John Wiley & Sons, Inc., Hoboken, New Jersey.

Stm32f103x8., 2015. Data sheet. STMicroelectronics. DocID13587 Rev 17. 2015 Aug 21, https://www.st.com/resource/en/datasheet/stm32f103c8.pdf. Accessed 2021 January 26.

7 RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.