PILOTS: Embedded Software Testing for High-Power Rocketry Payloads

Daniel Corey[1] and Sarah Dangelo[2]
University of Alabama in Huntsville, Huntsville, AL, 35899

A processor-in-the-loop system was developed for ground testing of software for high-power rocketry avionics and payloads. During flight, these devices are exposed to environmental conditions not easily replicated on the ground. However, flight testing is expensive and occasionally unsafe to perform without verification that the software functions properly. The system extends OpenRocket, an open-source rocket flight simulator, to output simulated flight conditions in real time to an Atmel microcontroller development board. This board emulates the behavior of certain sensors under those conditions. The behavior of flight software over a flight profile can be tested by connecting a device running that software to the system in place of its actual sensors. Emulation of a pressure and temperature sensor has been completely implemented, and emulation of an inertial measurement unit is in progress. The ground test system is actively being used for the development of rocket flight computers and payloads flown by the University of Alabama in Huntsville Space Hardware Club.

Nomenclature

CSV	=	Comma-Separated Values
GUI	=	Graphical User Interface
PCB	=	Printed Circuit Board
HIL	=	Hardware-In-the-Loop
I2C	=	Inter-Integrated Circuit
MIL	=	Model-In-the-Loop
PIL	=	Processor-In-the-Loop
PILOTS	=	Processor-In-the-Loop OpenRocket Testing System
PROM	=	Programmable Read-Only Memory
SIL	=	Software-In-the-Loop
SHC	=	Space Hardware Club
SPI	=	Serial Peripheral Interface
USART	=	Universal Synchronous/Asynchronous Receiver/Transmitter

I. Introduction

The University of Alabama in Huntsville Space Hardware Club (SHC) is an umbrella organization for a wide variety of space and flight related student engineering and science projects. Some of these projects involve flying payloads, controlled by flight software, on high-power rockets. One such project is CanSat, an annual design-build-launch competition organized by the American Astronautical Society and American Institute of Aeronautics and Astronautics.[1] Over the past 10 years, SHC has entered 16 teams in this competition. Though the details of the competition's challenge vary from year to year, the basic structure of the competition stays the same: teams develop and build a re-entry container and a science vehicle payload, which are launched in a high-power rocket and deployed. The container then descends to a competition-specified altitude, where the payload is deployed from the container. Both container and payload then descend at a safe rate to the ground. Depending on the year's challenge, the payload, the container, or both will include electronics and a flight computer onboard to control payload deployment, transmit sensor data telemetry to a ground station, and perform other year-specific tasks. While most subsystems of the CanSats can be tested on the ground, in a lab, or by dropping them from tall buildings, it is difficult to test some sensors and certain aspects of the flight software without actually launching something in a rocket.

[1] Undergraduate, Mechanical and Aerospace Engineering Department, dc0069@uah.edu, AIAA Student Member
[2] Undergraduate, Mechanical and Aerospace Engineering Department, srd0017@uah.edu, AIAA Student Member

A newer club project, Rocketry Avionics, also requires flight software for high-power rockets. This project is an internal effort for the club to develop its own rocketry flight control system, composed of a club-designed printed circuit board (PCB) and a microcontroller running club-written flight software. The project's goals include sensor data telemetry transmission, apogee and flight state detection, and rocket motor ignition. Because the avionics package will be capable of starting rocket motors, it is crucial that the flight software be determined to be working properly and safely before it is flown on a real rocket.

With these projects in mind, this paper's authors, both members of SHC, set out to design a system that would allow testing of flight software on the ground. This system, named PILOTS (Processor-In-the-Loop OpenRocket Testing System), emulates sensors commonly used by SHC CanSat teams and other club projects based on data from rocket simulations run in OpenRocket. Flight software calculations and sensor drivers can be tested using these pseudo-sensors.

II. Background

Because SHC has been participating in the CanSat Competition for almost a decade, some methods of testing flight software have already been developed. However, each of the existing solutions has its limitations, and none of them are sufficient for testing software for Rocketry Avionics.

The most realistic method for testing software is to send up a payload on a real rocket and test its responses under actual flight conditions. The club builds rockets expressly for this purpose and attends monthly rocket launches, at which project teams can fly any electronics or other hardware they have assembled. However, real rocket test flights are expensive, infrequent, and inconvenient. A single flight's motor for the club's CanSat testing rockets this year will cost no less than $76, which is relatively expensive for the club's budget. The monthly launch dates mean that most problems discovered in the software on launch day cannot be corrected and re-tested until a month later. Finally, the nearest launch site to the University of Alabama in Huntsville takes two and a half hours to reach. Launches last from approximately 10 AM to sundown, so teams testing at a launch need to give up an entire Saturday for just one or two test flights.

In addition to the downsides affecting CanSat teams, the club's Rocketry Avionics project cannot rely solely on rocket flights as a testing method due to safety concerns. Even under normal flight conditions, untested software cannot be trusted with something as potentially life and property threatening as igniting high-power rocket motors. Beyond that, the Rocketry Avionics team must also prove that their flight software can respond to abnormal flight conditions in order to be allowed to fly the avionics package on a multistage club rocket. For example, this year's multistage rocket will be flying at the KLOUDBusters event in Argonia, Kansas. The competition rules require that rocket control systems not ignite motors started in flight if the rocket is not oriented vertically.[2] The software's response to such conditions cannot be tested in the real world, since doing so would require creating an unsafe situation.

Some parts of flight software can be replicated less realistically on the ground by recreating the physical sensor environment of a rocket flight. Accurate altitude determination is important because it is typically one of the primary factors driving actions, such as motor ignitions or payload deployments, taken by the software. CanSats have traditionally used atmospheric pressure for altitude determination, and the Rocketry Avionics team plans to use pressure and temperature as well as double-integrated acceleration to reduce error. The Space Hardware Club owns a small vacuum chamber where payloads can be exposed to pressures in the range they would experience during a rocket flight. However, air is evacuated slowly to simulate ascent, and allowed to rush back in almost instantaneously once the target altitude is reached. These rates of pressure changes are nearly the opposite of a rocket flight, where the rocket ascends quickly but descends at a controlled speed. Therefore, the vacuum chamber is not a realistic representation of the operational environment the flight software is exposed. Furthermore, the club does not have an apparatus for changing air temperature, which is sometimes used as a component of altitude calculations, and nothing similar exists or reasonably can exist for simulating the acceleration of a rocket flight in a laboratory environment.

A precursor to this paper's topic was created in 2016 by the first author of this paper and another SHC CanSat team member, Elena Pradhan, as a tool for simulating pressure data from a rocket flight. A processor-in-the-loop system was created in which an Atmel microcontroller development board ran a simulation of a rocket flight using a 4th order Runge-Kutta solver, calculated the expected pressures based on the altitudes it was calculating, and emulated the pressure sensor the project's creators' CanSat team was using. The project described herein builds on that system for improved functionality. The student-made rocket simulation was replaced with OpenRocket, an open-source model rocketry simulation computer program that allows users to design their own rockets and set launch conditions.[3] Compared to the previous simulation, OpenRocket can be more accurate, because it uses a 4th/5th order Runge-Kutta solver rather than a 4th order. OpenRocket also simulates six degrees of freedom instead

of the previous simple one-dimensional simulation. Without any modification, OpenRocket already calculates a variety of different flight and atmospheric conditions that would have been time-consuming to implement in the existing simulation. OpenRocket also allows for customization of the rocket being simulated without modifying the software. Finally, OpenRocket runs on a desktop computer and has a GUI that many SHC members are already familiar with, whereas the existing simulation ran on a microcontroller with no screen. The GUI will allow users to verify that the data their flight software is reading matches the data that the simulation outputs, and allow them to easily modify the simulation to match the rocket they are trying to simulate.

III. System Design and Functionality

A. Overall Structure

PILOTS is a Processor-In-the-Loop (PIL) system. PIL is a testing technique used in the development of embedded systems software. In a PIL system, a development board (the processor) running system code interfaces with a computer simulating the system (also called the "plant"). Other similar "in-the-loop" systems include Model-In-the-Loop (MIL), where a model of the system software runs on the same computer that is simulating the system; Software-In-the-Loop (SIL), where the system software itself runs on the same computer that is simulating the system; and Hardware-In-the-Loop (HIL), where the system's actual control system, complete with software, interfaces with a physical model of the system. PILOTS is considered a PIL system rather than an SIL system because the flight software will not be running on either the computer simulating the rocket flight or the development board emulating the sensors, but instead on a separate processor. PILOTS is also not considered an HIL system because it is not interfacing with a physical environment. In addition, a development board will most likely be used for the flight computer in most tests rather than the actual project PCBs, because development boards are easier to physically connect to the system.

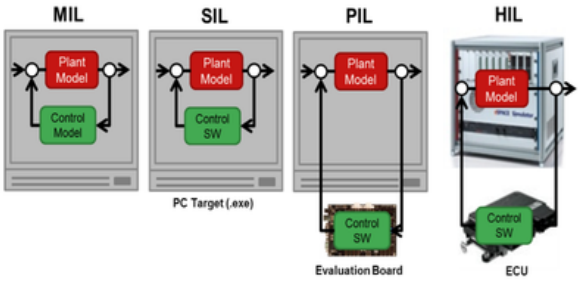

Figure 1. A comparison of Model-In-the-Loop, Software-In-the-Loop, Processor-In-the-Loop, and Hardware-In-the-Loop systems.[4]

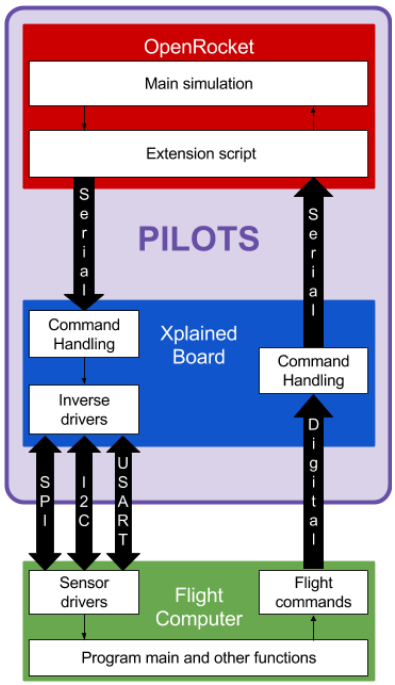

Figure 2. A diagram of the PILOTS structure.

For the plant simulation, PILOTS uses two distinct components: a desktop computer running a modified version of OpenRocket and an Atmel XMEGA-A1 Xplained development board. This is shown schematically in Fig. 2. OpenRocket runs a rocket flight simulation modified by an extension to wait a user-specified amount of time for launch, run in real time, and output flight and environmental data to the Xplained board over a serial USB connection. The Xplained board interprets the data sent to it and, on request from the flight software, converts the information into the values that individual sensors would use. These sensor values are output using the appropriate communication protocols (SPI, I2C, USART, etc.) to the flight software. The flight software then interprets those values and takes whatever actions the users have programmed. In the future, the flight software will also be able to send the Xplained board commands to modify the simulation as it runs. The Xplained board will pass the commands on to the OpenRocket simulation extension, which could trigger flight events such as motor ignition or ejection charges, or change values like fin angle in the simulation.

B. OpenRocket Modifications

OpenRocket is Java-based and was initially developed by Sampo Niskanen at Helsinki University of Technology as his Master's degree thesis.[5] The software uses a standard Runge-Kutta 4th/5th order differential equation solver to simulate flights with 6 degrees of freedom. Starting at a given launch altitude, the net linear and rotational

accelerations are computed about each axis, then used to compute the rocket's linear and angular positions and speeds at some small time step further along in the rocket's flight. This process is repeated for the newly calculated position and velocity until the rocket's flight is complete. This algorithm also assesses the accuracy of the numerical solution and decreases the length of the time step when needed to ensure solution accuracy. The code maintains a time step no higher than a user-specified maximum. For PILOTS, this maximum must be set well below the flight computer sampling interval to ensure up-to-date data.

During simulation, OpenRocket collects flight and environmental data from 54 different variables, including, but not limited to: time; altitude; vertical and total velocity and acceleration; lateral distance, direction, velocity, and acceleration; roll, pitch, and yaw rates; air temperature and pressure; and vertical and lateral orientation. This data can be plotted in the software or exported in a comma-separated value (CSV) format for analysis elsewhere.

Since version 15.03 of the software was released in March of 2015, OpenRocket has supported the addition of extensions, or additional pieces of software that the user can enable or disable to provide a particular behavior or functionality. These extensions can modify the initial conditions of a simulation, as well as modifying or extracting the information tracked by the simulation as it runs.[3]

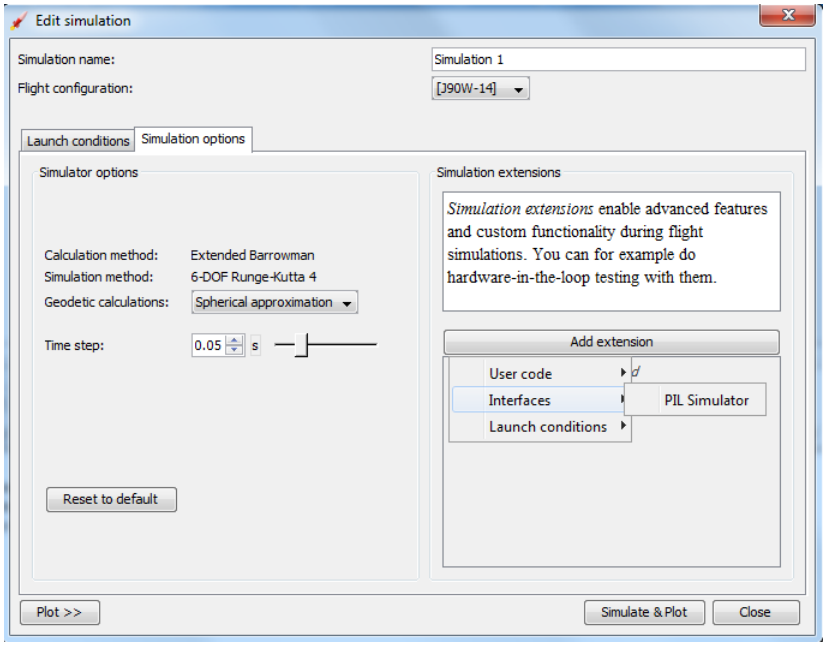

Figure 3. Enabling the PILOTS extension in OpenRocket.

The operational code for PILOTS was written in an extension. The extension is enabled by clicking the "Add extension" button under the "Simulation options" tab in OpenRocket's "Edit simulation" dialog box, then selecting it from a list of available extensions, as shown in Fig. 3. This list of available extensions also includes options to either insert a script or type in the name of a class in OpenRocket to use as the extension. PILOTS software could have been input and run using one of these methods, but including it in the list allowed it to bring up a dialog box when selected. The dialog box was customized in the software to allow users to input information to be used in the extension. PILOTS' dialog box, shown in Fig. 4, allows users to choose an available serial port for data transmission, set the time delay before launch, and enter standard deviation values for sensor noise. After the dialog box is closed, the user clicks "Simulate & Plot" and the extension is initialized. If a serial port was selected by the user, it opens the selected serial port; if the user did not select a port and ports are available, the first port on the list is opened.

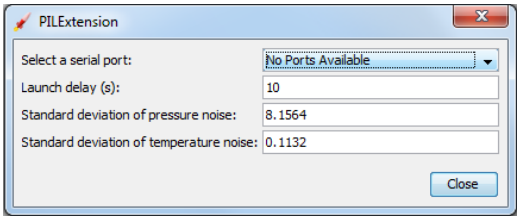

Figure 4. The configuration dialog box.

The extension extends OpenRocket's AbstractSimulationListener, which runs different functions when different simulation events such as time steps or motor ignitions occur. The PILOTS extension uses the postStep function, which fires at the end of each simulation step. The postStep function in PILOTS collects flight data, adds noise, outputs the data over serial to the sensor emulating Xplained board, delays the simulation to run in real time, and triggers initial motor ignition after ten seconds have passed.

The postStep function currently finds the simulation values for air pressure and temperature and will find the values for other variables such as acceleration and orientation in the future. Normally-distributed random noise, using the standard deviations entered by the users when the extension was enabled, is added to the data. If the users

do not choose to enter their own standard deviation, a default value, based on data taken by a CanSat flown in 2016, is used instead. Users can also set the noise standard deviation to zero if they do not want noise in their simulation. Once noise has been added, the data is output over the serial connection opened on initialization in a format that will be discussed in Section C, Embedded Programming.

OpenRocket, unmodified, runs simulations as fast as the computer can compute each time step, and usually finishes the entire simulation within 1-5 seconds. This is considerably faster than most actual flights, which would give the flight computers attached to the system an inaccurate rate of change for the data it receives. The postStep function delays the program by the difference between the total flight time, as tracked by the simulation, and the total runtime of the simulation, measured from the computer's clock. This keeps the simulation running in approximately real time.

Unmodified OpenRocket also only simulates the actual flight of the rocket, starting the moment the motor is ignited. This is not sufficient for testing flight software. Most devices will be turned on before launch manually on the launchpad, meaning that they will be collecting ground-level data for at least the time it takes team members to move to a safe distance away from the rocket. This data taken upon startup is used to establish the initial conditions, including ground elevation, that are used in later calculations. Therefore, it is important to simulate this static position for at least a short time before the simulation begins launch. Additionally, reliably detecting when launch occurs is important for many of the devices considered for PILOTS testing. The PILOTS extension implements SimulationEventListener and overrides its motorIgnition function, which returns a boolean value, and holds it to false until the time set by the user has passed. This keeps other parts of the simulation, which check the motorIgnition function to see if launch can proceed, from firing the initial motor at the beginning of the simulation. Because the other parts of the program that would normally trigger motor ignition are stopped, the postStep function adds the initial motor ignition flight event to the simulation once the set time has passed.

The PILOTS OpenRocket extension currently has one notable deficiency. Setting parachutes to deploy upon motor ejection, which is a certain period of time after motor ignition, causes PILOTS and OpenRocket as a whole to crash during the simulation. The source of this bug has not yet been discovered, but it is considered a low-priority problem, because there is an easy correction. The parachutes can be set to deploy at apogee instead, because the two events will occur at approximately the same time in a properly configured rocket.

C. Embedded Programming

Flight computers interact with their sensors through pieces of software called drivers. Each driver sends the appropriate commands to the sensor to obtain the desired output, reads in that output, and then converts that output to the appropriate format or units. Full testing of an embedded system requires the emulation of sensor behavior using their actual protocols. Since commonly used protocols such as SPI, I2C, and USART cannot be easily used from the desktop computers capable of running OpenRocket, an XMEGA-A1 Xplained microcontroller development board was programmed to act as an intermediary sensor emulation device. This development board includes multiple hardware peripherals for each of the SPI, I2C, and USART protocols, among others.[6] This is an advantage over other commonly used development boards, such as the Arduino UNO, which have only one hardware peripheral for each of these protocols or implement them using software only.[7] Both of these methods would be considerably slower and require more programming than using the Xplained board.

If implemented as a typical polling program, the emulator would go through each peripheral, check if it has received new data, respond if required, and keep checking subsequent peripherals. Between emulating sensors and parsing incoming data, the board may not send responses fast enough to keep up with the flight computer, corrupting sensor measurements. To avoid this, the emulator was implemented as an interrupt-based program. Interrupts are a mechanism by which a computer processor, such as the ATXMEGA used in PILOTS, can be triggered by some event to "interrupt" the normal flow of a program, run a specific piece of code, and then pick back up where the program left off. The PILOTS sensor emulator has prioritized interrupts to handle incoming data from OpenRocket and requests for data from emulated sensors. Providing a sensor value in response to a flight computer query is a highly time-sensitive task, because the real sensors have an expected response time that is often included in the flight software. Therefore, those tasks are prioritized over less-sensitive tasks, such as receiving new data from OpenRocket (see Fig. 5). As the program's main loop can be interrupted for brief periods of time (typically less than 100 microseconds), it is only used for tasks that can be put on hold, such as parsing the data received from OpenRocket into actual sensor measurements and routing it to the actual sensors.

Data are received from the computer in a Comma-Separated-Value (CSV) format. Between commas, each packet has two parts separated by a space: a keyword identifying what data is coming, and the actual data. The first character of the keyword can, optionally, be an axis or identifier, such as x, y, or z for acceleration; or s (static), t (total), or d (dynamic) for pressure. The data value can be an integer or a floating-point number, positive or negative. Incoming characters are stored in a buffer, and then parsed when full data points have been received. The received data is then routed to the appropriate sensor emulating function.

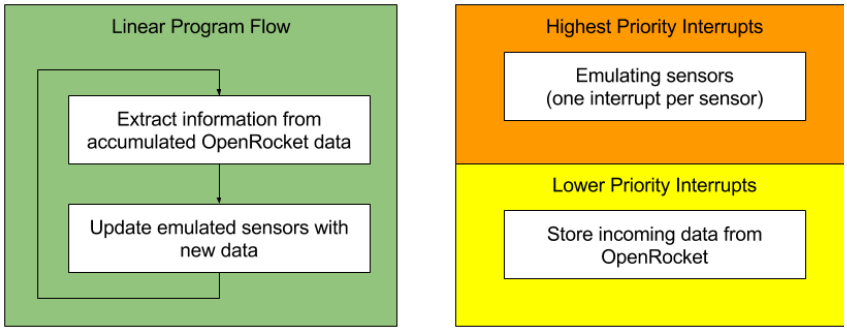

Figure 5. A diagram of the interrupt structure used for sensor emulation.

For each sensor to be emulated, the appropriate conditions, such as pressure, temperature, and acceleration, must be converted to the format used by the sensor. This value must then be communicated to the flight computer using the protocols specified by that sensor. The software responsible for emulating each sensor's behavior was termed an inverse driver, as it acts as an inverse function for the corresponding sensor driver.

Currently, one inverse sensor driver has been fully implemented. The MS5607 barometric pressure sensor is commonly used by the Space Hardware Club for determining altitude on rocketry and weather ballooning projects. It measures both barometric pressure and internal temperature and can communicate over either I2C or SPI.[8] The club commonly chooses to use SPI; therefore, the inverse driver for this sensor also uses SPI. The sensor takes five different kinds of commands: PROM read, reset, D1 conversion, D2 conversion, and ADC read.

According to the MS5607's datasheet, "Each module is individually factory calibrated at two temperatures and two pressures. As a result, 6 coefficients necessary to compensate for process variations and temperature variations are calculated and stored in the 128-bit PROM of each module."[8] These six coefficients can be read using the sensor's PROM read commands. The inverse driver returns the example values for these coefficients found in the datasheet when given these commands.

The physical sensor must be sent a reset command before the PROM values can be output. This was not strictly necessary for the emulated sensor outputs to work, but because the purpose of the system is to test software and all MS5607 drivers would need to include this command, the emulated sensor also will not send PROM values unless it has first been sent a reset command.

The D1 and D2 conversion commands start the sensor's data collection. The D1 value is based on both temperature and pressure, and the D2 value correlates to the temperature. The D1 and D2 values are not the pressure and temperature on their own, but instead must be converted by the sensor driver through a series of mathematical operations that use the coefficients read from PROM. The inverse driver receives pressure and temperature data from OpenRocket and outputs the D1 and D2 values Therefore, the inverse driver performs the inverse of these mathematical operations to determine these values when the conversion commands are sent.

The ADC read command is sent after a D1 or D2 conversion command to obtain the result of the conversion. The sensor emulator, like the real sensor, does not output the D1 and D2 values when it is done calculating them but instead waits for this command.

IV. Performance

The performance of PILOTS was assessed by comparing the output of a flight computer to the simulated data. For this paper, a model of a rocket built for CanSat testing was simulated in OpenRocket with PILOTS enabled, using the default configuration values: 10 second launch delay, 8.1564 Pascals pressure noise standard deviation, and 0.1132 degrees Celsius temperature standard deviation. The PILOTS sensor emulator board was connected to a development board used on an SHC weather ballooning project, running an unmodified driver for the MS5607 pressure and temperature sensor. Data exported directly from OpenRocket's simulation is compared to the output of the weather ballooning board in Fig. 6 and 7. Both graphs show how the values read by the board closely follow the OpenRocket simulation values. Figure 7 also clearly shows the added noise.

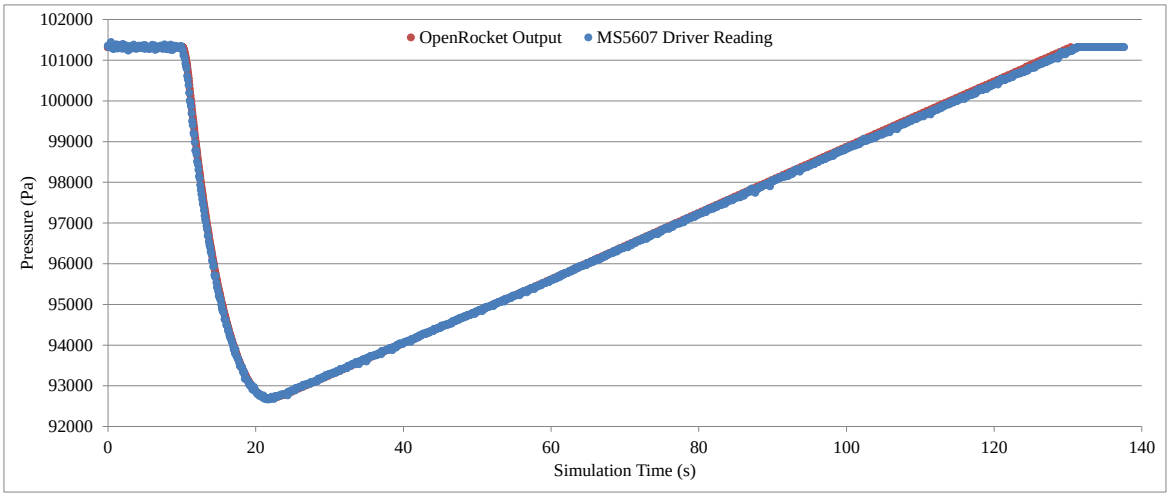

Figure 6. OpenRocket and flight computer pressure measurement comparison.

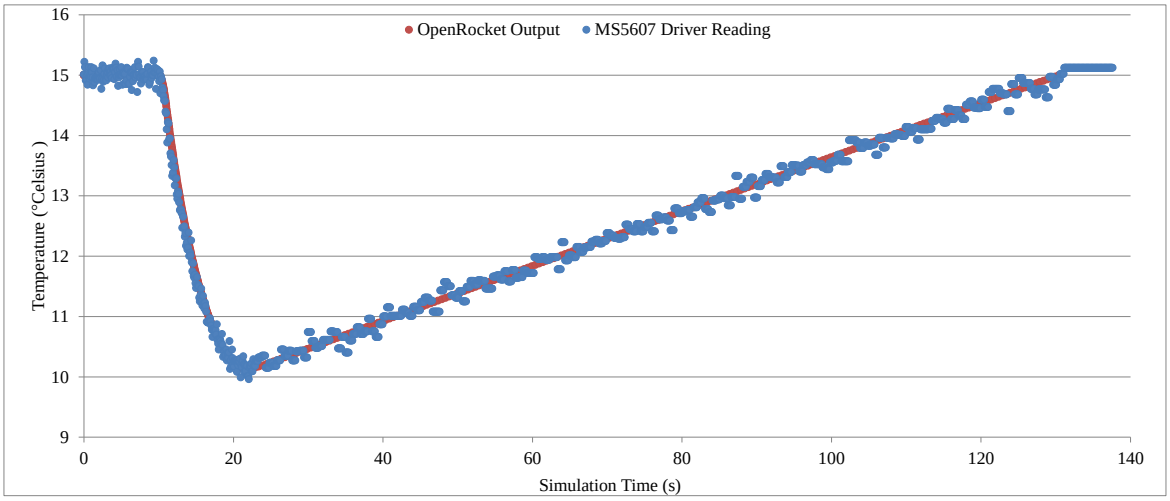

Figure 7. OpenRocket and flight computer temperature measurement comparison.

V. Conclusion

In its current state, PILOTS is a valuable tool for validating the flight software of the two teams SHC is sending to the International CanSat Competition in June. It will also aid in testing of some basic functions for SHC's in-house Rocketry Avionics project. However, multiple opportunities for improvement exist, and the authors intend to pursue them.

The next step in PILOTS' development is incorporating more sensors. In addition to the pressure and temperature values, the command structure used to send data between OpenRocket and the Xplained board sensor emulator is already programmed to handle acceleration, rotational velocity, and magnetometer heading. Therefore, these values will be procured in OpenRocket and transmitted. Inverse drivers will be developed or written for

sensors that can use these and other kinds of data. These sensors include the MPU-9250 inertial measurement unit, a to-be-selected GPS unit, and an analog temperature sensor. With the addition of more sensors, PILOTS will be both more useful for CanSat teams and an essential tool for Rocketry Avionics testing during development.

An essential feature for full "in-the-loop" simulation of a flight is feedback from the flight computer to OpenRocket. This will allow the flight computer being tested to trigger a flight event, such as a parachute deployment or motor ignition, and cause that event to occur in the OpenRocket simulation.

Another area for development is the user interface to the OpenRocket extension. Easier configuration of settings such as the physical pins used for each sensor will assist users of PILOTS in conducting their desired tests quickly and easily.

Acknowledgments

The authors would like to thank the University of Alabama in Huntsville Space Hardware Club for providing a motivation for this project and the computers, lab space, and Xplained boards used for development and testing. Dr. Francis Wessling and Dr. Brian Landrum served as advisors for this paper. The authors would also like to thank and acknowledge Elena Pradhan for her work on the pressure sensor emulation project that laid the groundwork for PILOTS, especially in determining the calculations for converting pressure data into the D1 and D2 values used in the MS5607 inverse driver.

References

[1]CanSat, "CanSat Competition," [website], URL: http://www.cansatcompetition.com/ [cited 13 February 2017].

[2]Tripoli Kansas #34, "KLOUDBusters Rules for Multi-Staged Rockets," [organization regulations], URL: http://www.kloudbusters.org/images/stagerules.pdf [cited 9 February 2017]

[3]OpenRocket, Software Package, Ver. 15.03, URL: http://openrocket.sourceforge.net/index.html

[4]Serughetti, Marc, "Accelerate Automotive Dev Time: Fill Hardware-in-the-Loop Gaps," *EE Times* [online periodical], URL: http://www.eetimes.com/author.asp?doc_id=1327063 [cited 9 February 2017]

[5]OpenRocket, "OpenRocket," [website], URL: http://openrocket.sourceforge.net/ [cited 9 February 2017].

[6]Atmel, "Atmel AVR1924: XMEGA-A1 Xplained Hardware User's Guide," [online documentation], URL: http://www.atmel.com/images/doc8370.pdf [cited 10 February 2017].

[7]Arduino AG, "Arduino - ArduinoBoardUno," [online documentation], URL: https://www.arduino.cc/en/Main/ArduinoBoardUno [cited 9 February 2017].

[8]TE Connectivity, "MS5607-02BA03," [datasheet], URL: http://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&DocId=Data+Sheet%7FMS5607-02BA03%7FB%7Fpdf%7FEnglish%7FENG_DS_MS5607-02BA03_B.pdf [cited 9 February 2017].