R-Log: A Low-Cost Low-Power Solution for Data Logging on Payload Missions

Robert Belter
University of Colorado Boulder
Robert.Belter@colorado.edu

Abstract

Throughout most scientific payload missions, especially high altitude balloon and sounding rocketry payloads, there is a certain set of base data that must be recorded. Generally, this data includes altitude, temperature, and acceleration, and can also include gyroscopic and magnetic field data. Typically, this means that every time a payload is created, a significant amount of work goes into designing and building the hardware and software necessary to record this data. This paper documents the design and testing of the “R-Log” a small, low-cost, low-power solution to this problem. The R-Log contains a 3-axis accelerometer, gyroscope, and magnetometer in addition to a combined temperature and pressure sensor. It is also capable of performing sensor fusion to create absolute orientation data and logging all of this information to an onboard microSD card. Additionally, it has an expansion IO header and is fully Arduino compatible to allow for easy integration with other payload systems. This paper will also discuss the development of data visualization tools that allow for the data from the R-Log to be rendered into 3D display video. Finally, it will discuss the results of testing the R-Log, which will be achieved through performing multiple test flights, hopefully on a variety of different platforms.

Contents

1. Introduction .................................................................1
   1.1 Data Loggers and Payloads .........................1
   1.2 Problems with Current Methods ............2
2. R-Log Design and Process ..............................................2
   2.1 Requirements .................................................2
   2.2 Design Methodology ....................................4
   2.3 Design Solution ..........................................4
3. Data Visualization Software ..........................................5
   3.1 Motivation and Requirements ..................6
   3.2 Implementation ..........................................6
4. Testing...........................................................................7
5. Concluding Notes ..........................................................7
6. Appendix.......................................................................8

1. Introduction

1.1 Data Loggers and Payloads

Data is an essential part of any scientific operation, and payload missions are no exception. The end goal of most missions is to record and analyze data. Additionally, for any payload that does not have the capability to transmit data, it must also be able to store the data in some form. In order to do this, most payload missions develop data logging functionality similar to the following diagram.

In this model of a data logger, there are sensors, that actually gather data from the physical world; a controller, generally a microcontroller such as an
Arduino which interfaces with all of the sensors; and some sort of storage medium. There can also be outputs which are controlled by the controller depending on the data (e.g. opening a hatch once a specific altitude is reached). While this model may not completely describe all payloads, most are either reducible to or contain a model very similar to this one.

Generally, during the development of a payload system a significant amount of effort is devoted to developing data logging capabilities. Depending on the scope of the payload, this can even be where the majority of the payload’s development occurs. However, there is generally a large amount of overlap between the data logging needs of different payloads.

1.2 Problems with Current Methods
Developing the required data logging capability to meet the data needs of a payload mission can take up a significant part of the work for developing a payload. Reuse of data logger designs is possible, but can still take up time due to differences in controllers and differing software and hardware requirements. Having to recreate hardware and software designs can take almost as much time as designing from scratch, giving limited benefit. Taking additional time for development also means that some non-system-critical features may have to be left off or underdeveloped, even if they would provide beneficial data.

Additionally, it is not within the scope of many payload missions to create customized PCBs. As such, they are reliant on the microcontrollers and sensor breakout boards that are available as commercial items. While there is a sufficiently large amount of options to cover the needs of the majority of payload missions, they create unnecessary demands on weight, power, and especially space requirements.

The solution that this paper will present, is to encapsulate some level of core functionality into a single reusable module (the R-Log). This model allows for recording this set of core data, but also the extendibility to include other sensors and outputs, and to be able to interface with an additional control module for payloads that require more advanced functionality. This model for data logging is displayed in the image below.

2. R-Log Design and Process

2.1 Requirements
With an understanding of the current problems in data logging for payloads, it is now possible to define the minimum requirements for the solution to meet. There are three main areas of requirements to examine: functionality, space and power, and ease of incorporation.

Functionality
Functionality, as it will be used for the purpose of this discussion, defines the minimum standards for the data recorded, and the format in which it is done. In order to define what data was necessary to record, I took an approach looking at the usefulness of data.

The most important data to be collected for the majority of payloads is altitude data. In a sense, rocket and balloon payloads exist primarily to perform scientific measurements at altitude, so it is difficult to conceive of an experiment for which altitude has no bearing. As such, the ability to record altitude data is one of the most important requirements of the system.

On a technical level, the only way to find altitude is to measure pressure. Most methods to sense pressure also require temperature for calibration. As
such, the requirement to record altitude can be decomposed into these two requirements.

Many payloads, especially sounding rocketry payloads, also require acceleration data. Even if acceleration data is not strictly required for scientific purposes, it is often extremely helpful to determine significant events during the mission’s lifetime (e.g. launch, parachute deployment, landing, etc.). Acceleration often gives clearer indications of these events and provides timestamps with much less uncertainty than altitude data.

In addition to acceleration data, rotation data can often be very useful for calibrating payload data. Even systems that do not have a specific need for rotation data may find it useful for determining overall stability of the payload platform. Adding in a rotation sensor leads to using an inertial measurement unit (IMU), instead of using separate sensors.

Although IMUs can be defined in a broader sense, for this paper I will examine it as a device capable of measuring three axes of acceleration and rotation data, and optionally three axes of magnetic field data. Notably, this data can be combined in a process called sensor fusion to create absolute orientation data.

The combination of an IMU and on board processing creates an attitude and heading reference system (AHRS). The advantage of using an AHRS system over just using an IMU is the ability to determine absolute orientation (e.g. roll, pitch, and yaw). Absolute orientation can be an extremely valuable tool for payloads, so it was added as a requirement, alongside the ability to record the raw data (3-axis acceleration, rotation, and magnetic field data).

**Space and Power**

In order to provide an easy to use system for payloads it is important to not make a significant impact on either their available space or battery life. To accomplish this, a minimal space and power draw had to be one of the requirements. However, the definition of what is sufficiently minimal can be somewhat vague.

With regards to space requirements, the easiest place to start is by requiring a single board design. The normal method of implementing a data logger in payloads is to use multiple boards, so a different multi-board implementation would not offer much improvement. However, the required size of the board is slightly more difficult to define. As the required functionality did not impose much in terms of space requirements, the board was simply created to be as small as possible, without needing to be concerned with a specific envelope.

Power concerns were also very straightforward. Because many of the devices are dependent on 3.3V logic, using that as the board’s operating voltage was the obvious choice. Additionally, as the other payload systems may be running at a different operating voltage, an onboard regulator was required.

The approach taken was to assume that the data logger would be given its own battery (this is only assumed for the purposes of determining reasonable power draw and does not create a requirement for how it is implemented in a payload), and to scale our maximum power draw based on reasonable battery sizes and mission lengths. Although these numbers are somewhat arbitrary, a lifetime of 12 hours and a battery size of no more than 1000mAH would be able to support the majority of payloads’ requirements. This gives us a maximum current draw of

\[
1000\text{mAH} \div 12\text{H} = 83.3\text{mA}
\]

Note that this is current draw on the unregulated voltage input.

**Ease of Incorporation**

Although an easy area to overlook, in many ways the ease of incorporating this system into other payloads is the most important design requirement. The simplest way to make the payload easy to understand and use was to make it compatible with the Arduino IDE. The Arduino IDE provides an easy to use platform, and is simple to work with, even without previous experience. In order to support being programmable through Arduino in an convenient manner, programming through USB was also considered a requirement.

One necessary feature is that the data must be available during the mission, and not only afterwards. Otherwise, payloads that require this data to perform command operations would be forced to add sensors that perform the same tasks. Additionally, it needed to be easy to incorporate additional sensors. A general purpose IO header was the easiest way to accomplish both interfacing with payloads and providing support for additional
sensors. In order to support standard sensor interfaces, it needed to have support for I2C, and several GPIO pins. SPI was not included, due to the much larger number of communication lines that it requires.

The requirements that are derived above are summarized in the table below.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functionality</strong></td>
<td>Required capabilities related to data collection and storage</td>
</tr>
<tr>
<td>Altitude</td>
<td>Ability to sense temperature and pressure, and fuse them to determine altitude</td>
</tr>
<tr>
<td>AHRS</td>
<td>Have a 9DOF IMU (3-axis accelerometer, gyroscope, and magnetometer) and the ability to perform sensor fusion.</td>
</tr>
<tr>
<td>SD card</td>
<td>Ability to write data to an SD card in .csv format</td>
</tr>
<tr>
<td><strong>Space and Power</strong></td>
<td>Maximum space and power demands</td>
</tr>
<tr>
<td>Voltage</td>
<td>Ability to be powered from a single lithium cell, 3.7V or a regulated 3.3V source</td>
</tr>
<tr>
<td>Current</td>
<td>Maximum average current draw of 80mA</td>
</tr>
<tr>
<td>Single Board</td>
<td>All components must be placed onto a single board</td>
</tr>
<tr>
<td><strong>Ease of Incorporation</strong></td>
<td>Capabilities necessary to ensure ease of integration with payloads</td>
</tr>
<tr>
<td>Arduino Compatible</td>
<td>The board must be programmable through the Arduino IDE</td>
</tr>
<tr>
<td>USB Programming</td>
<td>An on-board USB connector, through which the board can be programmed</td>
</tr>
<tr>
<td>Robust IO interface</td>
<td>A IO header that meets the following standards: at least two general purpose digital pins and two analog pins, an I2C bus, and UART serial.</td>
</tr>
</tbody>
</table>

2.2 Design Methodology

The methodology behind the design of the R-Log was very straightforward. The design requirements were met in the order of: ease of incorporation, functionality, then space and power.

First, the micro controller and supporting circuitry were selected to ensure that the minimum requirements for usability were met. Second, sensors and supporting circuitry to support the data collection and storage functionality were selected. Finally, these were combined into a single board in order to meet the space and power requirements.

2.3 Design Solution

During the development process, the R-Log went through a variety of different revisions, between which many changes were made. This section will document both the development process and the justification behind the design decisions.

The first major design decision to be made was what microcontroller to use. In order to provide the easiest integration into the Arduino IDE, it was decided to use one of the microcontrollers from the most popular Arduino boards. Initially, an ATMega32U4 was selected, because of its integrated USB communication capabilities. However, during testing it was discovered that the 32U4 was somewhat unreliable to program in the Arduino IDE, occasionally requiring hard resets in order to be programmed.

Because of the difficulties with the 32U4, the microcontroller was switched to an ATMega328p. This is the controller used in the Arduino Uno, and is very well supported in the Arduino IDE. This change also required the inclusion of a USB – Serial chip, in order to support the requirement of programming over USB. A FT230X chip was selected for its small footprint and the ability to use the standard drivers for the device that FTDI provides.

Because of the difficulties with the 32U4, the microcontroller was switched to an ATMega328p. This is the controller used in the Arduino Uno, and is very well supported in the Arduino IDE. This change also required the inclusion of a USB – Serial chip, in order to support the requirement of programming over USB. A FT230X chip was selected for its small footprint and the ability to use the standard drivers for the device that FTDI provides.

The next components to be selected were the sensors. For the pressure and temperature sensors, a BMP180 was selected. I have used it in a variety of data loggers and have found it to be reliable and easy to implement. However, during the development of the R-log the BMP180 was placed into end-of-life by its manufacturer. Rather than merely update it to the BMP280, its replacement, I opted to use the BME280 which, despite also including a humidity sensor, has a smaller footprint than the BMP180 and similar power requirements.
For the IMU, initially a MPU9050 was selected. It has the required nine degrees of freedom and can do some rudimentary sensor fusion. However, this sensor was eventually replaced by the BNO055, for its superior ability to perform sensor fusion.

The necessary circuitry to support a SD card was only a microSD socket, due to the board’s 3.3V operating voltage. Thus, its addition was straightforward, although it was the component with the largest board space requirement.

Next the circuitry to support the power needs of the board were selected. Because the board needed to be able to be powered from either a battery input or USB, it required power ORing circuitry in addition to 3.3V regulator.

The above covers the justification behind the components selected, to see more information about the circuitry and a detailed schematic, see Figures 1-6 in the appendix.

After the creation of the schematic, the components were laid out onto a PCB design. In order to perform both the schematic capture and PCB design, the DesignSpark software from RS components was used. Initially, a 2-layer PCB was used, but this was later switched to a 4-layer PCB to obtain a smaller space envelope. This switch shrunk the size of the board from 2”x1.1” to its current 1.5”x0.875” size. In order to support easy mounting, 1/8” mounting holes are provided on two corners of the board. A header row on the bottom provides the standard IO interfaces, while a two pin header on the top allows for easy connection to an external power source.

After the design of the board, an Arduino library was developed for use with the board. This library includes libraries for communication with all devices on the board, as well as the ability to provide standard capabilities to easily obtain and record data.

See opposite for a table detailing the design, organized by the requirements in section 2.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functionality</strong></td>
<td>Required capabilities related to data collection and storage</td>
</tr>
<tr>
<td><strong>Altitude</strong></td>
<td>BME280 sensor supports recording pressure, temperature, and humidity, which can be fused to provide altitude data.</td>
</tr>
<tr>
<td><strong>AHRS</strong></td>
<td>BNO055 is capable of providing absolute orientation data in a variety of formats.</td>
</tr>
<tr>
<td><strong>SD card</strong></td>
<td>On board SD card socket</td>
</tr>
<tr>
<td><strong>Space and Power</strong></td>
<td>Maximum space and power demands</td>
</tr>
<tr>
<td><strong>Voltage</strong></td>
<td>Can be powered from regulated 3.3V, or any power source from 3.7V-12V</td>
</tr>
<tr>
<td><strong>Current</strong></td>
<td>Average current draw of ~ 40mA. On a 1000mAH battery this corresponds to ~25 hours of data logging time</td>
</tr>
<tr>
<td><strong>Single Board</strong></td>
<td>Single 1.5”x0.875” board</td>
</tr>
<tr>
<td><strong>Ease of Incorporation</strong></td>
<td>Capabilities necessary to ensure ease of integration with payloads</td>
</tr>
<tr>
<td><strong>Arduino Compatible</strong></td>
<td>The board can be programed through the Arduino IDE, and a library exists for easy use with the board.</td>
</tr>
<tr>
<td><strong>USB Programming</strong></td>
<td>The board has a USB connector, through which it can be powered/programed</td>
</tr>
<tr>
<td><strong>Robust IO interface</strong></td>
<td>The on board IO includes: 4 DIO pins, 2 Analog inputs, an I²C bus, a UART (Serial) bus, and GND and 3.3V connections.</td>
</tr>
</tbody>
</table>

### 3. Data Visualization Software

In addition to the R-Log board, a 3D data visualizer was also developed as part of this project. It allows for easy and quick analysis of data files recorded on the R-Log. It also provides a more useful output for analysis than scatterplot graphs.
3.1 Motivation and Requirements
The motivation behind the R-Log was to provide an easy, reliable system for performing data logging on payload missions. However, because of the generic types of data being recorded it was also possible to create a generic data analysis tool. While it is impossible to cover all of the possible needs for data analysis that a payload could require, it is possible to provide a more useful interface for data analysis than merely providing the raw data.

The main goal of the data analysis software would be to provide a visualizer that would display information about the flight in a useful manner. The most natural way to understand orientation data is to visually see the orientation of the object. Thus, the main desired feature for the data visualizer was the ability to render video of a 3D model to display the absolute orientation data.

Other desirable features for the data logger were real time updates on altitude and a timestamp. In order to support this sort of information, a text readout was added in the corners of the display that would support any data that was desired. This includes the standard data that is logged by the R-Log, but can also include data from other user sensors.

Another feature that the data visualizer needed was the ability to parse the raw .csv files recorded from the R-Log. This allows for "plug and play" compatibility with the data, such that immediately after flight it can be directly used in the visualizer.

Finally, in order to make the visualization software easy to use and understand, it needed a graphical interface that was sufficient to perform the majority of desired visualization behavior.

3.2 Implementation
In order to support the required 3D transformation and rendering capabilities, the open source 3D modeling software Blender was used. Because Blender provides built-in Python support, Python was used as the scripting language to implement the other functionality of the data visualizer as well.

Another advantage that comes with using Python is that advanced users can modify the scripts to support behavior that was not already supported, without having to worry about recompiling the visualizer.

I had written scripts for this sort of data analysis for past payload missions, which provided me a decent start for the operations that I wanted to do. The next step, was to make the code more general, as well as more easily understood by a user.

To accomplish this, I reorganized the script into the following hierarchy:

In this hierarchy, datavisualiser.py is the file that performs most of the heavy lifting related to running the 3D transformations and rendering. Additionally, it is the only file that relies on the Blender API. By making all of the other files Blender independent, they can be edited by a user with only a standard knowledge of Python, without needing to know anything about Blender.

Runfile.py performs the actual rendering operations, using the functions defined in Datavisualiser.py. It relies on both Definitions.py, which contains the definitions for the data import and the lambda expressions which handle the text displays, and Settings.ini, which provides the settings for the render, which are editable in the GUI (frame rate, resolution, etc.). The lambda expressions in Definitions.py can be modified by advanced users to allow for more flexibility in what information is displayed.

Finally, Application.py provides a GUI for editing Settings.ini and launching Runfile.py within Blender. For an image of the application interface, see Figure 7 in the appendix.

An example picture of what the rendered output looks like is shown below.
4. Testing

Unfortunately, the final revision of the R-Log has not been received from the manufacturers before the writing of this paper. The preliminary tests from prior revisions have included recording data onto an SD card from all attached sensors, verifying that the fused orientation correlates to that experienced physically, and measuring current draw. More intensive testing will be performed once the final revision of the board, containing the BME280, is received.

Testing will be done using a variety of different launch platforms. One way in which data will be gathered early on is through the use of model rockets. Because of the R-Log’s extremely small size, it can be used on nearly any model rocket, provided that it does not unbalance the rocket. Additionally, testing on high altitude balloon payloads is desired, but is contingent on getting permission to be on a suitable launch.

5. Concluding Notes

The community surrounding scientific payloads, especially those made by students and hobbyists has grown tremendously in the recent past. It has become easier than ever to develop payloads with easier to use microcontrollers such as Arduino and sensor breakout boards from companies such as SparkFun and Adafruit.

The R-log is a natural next step for data loggers in payloads. It is a complete system that is small, lightweight, easy to use, and has sufficient functionality to cover the needs of the majority of payloads. Additionally, the generic form of the R-Log allows for better data visualization tools to be developed, making it easier to perform post flight analysis of data.

The design, build, and testing of the R-Log has been a great learning experience, and I look forward to continuing to develop it to be even better suited for payload applications, with input from the community. The source code for the R-Log library and data visualizer tool is available at the links given in the appendix.
6. Appendix

Figure 1- Power Circuitry
Figure 2: Connectors
Figure 3: USB and USB-UART
Figure 4: Processor and Supporting Circuitry
Figure 5: IMU
Figure 6: Combined Pressure, Temperature, and Humidity Sensor
Figure 7 - Interface for Data Visualizer

Links to code repositories

<table>
<thead>
<tr>
<th>R-Log Arduino Library</th>
<th><a href="https://github.com/robbotorigami/RLogLib">https://github.com/robbotorigami/RLogLib</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Visualizer</td>
<td><a href="https://github.com/robbotorigami/DataVisualizer">https://github.com/robbotorigami/DataVisualizer</a></td>
</tr>
</tbody>
</table>