HARDWARE-IN-THE-LOOP SIMULATION ENVIRONMENT FOR TESTING OF TILT-ROTOR UAV'S CONTROL STRATEGIES

Arthur V. Lara* Iuro B. P. Nascimento* Janier Arias-Garcia† Leandro Buss Becker‡ Guilherme V. Raffo*,†

*Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Minas Gerais
CEP 31270-901, Belo Horizonte, Minas Gerais, Brasil

†Departamento de Engenharia Eletrônica - Universidade Federl de Minas Gerais
CEP 31270-901, Belo Horizonte, MG, Brasil

‡Departamento de Automação e Sistemas, Universidade Federal de Santa Catarina
CEP 88040-900, Florianópolis, SC, Brasil

Email: avlara@hotmail.com, iuro@ufmg.br, janier-arias@ufmg.com, lbecker@das.ufsc.br, raffo@ufmg.br

Abstract— This paper presents a hardware-in-the-loop simulation environment for testing of control strategies for Unmanned Aerial Vehicles in Tilt-rotor configuration. This proposal consists of a distributed system composed by a general purpose computer, that runs the Gazebo simulator, and a stm32f4 discovery development board, that executes the control strategy. For instrumentation and reference data transfer it was developed a serial communication protocol that includes an error-detecting method for serial transmissions, which uses a checksum for accidental change detection in data transfer chains. Besides, it was created a dynamic library in order to provide an external channel with Gazebo Simulator to allow interaction between simulation and embedded system. Also it was designed a trajectory creator program to select the reference points to be reached by the control strategy, being either automatically or manually via joystick. Finally, the system is validated through simulation of a Tilt-rotor UAV being controlled by a DLQR strategy.

Keywords— Tilt-rotor UAV, Hardware-in-the-loop, Embedded systems, Robotic simulator, Control systems

Resumo— Este artigo apresenta um ambiente de simulação via hardware-in-the-loop para testes de estratégias de controle para Veículos Aéreos Não Tripulados na configuração Tilt-rotor. O sistema proposto consiste em um sistema distribuído composto por um computador de uso geral, que executa o simulador Gazebo, e uma placa de desenvolvimento stm32f4 discovery, que implementa a estratégia de controle. Para transferência de dados de referência e instrumentação entre o simulador e o controlador, desenvolveu-se um protocolo de comunicação serial que inclui um método de detecção de erro para comunicações seriais usando um checksum com o objetivo de detectar mudança acidental em cadeias de dados em transferência. Além disso, foi criada uma biblioteca dinâmica para prover um canal externo com o simulador Gazebo e permitir a interação entre a simulação e o sistema embarcado. Ademais, desenvolveu-se um programa de seleção de trajetória para selecionar os pontos de referência a serem alcançados pela estratégia de controle, podendo ser de forma automática ou manual via *joystick*. Por fim, o sistema é validado através de simulação de um VANT Tilt-rotor sendo controlado por uma estratégia de controle DLQR.

Palavras-chave— VANT Tilt-rotor, Hardware-in-the-loop, Sistemas embarcados, Simulador Robótico, Sistemas de controle

1 Introduction

Nowadays, due to Unmanned Aerial Vehicle's range of applications as for example: spraying of crops, livestock management, road monitoring, inspection of power lines and delivery of supplies in risky zones; there are a significant effort in enginnering and science in order to design and improve their technology. The process of development and implementation usually is too expensive and slow, mainly because of difficulties in performing hardware and firmware tests. Thus, it is necessary to adopt new methologies in order to speed up the implementation process and to decrease the financial costs.

Given the circumstances, the present work features the development of a simulation environment based on Hardware-in-the-loop Simulation (HILS) of a Tilt-rotor UAV for testing of control strategies. Tilt-rotor is a kind of hibrid aircraft characterized by two propellers and mechanisms responsible for tilting them, allowing two modes of operation: helicopter and airplane. Due to this configuration, it is capable of performing both vertical takeoff and landing, and forward flights with higher speeds when compared to a rotary wing UAV. The model of Tilt-rotor UAV, illustrated in Figure 1, is used in this work to demonstrate the HILS environment. Such aircraft was completely designed and assembled in our research group.

In the context of control systems' design, simulation is a process of conducting experiments with a computational/mathematical model of a physical system in order to test the effectiveness of some control strategy. However, rather than testing the control algorithm on a purely simulation environment (like Matlab/Simulink), HILS includes a part of the real hardware in the

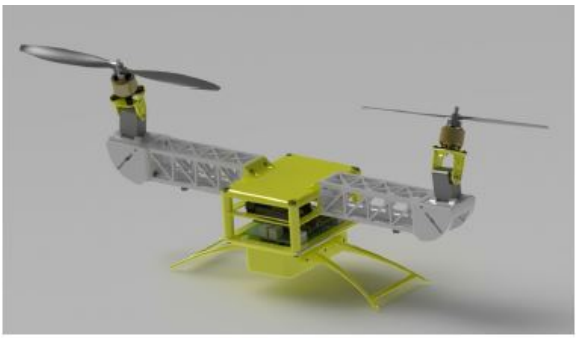

Figure 1: Tilt-rotor UAV 2.0.

simulation loop during the system development. This embedded system has real-time requirements to maintain the flow of input and output signals between the simulation environment (running the model) and the embedded system (running the control strategy).

Recently, there are some approaches in the literature that look into the small UAV control design using HILS. Gans et al. (2005) developed a HILS environment for airplane UAV control tests. It uses virtual reality software to produce real-world scenarios and a wind tunnel for aerodynamic simulation of the aerial vehicle. Trilaksono et al. (2011) designed a HILS for visual target tracking of an octorotor UAV with onboard computer vision. In Cheon et al. (2016), a HILS platform was designed for verifying the image-based object tracking method used in a UAV, composed by a image processing, scene generation module and a flight control modules.

The present work uses the Gazebo simulator[1], which is a 3D simulation software with free license under the responsibility of Open Source Robotics Foundation (OSRF). In order to allow fast data tranfer between the embedded system and the general purpose computer, it was proposed a new serial communication protocol. Besides, a dynamic library was designed to create an external channel with the Gazebo Simulator, allowing an interaction between the embedded system and simulation. Also, it was developed a trajectory creator program to select the reference points to be achieved by the control strategy.

The design of HILS is an improvement of the ProVANT Simulator (Lara et al., 2017). ProVANT Simulator is a simulation environment that uses models based on CAD (Computer Aided Design) 3D models and the Gazebo Simulator, with the purpose of validation and implemetation of control strategies, which is the previous stage of flight testing. The inclusion of HILS in the ProVANT Simulator allows not only test the control strategy, but also test its implementation on the actual hardware that will, in fact, be used in the prototypes.

[1] http://gazebosim.org/

This paper is organized as follows: Section II presents the development of the HILS environment; Section III decribes the experimentation and obtained results; and, finally, Section IV concludes this work and discusses future works.

2 Hardware-in-the-loop Simulation Environment

The proposed HILS presents some requiments. As a functional requirement, we must use a protocol of communication that has a worthless time lag. And, as a non-functional (real-time) requirement, the period of time between requests for sensor data from the embedded system to the simulator should be 12 ms. This sampling time value was defined according to the limitation of the Tilt-rotor UAV's instrumentation.

The general purpose computer, which runs the simulator and the trajectory creator program, has a 64 bits processor Intel Core i7-4790 CPU @3.60GHZx8; a Graphics Processing Unit: Intel Haswell Desktop; and 7,3 GB of RAM and 659,4 GB of hard drive.

The embedded system, which runs the control law, is a stm32f4 discovery development board composed by an ARM Cortex-M4 with FPU core, 1 MB of Flash memory and 192 KB of RAM, and possesses the following channels: SPI, UART, I2C and CAN.

2.1 Communication

To perform the communication between the general purpose computer and the embedded system, it was chosen the UART (Universal Asynchronous Receiver-Transmitter) protocol in the embedded system side and the USB (Universal Serial Bus) protocol in the general purpose computer side. Thus, to make the translation of these serial protocols, it is used two converters FT232RL: one is responsable for sensors and actuators data transfer and the other for reference data transfer.

The serial communication uses baud rate of 921600 bps in order to provide fast communication with a worthless time lag. For this work, the maximum payload to be transmitted is 132 bytes with time lag of 1.2 ms, which can be acceptable, being 10% of the sampling time.

In order to have a reliable communication between the embedded system and the computer, it is necessary to guarantee a safe exchange of packages between systems. The main related problems are the late delivery of packages, data corruption and the reception of duplicated packages.

These problems are mittigated using a two layer communication protocol. The low level layer

tackles data corruption and the entanglement of packages by sending the data with redundant information as to solve these problems. The high level protocol handles the flow of communication by implementing a client/server protocol.

The low level protocol was designed as an adaptation/simplification of the Point-to-Point (PPP) protocol (Simpson, 1994), widely used by Internet providers for DSL (Digital Subscriber Line) connections. The data structure of the protocol is showed in Figure 2. In the data structure, it can be seen the start and the end fields, aiming to avoid package entanglement. Besides, in order to avoid package corruption, a 2 byte checksum is added before the end byte, and it uses the Fletcher's checksum algorithm as error-detecting code (Fletcher, 1982). In addition, the data can have any length, and starts right after the start byte.

Figure 2: Low level protocol's frame.

The start and end bytes used are 0x7E. Besides, in order to avoid confusion with a possible start byte in the data, an escape byte is introduced with value 0x7D. Whenever a start, end or escape flag byte occurs in the data or checksum, the escape byte is inserted and, then, the *XOR* operation of the flag with 0x20 is inserted after it. To recover the original frame, it is removed the escape and substituted the next value with the *XOR* operation of the byte with 0x20.

The high level protocol consists in a client/server communication, where the embedded system is the client of the application and asks to the general purpose computer for services identified through its ID and some information, if necessary, as showed in Figure 3. The following services are provided: i) Simulation start; ii) Reading of sensors data; iii) Transmission of data to actuators.

Figure 3: High level protocol's package.

The data flow in the communication works as showed in Figure 4. First of all, the client commands to start the application, sending a mensage with ID 1 without waiting for any response. From this moment, it starts a periodic cycle of sensor data request and sending actuator commands. In sensor data request, the client sends a message with ID 2 and waits for a server's answer, which consists of one float array of 16 elements. However, when it sends actuator commands, it uses a message with ID 3 with more 4 float numbers corresponding to the control signals resulting from the control law. As a client/server protocol, this periodic cycle is set on the client-side, and the server just waits for requests.

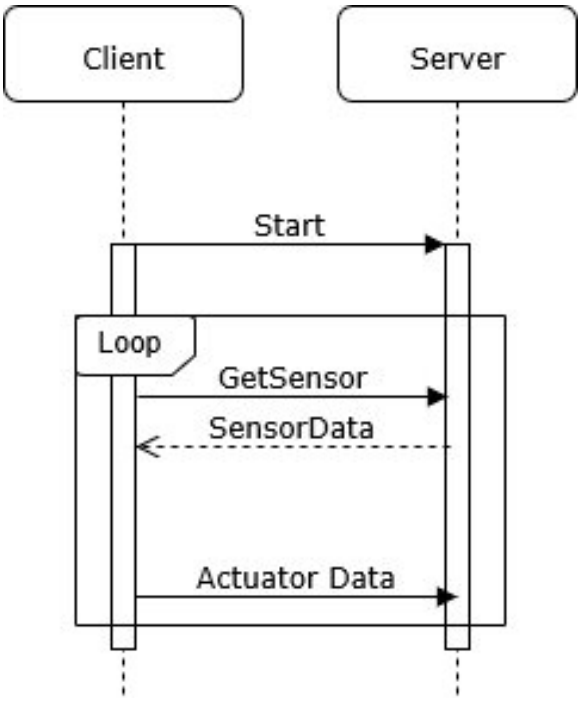

Figure 4: Communication data flow between server and client.

2.2 General purpose computer configuration

The general purpose computer uses the linux distribution Ubuntu 16.04 LTS (Long Term Support) as operating system, which does not have real-time support. Besides, it is also used the Gazebo Simulator to simulate the dynamic behavior of the Tilt-rotor-UAV.

The simulation step of Gazebo simulator was adjusted for 4 ms. The simulation step is sampling time of the simulator. They are the time instants where the simulator obtains the data of control signals and computes the new states of the system. By increasing the simulation step, it is lost the precision of calculation, but the amount of processing used by the simulator is decreased.

In addition, the Gazebo Simulator is configured with *real-time_factor* setting equals to 1, making the simulator tries to keep the simulation in real time according to the system clock.

In order to interact with the simulation, either by acquiring data and applying control signals, or by changing simulation configurations, it was created a dynamic library called Plugin to be responsible for getting requests from serial communication. It was used Boost[2] ASIO

[2]http://www.boost.org

API (Application Program Interface), that is a cross-platform C++ library for network and low-level I/O programming, and Boost Thread API for creating and managing threads.

Plugin works as showed in Figure 5. It is composed for one thread, that waits for external requests, and one callback that is called in every simulation step. For each type of request, it has a specific reaction and the verification process happens in the following sequence: 1) start simulation, 2) send actuator data to the simulator and 3) get sensor data. Besides, a callback is needed in order to ensure that the last value of control input is applyed on the model, since all input variables in simulation are reset in every new simulation step in Gazebo simulation.

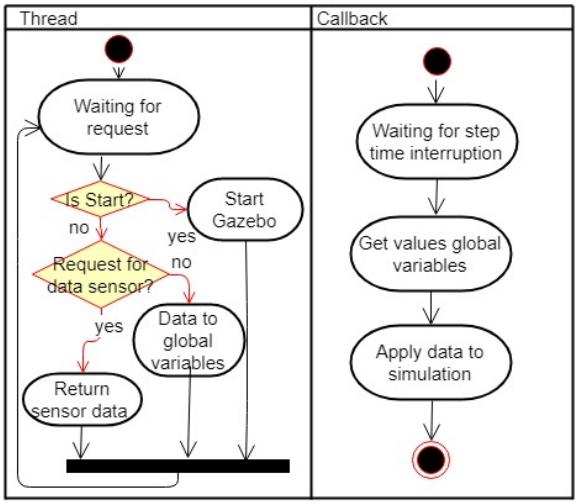

Figure 5: Logic of server software.

In addition to the Plugin, the general purpose computer has also two different programs which send reference points to be tracked by the control system: the base station program and a joystick reader. Both of them uses just the low level protocol to communicate with embedded system, but each one has a different behavior, as can be observed in Figure 6, when one of them is turned on, the other must be turned off.

The joystick reader receives data from joystick driver and converts the data to magnitude and the appropriate scale before sending references to the embedded system. However, the base station program gets the clock update of Gazebo simulation and computes a new reference to the UAV and, then, sends this data to the embedded system.

2.3 Embedded system configuration

The stm32f4 discovery development board uses a software architecture structured in a hierarchical verticalization, where the elements of a lower level provide services for elements of higher level, as can be seen in Figure 7. In general

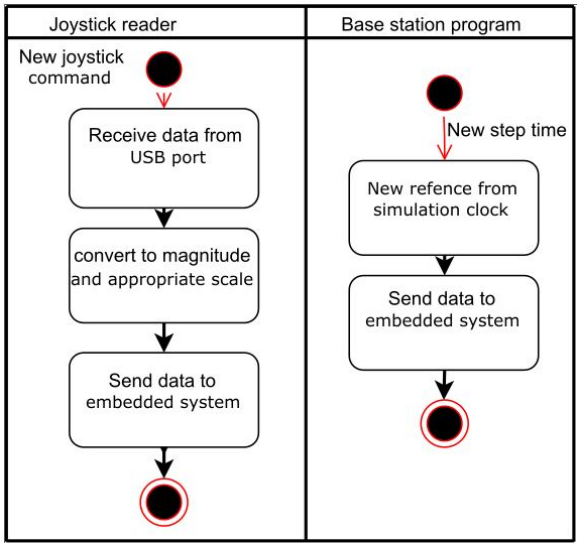

Figure 6: Logic of joystick reader and base station.

there are three layers: Core level, Middleware level and Application level. At the first level, it implements the basic run-time system for a Cortex-M device and gives the access of processor core and peripherals for the user. Middleware level is the layer responsible for encapsulating communication with peripherals and commonly used code for dealing with the communication problem between processes and mechanisms of concurrence. Finally, Application implements the modules of stabilization and navigation codes.

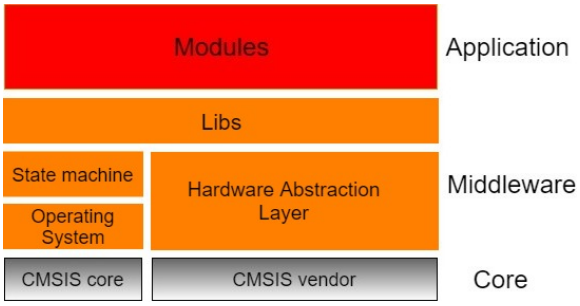

Figure 7: Client software architecture.

Besides, it has been used FreeRTOS[3] as an operating system. It is a real-time kernel for embedded systems that allows the creation of tasks and provides mechanisms in order to achieve their real-time objectives. There is also an inter-task communication mechanism using thread queues, which is thread safe.

Thus, through the described infrastructure, it was implemented a control system which has four threads: one thread for getting sensor data, one for processing the control law, one for sending actuator data, and other to get reference data. Their operation are showed in Figure 8. In addition, theses threads are called by the time

[3]https://www.freertos.org/

interruption every sample time, which period is 12 ms.

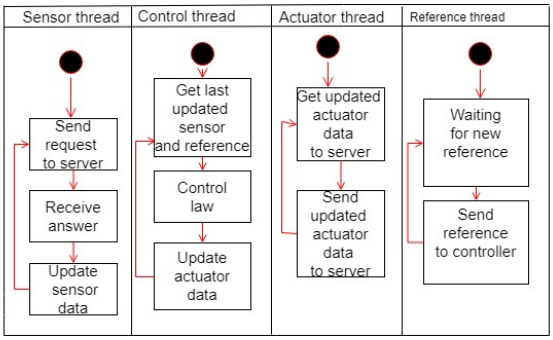

Figure 8: Client software.

3 Experimental Results

In order to evaluate the proposed HILS, experimental results were carried out, where the embedded system was configured with a Discrete Linear Quadratic Regulator (DLQR) control technique proposed by Rego e Raffo (2016), which was designed based an the physical model presented in Donadel et al. (2014). In this experiment the Tilt-rotor UAV must track some trajectory generated by the base station program.

The kinematic description of the system is perfomed according to Figure 9, where $\boldsymbol{\xi} \triangleq [x\, y\, z]^T$ corresponds to the position of the main body with respect to the inertial frame, ϕ, θ and ψ describe the orientation of the main body with respect to the inertial frame through the Z-Y-X convention on local axes (Jazar, 2010), α_{R} and α_{L} describe the inclination of the propellers with respect to the main body of the aircraft. The vectors $\boldsymbol{d}^{\mathcal{B}}_{\mathcal{C}_i}$, with $i \in \{1,2,3\}$, and the angle of inclination β corresponds to design parameters of the aircraft. Table 1 presents the physical parameters of the UAV model used to tune the DLQR controller.

The control strategy is based on the linearization and discretization of the system state equations, obtained through the Euler-Lagrange formulation, around the reference trajectory. The generalized coordinates were $q = (x, y, z, \phi, \theta, \psi, \alpha_R, \alpha_L)$, and integral actions are added on the regulated variables (x, y, z and ψ). The parameters used for control design are

$$\boldsymbol{R} = \mathrm{diag}(\sqrt{0.2}, \sqrt{0.2}, \sqrt{0.05}, \sqrt{0.05})$$
$$\boldsymbol{Q} = \mathrm{diag}(5,5,5, \tfrac{4}{\pi^2}, \tfrac{4}{\pi^2}, \tfrac{15}{\pi^2}, \tfrac{40}{\pi^2}, \tfrac{40}{\pi^2}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{9}{\pi^2}, \tfrac{9}{\pi^2}, \tfrac{16}{\pi^2}, \tfrac{5}{9\pi^2}, \tfrac{5}{9\pi^2}, 50, 50, 50, 1)$$

with $f^R_{eq} = 10.2751$ N, $f^L_{eq} = 10.2799$ N, $\tau^R_{eq} = 0$ N.m, $\tau^L_{eq} = 0$ N.m.

Table 1: Physical parameters of system.

Parameter	Value
m_1	1.92 Kg
m_2, m_3	0.08 Kg
$\boldsymbol{d}^{\mathcal{B}}_{\mathcal{C}_1}$	$[-0.0025\ 0.00013\ -0.018]^T$ m
$\boldsymbol{d}^{\mathcal{B}}_{\mathcal{C}_2}$	$[0.00002\ -0.27761\ 0.05621]^T$ m
$\boldsymbol{d}^{\mathcal{B}}_{\mathcal{C}_3}$	$[0.00005\ 0.27761\ 0.05621]^T$ m
$\boldsymbol{I}_1$	$\begin{bmatrix} 6900.0 & 4.7 & 36.0 \\ * & 790.0 & -1.07 \\ * & * & 6600.0 \end{bmatrix} \cdot 10^{-5}$ Kg·m^2
$\boldsymbol{I}_2$	$\begin{bmatrix} 3.88 & 0 & 0 \\ * & 10.0 & 0 \\ * & * & 8.3 \end{bmatrix} \cdot 10^{-5}$ Kg·m^2
$\boldsymbol{I}_3$	$\begin{bmatrix} 3.83 & 0 & 0 \\ * & 10.0 & 0 \\ * & * & 8.3 \end{bmatrix} \cdot 10^{-5}$ Kg·m^2
$\hat{\boldsymbol{g}}$	$[0\ 0\ -9.81]^T$ m/s^2
k_τ	$1.7 \cdot 10^{-7}$ N·m·s^2
b	$9.5 \cdot 10^{-6}$ N·s^2
$(\lambda_{\mathrm{R}}, \lambda_{\mathrm{L}})$	$(1, -1)$
β	5^{o}

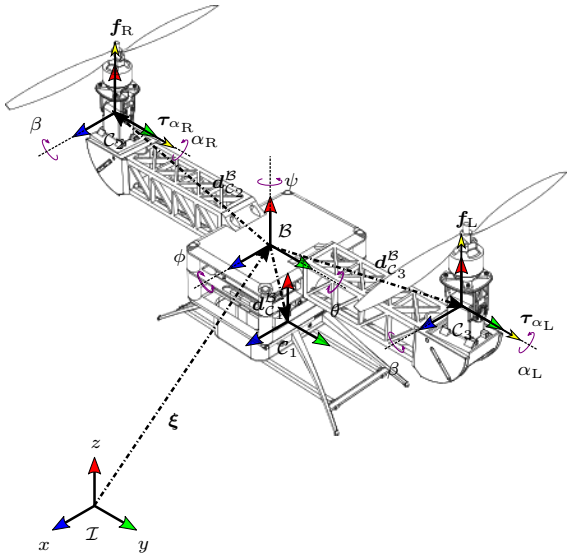

Figure 9: Tilt-rotor UAV coordinates

During the experiment, it has been asked for the UAV to track a spiral trajectory described by equation (1), where $n = 3$ is the number of turns, $T_f = 60s$ is the total time, $d = 5m$ is the trajectory radius, $h = 10m$ is the height, and $(x_0, y_0, z_0) = (2, 0, 0.5)$ is the starting point of the Tilt-rotor UAV in the trajectory.

$$\begin{aligned} w &= \frac{2n\pi}{T_f} \\ y_r(t) &= d \sin wt - y_r(0) + y_0 \\ z_r(t) &= \frac{ht}{T_f} - z_r(0) + z_0 \\ x_r(t) &= d \cos wt - x_r(0) + x_0 \end{aligned} \tag{1}$$

Besides, it has been chosen the ODE engine for Gazebo's configuration. The results of HILS is showed in Figures 10, 11, 12 and 13.

Figure 10 shows that the UAV tracked properly the trajectory, in Figure 11, it can be

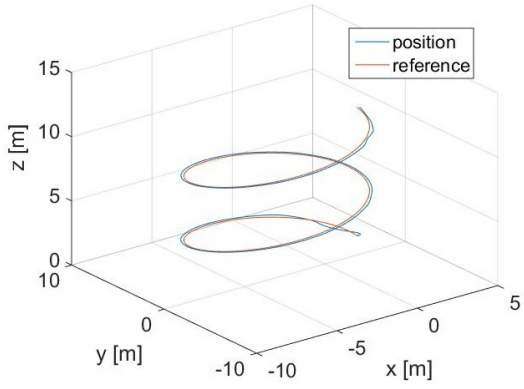

Figure 10: Trajectory of model in HILS.

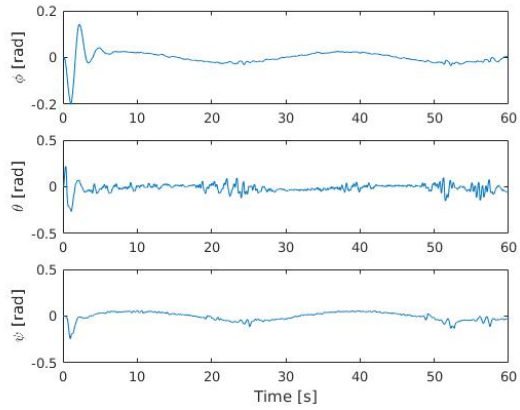

Figure 11: Roll, pitch and yaw.

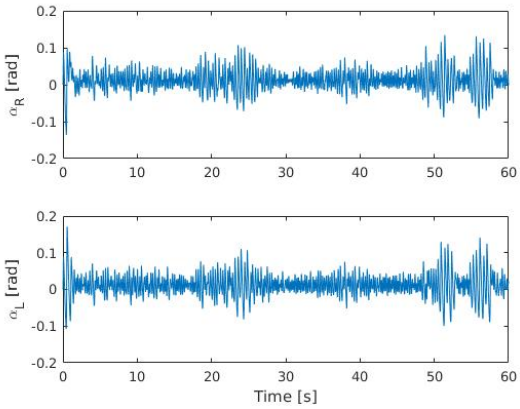

Figure 12: Angles of servo motors.

observed that the yaw angle remained close to the origin. In addition, the remaining states of the system were maintenained stable, and the control signals were properly generated without saturation.

Another information obtained from the HILS was the time periods between requests for sensor data, which were measured in the server side. This information was put in a histogram as can be seen in Figure 14. Most periods between requests for sensor data remain around 12 ms as

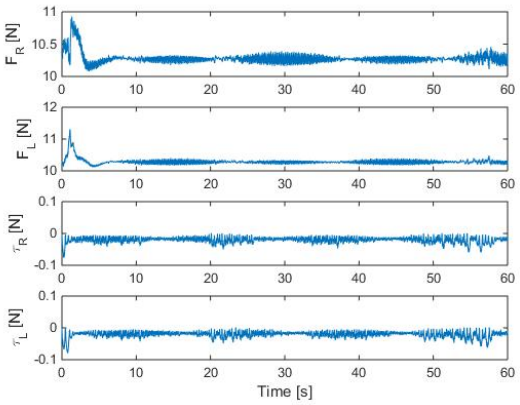

Figure 13: Control inputs.

specified by real-time requirement, but there is a small volume of data concentrated around 13 ms. This volume, probably, is the result of using the Ubuntu 16.04 on the general purpose computer, which is not a real-time operating system and has others processes running concurrently with HILS. Therefore, it does not guarantee that the data obtained from the serial port is handled immediately and, thus, causes such delays as seen in the histogram.

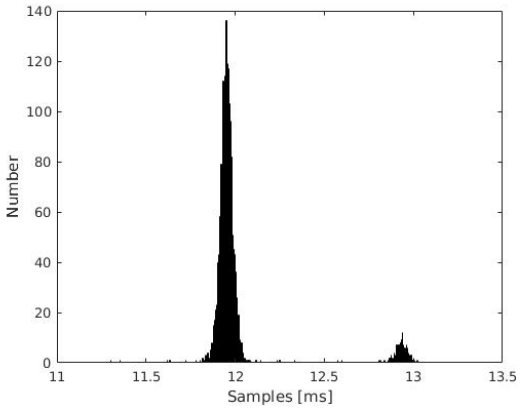

Figure 14: Period of time between sensor requests.

4 Conclusion and Future Work

This work developed a HILS[4] for the tilt-rotor UAV that will serve as testbed for embedded control systems. To enable the communication between the embedded system and the general purpose computer, it was designed two new protocol layers of serial communication that provide flow control of information and worthless time lag. Besides, it was designed a dynamic library to create a channel of communication between the embedded system and the simulation. Also, it was designed an trajectory program to

[4]Video with HILS is available at https://youtu.be/DQ-2-HSwcnU

select the reference points to be tracked by the control strategy. From numeric experiments, it was observed that the application obeys most of the time the real-time requirement. However, even if there are some moments that the real-time requirement is not fulfilled, the control strategy, implemented in a stm32f4 discovery development board, was able to control UAV Tilt-rotor and it tracked properly a spiral trajectory .

As future works, it is expected to use the HILS for testing the implementation of computationaly expensive control algorithms using parallel programming in embedded systems with GPU processors, for instance, model predictive controllers using nonlinear models. Besides, it will be used HILS for testing fault-tolerant distributed architectures of embedded systems, verifying the robustness of the hardware and software against expected failures.

5 Acknowledgment

This work was supported by the projects INCT InSAC and Universal under the grants CNPq 465755/2014-3 and 426392/2016-7, and also by the Brazilian agencies CAPES and FAPEMIG.

References

Cheon, S., Ha, S. e Moon, T. (2016). Hardware-the-loop simulation platform for image-based object tracking method using small UAV, *Digital Avionics Systems Conference, IEEE/AIAA 35th* .

Donadel, R., Raffo, G. V. e Becker, L. B. (2014). Modeling and control of a tiltrotor uav for path tracking, *IFAC Proceedings Volumes* **47**.

Fletcher, J. (1982). An arithmetic checksum for serial transmissions, *IEEE transactions on Communications* **30**(1): 247–252.

Gans, N., Dixon, W., Lind, R. e Kurdila, A. (2005). A hardware-in the-loop simulation platform for vision-based control of unmanned air vehicles, *Mechatronics* .

Lara, A. V., Rego, B. S., Raffo, G. V. e Arias-Garcia, J. (2017). Desenvolvimento de um ambiente de simulação de VANTS Tilt-rotor para testes de estratégias de controle, *XIII Simpósio Brasileiro de Automação Inteligente* .

Rego, B. S. e Raffo, G. V. (2016). Path tracking control based on guaranteed state estimation for a Tilt-rotor UAV, *XXI Congresso Brasileiro de Automática* .

Simpson, W. (1994). Ppp in hdlc-like framing, *STD 51*, RFC Editor.

Trilaksono, B. R., Triadhitama, R., Adiprawita, W. e Wibowo, A. (2011). Hardware-in-the-loop simulation for visual target tracking of octorotor UAV, *Aircraft Engineering and Aerospace Technology: An International Journal* .