HyRo
Team 28

Jason Klindtworth | Josh Asher | Layne Nolli

CS461

Fall 2016

Technology Review

Abstract

November 13, 2016

Contents

I. Introduction

II. Technology Reviews

A. Generating and Capturing of Data on a Micro Controller

Onboard the hybrid rocket there will be a collection of sensors, controllers, and a radio transceiver that data will need to be collected from and transferred too. A micro controller will be connected to all these devices to collect and transfer this data amongst components. There are multiple options to choose from when it comes to which micro controller to choose and what API to access the communication port with. Different languages allow for access to different APIs to communicate with system devices.The three options being examined in this section are as follows.

1) Options:

- Using the Python Language and its API (PyUSB) to serial communications ports and GPIO lines with the Beagle Bone Black (BBB) micro controller.
- Using the C/C++ Language and its API to serial communications ports and GPIO lines with the Beagle Bone Black micro controller.
- Using either C/C++ or Python on a different micro controller like the RaspberryPI 3.

2) Goals: The goals in choosing the technology to generate and capture data on a micro controller are (1) choose a technology with simplicity as to allow data to be easily captured and generated. (2) To choose a technology that can communicate do these functions quickly (3) To choose a reliable platform that is not prone to errors.

3) Criteria: The criteria that these choices will be based on are: speed of language, complexity of communications USB API, complexity or existence of GPIO API, Processor of Micro Controller, amount of RAM on micro controller, amount of long term memory on micro controller, and other capabilities or packages the micro controller comes with.

4) Comparison Table: The next two tables compare criteria between the 3 options.[1][2][4]

Criteria Comparison Table 1				
	Speed of Language	Complexity of USB API	Processor	RAM
Python on BBB	Medium	Medium	AM335x 1GHz ARM Cortex-A8	512MB DDR3L (800 MHZ)
C/C++ on BBB	Fast	Medium	AM335x 1GHz ARM Cortex-A8	512MB DDR3L (800 MHZ)t
Raspberry PI 3 with either language	Medium or Fast	Medium	1.2GHz 64-bit quad-core ARMv8	1GB LPDDR2 (900 MHz)

This next table is an extension to the previous.

Criteria Comparison Table 2			
	Long Term Memory	Other capabilities/packages	Price of Micro Controller
Python on BBB	4G Embedded + MicroSD expansion	69 GPIO, I2C, 4 Serial Ports (lots more not specific to the project)	$55
C/C++ on BBB	4G Embedded + MicroSD expansion	69 GPIO, I2C, 4 Serial Ports (lots more not specific to the project)	$55
Raspberry PI 3 with either language	MicroSD Storage	17 GPIO	$40

5) Discussion: Using the Python Language and its API (PyUSB) to serial communications ports and GPIO lines with the Beagle Bone Black (BBB) micro controller will provide a reasonably fast system with plenty of storage at a slightly higher price. One of the main allures to this option is Python. Python tends to allow for faster development time because of its simplistic syntax and well documented USB interface, PyUSB [PyUSB]. Python tends to run slower than C or C++ which can be used on this platform to accomplish the same goals, but results from last year show that it works at an adequate speed to perform serial communication and GPIO communication. Another pusher for Python is that last year's team used it as the language for their system. PyUSB shares a similar complexity to libusb which C or C++ would use. Either choice would have an appropriate learning curve. Using Python would make the code very readable to future teams, but does suffer some in performance. The micro controller environments that Python runs on will have an accept on performance also. Python Running on the BBB has the advantage of access to more communication ports than it would on the Raspberry PI 3. The processing power is slightly less than that of the PI, but we do benefit from fast on board flash storage where our software can run. [3]

Using the C/C++ Language and its API to serial communications ports and GPIO lines with the Beagle Bone Black micro controller would provide faster serial and GPIO communication than using Python. Running on the BBB C/C++ would have access to the same environment as Python and the only other major benefit beside speed would be memory control. C/C++ provides the ability to access and store memory in more detail/control than Python does. Since we are not planning on any operations that are memory or CPU intensive I did not include this as criteria. C/C++ would be more time consuming and complex than Python, but not to a large degree. This would also have to be built from scratch, but in the end, would provide faster communication. This might not be required if the speed of the controllers, sensors, or radio transceiver are not as fast as the C/C++ software could produce. This will be tested down the road when the system begins to be built. [3][5]

Using either C/C++ or Python on a different micro controller like the RaspberryPI 3 is listed as an option to compare the abilities of a different micro controller to the one that was used last year. First of the RaspberryPI 3 is at least $15 cheaper than the BBB. It also has a processor that is more powerful (that is judging the speed and core count) than the BBB. The quad core would be great for multithreading and there is twice as much RAM on the RaspberryPI. Though, it uses DDR2 which has half the number of transfer per cycle than DDR3. I believe the RaspberryPI would be a great cheaper solution, but it does not support as many low-level communications ports as the BBB. The BBB has also been proven a successful option from last years' experience.

6) Beagle Bone Black with Python and PyUSB: It was not easy to decide on which combination of these technologies would be the best for our Hybrid rocket system. The major influencing factor on my decision is that the team already has access to a BBB from last year and that component would not have to be purchased again this year. Further the team last year started to develop on this board using the Python language as the solution to collecting and generating data last year successfully. This has influenced me to choose this combination for this year's Hybrid rocket onboard system. C/C++ might be faster but this is not significant enough to merit a switch to those languages.

B. Generating and Capturing of Data on a Traditional Computer

On the other end of our system we will need to communicate with a USB device on a traditional computer to collect and pass generated data. There are many different APIs to access the serial devices connected to a computer depending on language and operating system. We could be targeting Windows or Linux when we build this software. The three options being examined in this section are as follows.

1) Options:

- Using Python with PyUSB on either Windows or Linux.
- Using C++/C# with WinUSB on Windows.
- Using C/C++ with libusb on Linux..

2) Goals: The goals in choosing the technology to generate and capture data on a traditional computer are (1) choose a technology with simplicity as to allow data to be easily captured and generated. (2) To choose a technology that can communicate do these functions quickly (3) To choose a technology that can run under a single operating system at minimum, but be preferable to run under at least 2 different operating systems.

3) Criteria: The criteria that these choices will be based on are: speed of language, complexity of communications USB API, and the ability for the technology to run on at multiple operating systems.

4) Comparison Table: Comparison of Options vs criteria

Criteria Comparison Table			
	Speed of Language	Complexity of USB API	Ability to run on multiple operating systems
Using Python on Windows or Linux	Medium	Medium	Yes, with only slight modifications.
Using C/C++ on Windows	Fast	Medium	Only Windows
Using C/C++ Windows or Linux	Fast	Medium	Yes, depending on the presence of OS specific code in the software.

5) Discussion: Using Python with PyUSB on either Windows or Linux provides a slower, but easy to access serial interface that will meet requirements in this part of the project. Python does not preform as fast as C/C++ when compared in bench marks, but its performance will be adequate for our situation. It also provides quicker development times and code that is easier to read. Python would coincide with the language chosen to be used on the BBB and if the Python is chosen for visualization. It's preferable to have all the pieces of the software written in the same language. Python also provides a API that can be easily ported between Windows and Linux allowing us to potentially run our software on multiple operating systems with only slight modifications.[7]

Using C++/C# with WinUSB on Windows would promise to be faster than running python on Windows. WinUSB is well documented and has many examples to help someone get started with USB communication. This would be a good option if the only desired operating system is Windows. If performance becomes a problem this would be a good option also. It is however preferable to have a cross compatible solution as the engineering team tends to use both Windows and Linux, so this option does not stand a high chance of being chosen.

Using C/C++ with libusb on Windows or Linux provides both a speed boost and cross platform capabilities. Libusb is well document with examples to get started and is provenly reliable interface. This option would be the fastest amongst the three if speed starts becoming an issue. It also may be over kill, there are no serious processor operations we must perform which C/C++ executes must faster. It is also more complex than writing code in Python, but doable if desired. This option has a lot of history in similar data collecting and gathering applications .[5][6]

6) Python with PyUSB: I have selected to use Python with PyUSB that provides a reasonable fast USB access coupled with the ability to be easily portable between Windows and Linux. Using libusb with C/C++ would be the faster and cross compatible, but most of the other code in this project has chosen Python and we want to keep everything in the same language if possible. Designing the product just for windows using WinUSB is too restrictive and option 2 is not going to be considered. Last yea"s Python code was successful and already has code available that can be improved on. No sense in re-inventing the wheel.

C. Handling and Storage of Data

We must handle and store data from user input, sensors/controls input/output, and radio transceiver input/output. We must record user commands and sensor data to the hard drive of the traditional computer. This part describes 3 methodologies of how to accomplish these tasks.The three options being examined in this section are as follows.

1) Options:

- Option 1: Buffer data in memory, process the data, then record the data that has been made suitable for reading to a text file for later retrieval.
- Option 2: Buffer data in memory, process the data, then send the data to a SQL database for storage and later retrieval.

- Option 3: Do not buffer data but instead directly write data to hard drive on computer and have that data accessible through a file descriptor.

2) Goals: The goals in choosing the technology to handle and store data are (1) to deliver data to the visualization/storage algorithms quickly (2) to allow the visualization/storage algorithms to easily access data (3) to save the data in long term storage (4) make the data in long term storage easy to access and understand.

3) Criteria: The criteria that these choices will be based on are: speed, ease of access for visualization, ease of access for long term data, and complexity introduced into software.

4) Comparison Table: Comparison of Options vs criteria (Options are labeled with corresponding tag from section a).

Criteria Comparison Table				
	Speed	Ease of Access for Visualization/Storage	Ease of Access for Long Term Data	Complexity introduced into software
Option 1	Fast	Easy/Fast	User needs to read text files one by one.	Trivial and uses simple techniques like file descriptors to pass data.
Option 2	Medium	Easy/Fast	User would need to make queries to database, but data might be easier to select and retrieve.	Less Trivial as data in memory would have to be accessed and written to a SQL database. Providing more overhead and complexity to the program.
Option 3	Slow	Harder/Slower	Data would be hard to read and need further processing down the road.	This would add the most complexity and code would be more convoluted.

5) Discussion: Buffering data in memory, processing the data, then recording the data that has been made suitable for reading to a text file for later retrieval is a great option. It would be the fastest code with less overhead than the other options for these reasons. One, it would not have to communicate with an outside data base and incur the overhead brought on by an extra API. Two, it is fast to write data to text files and easy to replay the data into the visualization program. If the data is kept in a format that would could be easily read by our visualization system, it could be viewed repeatedly. The text file would be harder for humans to read than queries from and SQL database, but then you would have to have someone with knowledge of the SQL database to format the data. The complexity of this option is the simplest of the three options. We would put data in memory to be accessed by both the logging and visualization systems. Once both have used the data its memory can be recycled. Operations retrieving this data are very fast from memory. Transferring the data to a file would require simple file descriptor read/write access and an appropriate formatting definition to make the logged information readable.

Buffering data in memory, processing the data, then sending the data to a SQL database for storage and later retrieval is another viable option. The plus side of this option would be the ease of representing different forms or data using database tables and entries. These later can be taken and queried from in specific ways to present data that is requested. This would allow better data visualization in the long run, but would require a database access software to be written. This is now out of the scope of our project. It would be a good goal for the next time around. Database API are not hard but do introduce more overhead and complexity. They also introduce delay in storing the data as the database must be contacted to preform input operations. Visualization access would be the same as the previous option. The data would be processed and buffered in memory to be accessed by the visualization software and the SQL data storage software then recycled after both sub processes finish with that piece of data.

If we do not buffer data but instead directly write data to hard drive on computer and have that data accessible through a file descriptor we end up with a messy solution. It would be far more complex and much slower than the other two options. This option still would work. In this case, we would collect the data and write it to a file immediately. This data would be raw and not converted. It would be the responsibility of the other components of the software to read from this file and perform their actions using the raw data. This could introduce conflicts if the program is multi threaded which would need to be resolved with mutex locks. This is more of a brute force option that appeals mainly to getting the data written to the hard drive the fastest. The data would most likely be hard to read in text format.

6) Option 1: I have selected to process and buffer the data in memory and then have the visualization software/logging components access it. The logging component will write the data to a text file. This will allow for easy access to logged data, less complexity, and fast access for other software components. SQL might be able to present the data better, but it is out of the scope of the project and I believe the less overhead the better since our software needs to run fast to communicate all data in a timely fashion.

D. Processing Data to make it Suitable for Visualization and Storage

Data from the rockets sensors and controls needs to be made suitable for visualization/storage on the traditional computer. The raw data from the sensors will be given in binary format and must be converted using formulas provided by the manufacture to a format understood by the visualization/storage algorithms. For example, a temperature might be given as FF3386, but needs to be converted to 120 to be graphed on a temperature gauge. Three possible methods are presented below to accomplish this.

1) Options:

- Option 1: Use an array/table of conversions on the traditional computer to convert data and store data in a buffer to be read by the visualization/storage algorithms.
- Option 2: Use Object Oriented programming to define each data item as an object and make a member function to do the data conversion on that object. Data is then passed directly into that object when it is identified in the data stream.
- Option 3Use data in its raw form as input into the visualization algorithms. The algorithms would have to have a way of knowing the bounds of the data to make an accurate visualization.

2) Goals: The goals in choosing the technology to process data and make it suitable for visualization are (1) to make a programmatically well structured design (2) to provide fast and accurate data conversion (3) to provide data that is easy to visualize (4) to provided data that is easy to store.

3) Criteria: The criteria that these choices will be based on are: speed, complexity of program structure, ease of communicating data between algorithms, and handling data in memory.

4) Comparison Table: Comparison of Options vs criteria (Options are labeled with corresponding tag from section a).

Criteria Comparison Table				
	Speed	Complexity of program structure	Ease of data communication	Handling data in memory
Option 1	Fast/Medium	Simpler but not easy to understand	Easy	Arrays/Buffers
Option 2	Fast/Medium	More complex but uses built in components of language to make a system with flow and elegance	Very Easy	Objects
Option 3	Slow-Fast (Depends on design)	Complexity would be added to the visualization and storage algorithms in order for them to be able to decipher the raw data stream.	Hard	Arrays/Buffers

5) Discussion: Using an array/table of conversions on the traditional computer to convert data and store data in a buffer to be read by the visualization/storage algorithms is a decent option. Classic data structures and operations would be used to organize data conversion formulas. Providing fast easy access of the data. The data would then be buffered in an array or a structure that would provide quick access for the visualization/storage components of our software. The downside to this is that the structures wouldn't be as easy to understand in code and operations might gain complexity as the data needs to be handled carefully. More data control mechanism would have to be put in place to know when to convert, store, and recycle the data to make room for more.

Using Object Oriented programming to define each data item is an excellent choice. An object can have a member function to do the data conversion for its data and store it in its own memory. Data is then passed directly into that object when it is identified in the data stream. This is the most elegant way to approach this problem as it provides clear structure and predefined storage space. Objects would all be predefined in a list have the capability of conversion and buffering data internal to them. This might create more overhead, but defines the data in a realistic way. Visualization and storage components would simply iterate through the list and collect data from the objects internal buffers. All possible languages mentioned in this document have Object oriented capabilities except for C.

Using data in its raw form as input into the visualization algorithms. The algorithms would have to have a way of knowing the bounds of the data to make an accurate visualization. This option would allow the data to be passed quickly to the visualization/storage components of the system, but would cause them to have to both have a way of deciphering the data. This would add quite a bit of overhead to each of those components when they don't need it.

6) Option 2: I am choosing to use Object Oriented programming paradigms so that this component of the software is easy to read, and manipulate. Providing convenient inheritance that allows us to not duplicate code. I believe that this problem lends itself best to this paradigm. The rocket components connected to the system as easily visualized as object with their own attributes. Each sensor or controller will have its own class inheriting from a general class. These sub classes will expand the base class with unique attributes form the components. Every object tis self-contained including any conversion formulas required by the data. Objects are also easily iterated through and accessed.

E. Technology 5

1) *Options:*
2) *Goals:*
3) *Criteria:*
4) *Comparison Table:*
5) *Discussion:*
6) *Selected Option:*

F. Technology 6

1) *Options:*
2) *Goals:*
3) *Criteria:*
4) *Comparison Table:*
5) *Discussion:*
6) *Selected Option:*

G. Technology 7

1) *Options:*
2) *Goals:*
3) *Criteria:*
4) *Comparison Table:*
5) *Discussion:*
6) *Selected Option:*

H. Technology 8

1) *Options:*
2) *Goals:*
3) *Criteria:*
4) *Comparison Table:*
5) *Discussion:*
6) *Selected Option:*

I. Technology 9

1) *Options:*
2) *Goals:*
3) *Criteria:*
4) *Comparison Table:*
5) *Discussion:*
6) *Selected Option:*

J. Technology 10

1) *Options:*
2) *Goals:*
3) *Criteria:*
4) *Comparison Table:*
5) *Discussion:*
6) *Selected Option:*

III. Conclusion

IV. Bibliography

References

[1] BeagleBone.org, *(Thu Oct 20 2016)*, *Beagle Bone Black product information and website*, URL http://beagleboard.org/black

[2] RaspberryPI foundation, *(Accessed Sunday Nov 11 2016)*, *Raspberry PI 3 product information and website*, URL https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[3] Michael Leonard, *(July 18 2013)*, *A comparison between the Raspberry PI and the BBB*, URL http://michaelhleonard.com/raspberry-pi-or-beaglebone-black/

[4] Multiple Wikipedia Users, *(Sun Nov 13 2016)*, *Beable Bone Black wiki page*, URL https://en.wikipedia.org/wiki/Raspberry_Pi/#Specifications

[5] Benchmark Task Performance, *(Accessed Sunday Nov 11 2016)*, *Comparison of C++ and Python over base tests*, URL https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=python3&lang2=gpp

[6] LIBUSB, *(Accessed Sub Nov 11 2016)*, *Documentation on LibUSB API*, URL http://libusb.info/

[7] PyUSB Team, *(Sep 9 2016)*, *PyUSB API and documenation*, URL https://github.com/walac/pyusb/blob/master/docs/tutorial.rst