Modeling and Simulation-based Development of Autonomy Features for Drones

Shihui Chen[1*], Umut Durak[1,2], Sven Hartmann[2]

[1]TU Clausthal, Institute of Informatics; **shihui.chen@tu-clausthal.de*

[2]German Aerospace Center (DLR), Institute of Flight Systems

SNE 28(2), 2018, 55 - 60, DOI: 10.11128/sne.28.tn.10413
Received: April 15, 2018 (Selected ASIM GMMS/STS 2018 Postconf. Publ.), Accepted: May 15, 2018
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna, ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. In the last decade, more and more aerial robotics researchers show interests in developing autonomy features for drones to solve problems in different areas. But the development of autonomy features is complex and labor intensive. Accordingly, model-based design and simulation-based verification is becoming an industry standard in development of autonomous airborne systems. This we call modelling and simulation-based development. However, commercial model-based design and simulation tools and supporting testing environments require a considerable amount of investment. In oder to provide a more economic and efficient solution, this paper investigates a pipeline for modeling and simulation-based development of autonomy features for drones using open source software and hardware stacks. In this context, a generic drone architecture is being designed based on open source hardware platforms, namely CC3D and Raspberry Pi. In the software stack, LibrePilot, an open source software suite to control multicopters is extended to support the designed architecture. The design of the autonomy features is developed using the model-based design in Scilab/Xcos. Xcos Reuseable and Customizable Code Generator is utilized for automatic code generation. The software stack will also include a generic plant model. The workflow starts from autonomy feature modeling and ends with flight testing through Model-in-the-Loop (MiL) testing, Software-in-the-Loop (SiL) testing, target deployment, Hardware-in-the-Loop (HiL) testing. The approach is demonstrated with a simple case study about an autonomous landing feature.

Introduction

Overview

An aircraft without a human pilot aboard is called an unmanned aerial vehicle (UAV), commonly known as a drone. A UAV is regarded as an essential part of an unmanned aircraft system (UAS). The other parts of the system are ground control system (GCS) and the communication system between the UAV and GCS. The usage of UAS has increased sharply in the recent years. Varies researchers have developed different autonomy features of drones to solve problems in many fields, such as health care emergency response. Especially in some dangerous situations, UAS that incorporate a high level of autonomy has the ability to accomplish the missions more efficiently without risking lives.

The architecture of autonomous systems is very important. It is regarded as a method to structure the algorithms for creating functionalities. Figure 1 depicts the general autonomy architecture [15] for UAS.

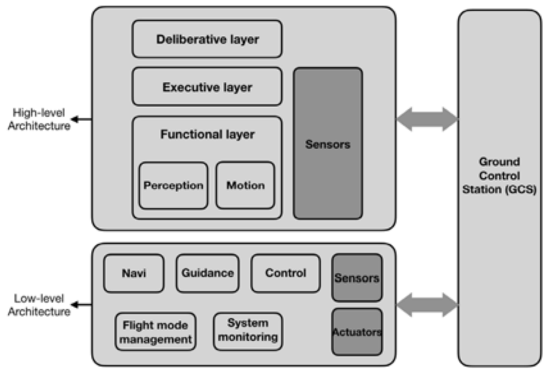

Figure 1. General autonomy architecture for UAS (Adapted from [15]).

The low-level architecture implements the basic functions like navigation and control algorithms that keep the UAV stable in the air and listen to commands from the high level. In this paper, the low-level architecture is realized using OpenPilot CC3D controller [5].

Clough [4] and Merz [13] have elaborated the difference among automatic, autonomous and intelligent systems. An automatic system will exactly do as the programmings say while an autonomous system has the capabilities to make decisions for achieving the missions. An intelligent system can do whatever an autonomous system does and it can produce the goals by its own motivations without any instructions and influence from the outside world. In this paper, system development towards autonomous drones will be discussed.

Modeling and simulation-based development

Model-based design [8] and simulation-based verification are becoming an industry standard in development of autonomous airborne systems. This we call modeling and simulation-based development (Figure 2).

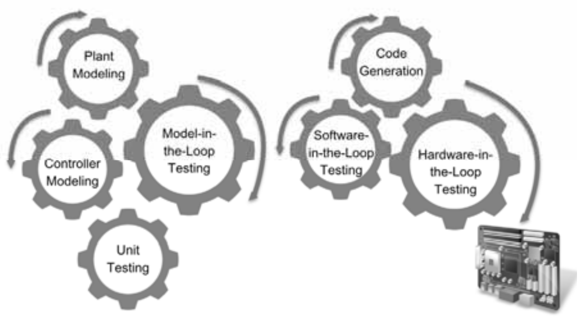
Plant Modeling
Code Generation
Controller Modeling
Model-in-the-Loop Testing
Software-in-the-Loop Testing
Hardware-in-the-Loop Testing
Unit Testing

Figure 2. Modeling and simulation based development.

A plant [9] is often desired with a transfer function which indicates the relation between the input signals and the output signals of a system without feedback, commonly determined by physical properties of the system. Usually a plant model is identified by collecting and processing raw data from the real world. We could define the plant model by using mathematical equations or creating a block diagram model that implements known differential-algebraic equations governing plant dynamics. This is called plant modeling.

The mathematical model conceived from the plant is applied to identify dynamic characteristics of the system. According to those characteristics, a control algorithm that can be executed under the condition which the physical processes are controllable is derived and a suitable controller is chosen. The controller has two levels, the supervisory control that determines the mode transition structure and the lowlevel control that decides the time-based inputs to the plant [12]. For UAS, low-level controller corresponds to low-level architecture and supervisory control is a part of high-level architecture.

To verify the system flexibility, we use simulation-based verification. Each unit and subsystem should be tested and finally achieve a Model in-the-Loop (MiL) testing. Then it leads to Code Generation (CG) [7]. Software in-the-Loop (SiL) testing follows CG to verify the generated code by checking its conformance to the model. Then it comes to the Hardware in-the- Loop (HiL) testing where the generate code is tested using the target hardware.

1 Autonomy Feature Development Pipeline

1.1 Architecture

This paper aims at proposing a process to enable developing complex autonomy features for drones by using open source software and hardware.

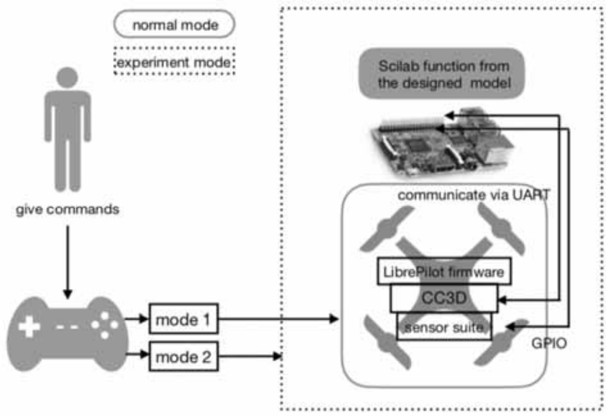

Figure 3. Testbed architecture.

In this paper, regarding their availability, accessibility, cost and flexibility, 250mm class racing drones are used as testbeds. Figure 3 illustrates the testbed architecture. CC3D running the LibrePilot firmware is utilized as the Flight Controller (FC). In the normal mode, users can give command to the Remote Controller (RC) to control the drone. Once the flight mode is switched to experiment mode, Raspberry Pi 3, the target hardware platform of the testbed, will take charge of the controlling while the RC will be disabled. Scilab/Xcos will be utilized as the modelbased design and simulation environment. Xcos Reuseable and Customizable Code Generator [16] is used for generating Scilab scripts from Xcos model. For the use case, the generated Scilab script for the autonomous landing feature is deployed to Raspberry Pi 3. It gets data from ultrasonic distance sensor and CC3D controller, computes the next command and send it to CC3D controller as the new command.

The communication between Raspberry Pi 3 and CC3D is physically established by Universal Asynchronous Receiver/Transmitter (UART) interface where SciPy [11] is used to execute Scilab scripts on target platform through. Related Python libraries are used for interface implementations. UART can control the series device that attached to the computer interface. It provides the computer with the RS-232C Data Terminal Equipment (DTE) interface so that it can "talk" to and exchange data with modems and other serial devices.

1.2 Open source software and hardware stacks

Scilab/Xcos Scilab [3] is a open source software for numerical computation providing a powerful computing environment for engineering and scientific applications. It is a platform to be utilized for model simulation, loading, design, saving and compilation using a graphic editor called Xcos. Xcos has some core features like standards palettes and blocks, model building and modification, model customization and simulation. For some special requirement blocks which are not provided in the Xcos palette browser, users can create their own module by toolbox skeleton to achieve the specific goals.

LibrePilot LibrePilot [17] is an open source research project which focuses on research and development of software and hardware to be utilized for different applications like vehicle control and stabilization, unmanned autonomous vehicles and robotics.

LibrePilot includes hardware and software elements (Figure 4). In the hardware side, UAV is manly controlled by a RC called transmitter. The transmitter has a paired receiver for signal receiving. This receiver also connects with the FC and directly control the actuators. The role of FC is to interpret the control command from RC and runs control algorithm and flight code on the aircraft. If FC is connected to PC where runs LibrePilot Ground Control Station (GCS), users are able to monitor and log flight telemetry data of their vehicle in a real-time environment. This is the software system of LibrePilot which includes GCS software and flight firmware. The flight firmware is implemented in C and C++ using the FreeRTOS [1] embedded real time operating system and typically runs on ARM architecture micro controllers. The communication between GCS and FC is implemented via UAVTalk protocol.

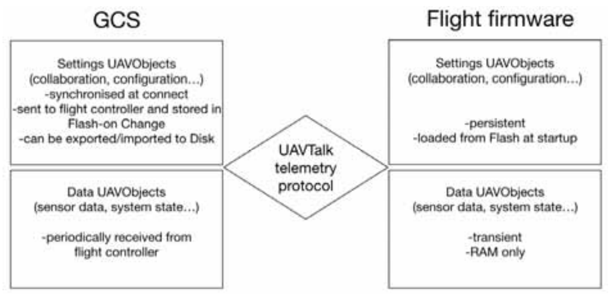
GCS
Flight firmware
Settings UAVObjects (collaboration, configuration...) -synchronised at connect -sent to flight controller and stored in Flash-on Change -can be exported/imported to Disk
Settings UAVObjects (collaboration, configuration...) -persistent -loaded from Flash at startup
UAVTalk telemetry protocol
Data UAVObjects (sensor data, system state...) -periodically received from flight controller
Data UAVObjects (sensor data, system state...) -transient -RAM only

Figure 4. Elements of LibrePilot (adapted from [6]).

UAVTalk is a highly efficient, extremely flexible and completely open binary protocol designed specifically for communication with UAVs [6]. It implements the low level communication between the GCS and the autopilot. It acts as a transportation tool for the data structures defined by the UAVObjects, a data container written in XML format for all of the telemetry data. This protocol does not need to know the details of the data structure, its mission is to send byte arrays and routing received byte arrays to specified object for dealing with the data. For example, all of the RC commands are stored in an UAVObject called ManualControlCommand. Meanwhile, the states of the vehicle can be easily accessed from UAVObjects including accelerate states and attitude states. This is also the way to establish the communication between CC3D and Raspberry Pi 3.

Raspberry Pi Integration. Raspberry Pi [14], series of small single-board computers, could be equipped with operating system. Raspberry Pi 3 Model B is the third generation of Raspberry Pi family. It has the quad core 64bit CPU that has the best performance. To physically connect Raspberry Pi and CC3D, we use the main port of CC3D which can be configured as a serial port and GPIO pin module of Raspberry Pi as it shown in Figure 5. Moreover, To achieve an autonomous system, we utilize an ultrasonic distance sensor to measure distance between the drone and the at ground. An ultrasonic distance sensor transmit from and receive an ultrasonic wave with a single ultrasonic transmitting and receive element to measure proximate distances such as vehicle floor heights or distances to obstacles or pedestrians approaching relative to a vehicle [10]. For the testbed, a low-cost sensor called HC-SR04 is selected. However, The ECHO pin of the sensor is rated at 5V while the GPIO input pins are rated as 3.3V. Therefore two resistors are added to protect the GPIO module. To access the telemetry data in Python, UAVTalk protocol is applied. For instance, below is a code excerpt to get attitude state of the drone.

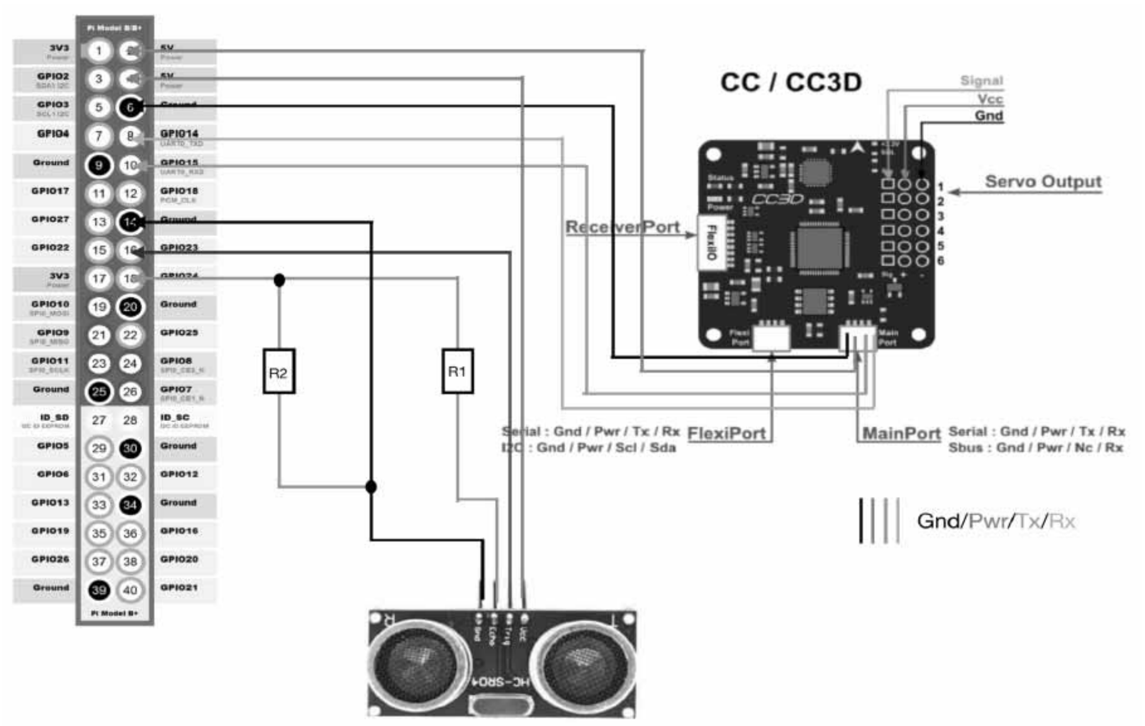
CC / CC3D
Servo Output
ReceiverPort
FlexiPort
MainPort
Serial : Gnd / Pwr / Tx / Rx
I2C : Gnd / Pwr / Scl / Sda
Sbus : Gnd / Pwr / Nc / Rx
Gnd/Pwr/Tx/Rx
R1
R2
HC-SR04

Figure 5. Hardware connection diagram.

```
self.objMan.AttitudeState.metadata.telemetry
    Update-
    Mode=UAVMetaDataObject.UpdateMode.PERIOD
    IC
self.objMan.AttitudeState.metadata.telemetry
    UpdatePeriod.value=50
self.objMan.AttitudeState.metadata.updated()
...
Yaw=self.objMan.AttitudeState.Yaw.value
Pitch=self.objMan.AttitudeState.Pitch.value
Roll=self.objMan.AttitudeState.Roll.value
```

Until here, all of the open source hardwares are integrated together for data communication between the Raspberry Pi 3 and the CC3D.

2 Demonstration

2.1 Workflow

To demonstrate the pipeline that promotes modelling and simulation-based development using open source software and hardware stacks, we developed an autonomous landing controller.

A generic Scilab/Xcos quadcopter model called Generic Quadcopter Simulation (GQS) that employs a proportional-derivative flight controller as a low-level architecture is used as a plant model. GQS model is based on [2].

This generic model can be tailored using parameters to represent a specific platform. The high level architecture is designed in a model-based fashion using Scilab/Xcos. MiL testing is to optimize and verify the controller design for the autonomy feature. Xcos Reuseable and Customizable Code Generator [16] performs as a mean of generating code for the autonomous landing model and to evaluate how good the generated code functions are, a SiL simulation will be tested. For HiL testing, Raspberry Pi 3 executes the code that automatically generated from code generator, A second Raspberry Pi 3 is used as a realtime simulation computer target that enables an UART communication between the plant model and the controller. Finally, to verify the autonomy feature, all of the hardwares stacks is assembled on the drone and flight testing is conducted.

Figure 6 explains the modeling and simulation based workflow applied:

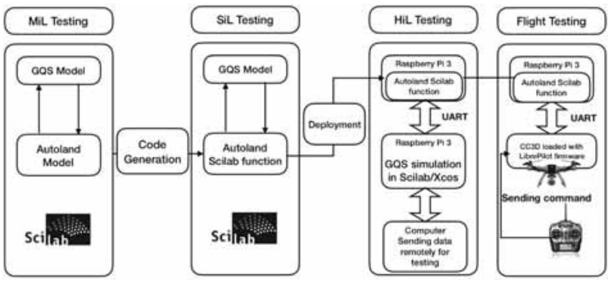

Figure 6. Demonstration workflow.

2.2 Model in-the-loop testing

In order to verify the autonomous landing controller, a MiL simulation (Figure 7) is conducted.

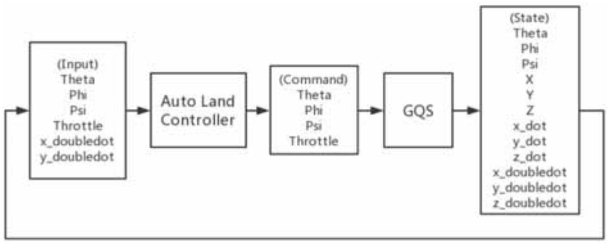

Figure 7. MiL simulation.

We prepared eight scenarios to test that in different situations this system will be working. Figure 8 shows the results for one of the scenarios.

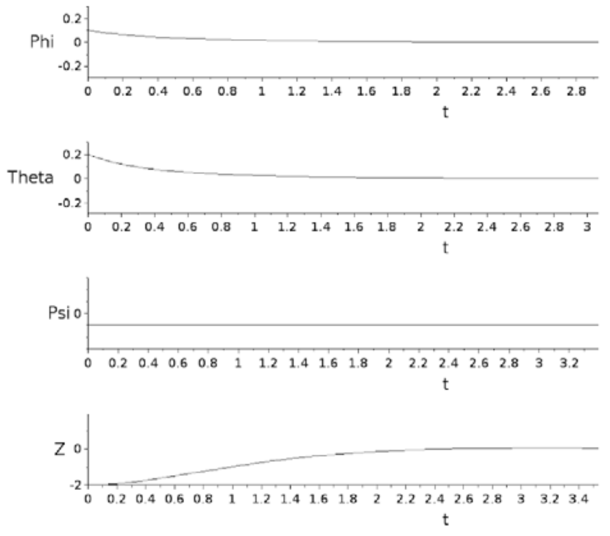

Figure 8. MiL simulation result.

The system can land for all the eight scenarios that means the designed autonomous landing controller is working properly.

2.3 Software in-the-loop testing

After MiL testing, the next step for evaluation is to generate source code out of autonomous landing controller model. As mentioned before, we use Xcos Reuseable and Customizable Code Generator for generating Scilab scripts. The auto generated code is then reintroduced in MiL schema. The same eight scenarios are executed and the results are compared with the MiL results.

2.4 Hardware in-the-loop testing

The HiL testing is essential a further step in testing the autonomous landing feature.

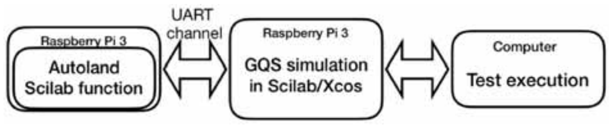

Figure 9. HiL simulation process.

Figure 9 depicts the test setup. The middle Raspberry Pi 3 runs the GQS simulates the drone with controller while the target Raspberry Pi 3 on the left side will execute the Auto Land Scilab function for further verification. It is quite convenient for two Raspberry Pis to establish UART communication by GPIO using RX and TX pins. To achieve UART between Pi and Xcos model, a ATOM toolbox named serial Xcos IO module [18] is used. This module provides a Xcos block to interface real hardware platform for a Xcos simulation via serial ports. It can also be applied into HiL simulation. Originally, the block supports Arduino and provides the bidirectional way to receive C structure input signal from an embedded system and then send back a C structure output signal to the embedded system. On the other hand, to execute autonomous landing Scilab function in Raspberry Pi, a python module called Scilab2Py [19], a mean to seamlessly call Scilab functions and scripts from Python is applied. The result for HiL simulation for the same eight scenarios matched with the MiL results.

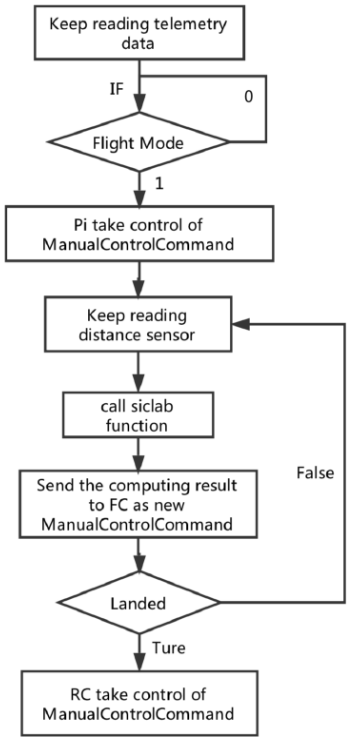
Keep reading telemetry data
IF
0
Flight Mode
1
Pi take control of ManualControlCommand
Keep reading distance sensor
call siclab function
Send the computing result to FC as new ManualControlCommand
False
Landed
Ture
RC take control of ManualControlCommand

Figure 10. Target execution logic.

2.5 Flight testing

Figure 10 describes the execution logic of auto generated Scilab script on the target platform for achieving the autonomous landing.

Raspberry Pi is listening to the flight mode from FC, once the flight mode changes to experiment mode, Pi will take control of the drone and receive the required system states which are sent to Scilab function for computing the movement of autonomous landing. When landed, RC will take back the control of FC.

Once the code is running on the testbed platform the drone is ready to fly (Figure 11).

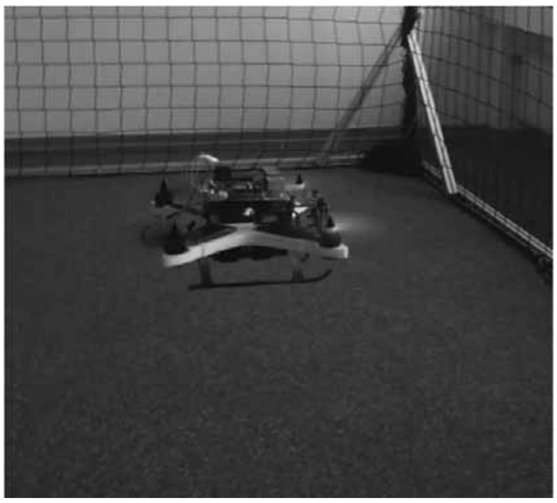

Figure 11. Flight testing.

3 Conclusion

Since the usage of drones is sharply growing while the modeling and simulation based development gets popular in many fields as well, it is quite meaningful to investigate a methodology to combine those two.

In this paper, we utilize the free open source software stacks including Scilab/Xcos which serves as a model design and simulation environment and LibrePilot. We also use open source hardware stacks like Raspberry Pi, HC-SR04 sensor and CC3D. They are sold with low price tags in the market and easy to acquire. The result shows that the pipeline is able to be used for simple autonomy feature design. However, since of all the resources are open source and low-end, the whole system does not perform perfectly. For example, the sensor can be easily broken so that the sensed data is not correct. For the future work, we will be using more reliable products with this pipeline to expect better performance.

After building the pipeline for the Simulation and Model-based development of autonomy features for drones using only open source software and hardware, more students who have interests in aerospace domain can take this pipeline as an guidance for developing their own autonomy features.

References

[1] Barry R. Using the FreeRTOS real time kernel: a practical guide. Real Time Engineers, 2010.

[2] Bouabdallah S. Design and control of quadrotors with application to autonomous flying. PhD thesis, 2007.

[3] La Vern Campbell S, Chancelier JP, and Ramine Nikoukhah. Modeling and simulation in Scilab/Scicos. Springer, 2006.

[4] Clough BT. Metrics, schmetrics! how the heck do you determine a uav's autonomy anyway. Technical report, Air Force Research Lab, 2002.

[5] LibrePilot/OpenPilot community. Coptercontrol/cc3d/atom hardware setup, 2016.

[6] LibrePilot/OpenPilot community. Librepilot documentation, 2017.

[7] Erkkinen T. Model style guidelines for flight code generation. In AIAA Modeling and Simulation Technologies Conference, 2005.

[8] Erkkinen T, Potter B. Model-based design for do-178b with qualified tools. In AIAA Modeling and Simulation Technologies Conference and Exhibit, 2009.

[9] Franklin GF, Powell JD, Emami- Naeini A. Feedback control of dynamic systems, volume 3. Addison-Wesley Reading, MA, 1994.

[10] Iwabuchi M, Ohzawa S. Ultrasonic distance sensor, April 17 1990. US Patent 4,918,672.

[11] Jones E, Oliphant T, Peterson P. {SciPy}: open source scientific tools for {Python}. 2014.

[12] Lee EA, Seshia SA. Introduction to embedded systems: A cyber-physical systems approach. MIT Press, 2016.

[13] Merz T. Building a system for autonomous aerial robotics research. In Proc. of the IFAC Symp. on Intelligent Autonomous Vehicles, 2004.

[14] Raspberry Pi—Teach. learn, and make with raspberry pi. Raspberry Pi, 2016.

[15] Viguria A. Autonomy architectures. In: Encyclopedia of Aerospace Engineering. 1–14, 2016. Proc.

[16] https://forge.scilab.org/index.php/p/xcos-code-generator/

[17] https://www.librepilot.org

[18] https://atoms.scilab.org/toolboxes/wgserialxcosio

[19] https://pypi.python.org/pypi/scilab2py