HyRo
Team 28

Jason Klindtworth | Josh Asher | Layne Nolli

CS461

Fall 2016

Technology Review

Abstract

It is important when begging to design a system to brainstorm options for each sub component. Certain technologies immediately pop into your mind when thinking of a solution to a problem, but it is important to make sure what you thought of was the best solution. This document is a way to compare possible technologies for the individual components of our solution and research which one would be the better choice. We do not want to reinvent the wheel when there is already a solution. We might also find a better solution when we conduct our research. This paper will lead us into our design with a better idea as to how we will accomplish HyRo.c

November 14, 2016

CONTENTS

I. INTRODUCTION

We are desiginging a data driven user interface. Data will be collected from a hyrbrid rocket's sensors and passed through radio waves for us to visualy represent on a screen. The users will be able to interact with the data and send data into the system in the form of rocket commands. To get a grasp on what techonologies or methodologies we present to you our technology reviews. As a team we split the project into 10 sections that we will be responsible for. Each of these sections was cerfully thought over and reviewed with 3 possibly technologies that could be used to build that section. The section and the people responsible are as follows: Generating and Capturing of Data on a Micro Controller (Joshua Asher), Generating and Capturing of Data on a Traditional Computer (Joshua Asher), Handling and Storage of Data (Joshua Asher), Processing Data to make it Suitable for Visualization and Storage(Joshua Asher), the toolkit to generate the UI (Jason Klindtworth), Visualization Tool Kit to Display Data in UI (Jason Klindtworth), The organization of the UI (Jason Klindtworth), Interaction Modes(Layne Nolli), UI Interaction Model(Layne Nolli), Mapping of Data onto Display Model (Layne Nolli).

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. Toolkit to Generate the UI

The tool kit that we use to display the collected data is a very important part of the system. It will allow the user to interact with the data and draw conclusions about what is happening in the rocket. It will need to be clear and concise but also low enough overhead in order to be fast and reliable. The rocket will be traveling very fast and data collection will need to be able to keep up with the changing conditions of the kinematics and pressures that the rocket is experiencing. The user interface will also need to be very modular in order to fit the unique needs of our customization demands and allow for many different types of data to be displayed in unison in a readable, easily understandable way. The three technologies we will be considering for the visualization tool kit are Unity, a game creation platform; VPython, designed for 3d modeling and rendering; and a JavaScript based web UI platform.

Unity is an easy to use highly customizable scripting based software development platform that is very highly used around the world for the creation and deployment of both desktop and mobile applications. One of the major benefits of this platform is the high level of customization that is allowed with things like script interaction, layout, and visualization of data. Another noticeable benefit is the availability of free assets and premade scripts that are used within the system. This will save us development time and cost because a lot of the things we will end up needing are already created and can be used for free within the Unity application. A major downside to using a large application like Unity is the amount of overhead that is required for the software to run. Because we are going to be restricted in the capability of our hardware, we might not have enough resources to run an application this heavy. The other downside is that there is a large learning curve to being able to utilize Unity to its full potential and the time that we make up with the premade scripts and assets we might lose again in learning how to implement those tools successfully.

VPython is a very robust and lightweight platform that makes it easy to display 3d models and manipulate those models using physics and kinematics. There is a lot of existing code in Python that the rocket team used last year which will save on some development time for this project, and python is easy to coordinate with other integrated platforms like C or C++. It is also very nice for modeling graphs and data over time changes which our project is going to need for measuring changing in the rocket data. During my research I found that one of the major downsides to VPython is the lack of documentation between integrating a VPython system into the microcontroller that we are using, so there will be a lot of upfront trial and testing to make sure that we can get the systems to work together. Another downside is that when talking to other members of our team (the ECE group that is working on the hardware) they have little to no experience with VPython so it would be a challenge to get it working together with their systems they are designing.

The third option available to us is a JavaScript web based application that can be used to display all of the data that we need. This is probably the easiest option to implement and also easy for getting data input from all of the hardware sensors we are going to use. The downside to a web app is security and reliability. Security is not really a problem for our system in general, and it would be relatively simple to add some level of encryption to the data because it will technically be sent across an internet protocol. The other problem is reliability. If we are using a web based application it will rely on the web service being up and running at the time of deployment. This is not usually an issue as these systems are generally always up, but it is one more concern that we would have to be aware of.

After doing quite a bit of research on this topic and weighing the pros and cons of all the options, we determined that VPython would be the best platform to use for our needs. The amount of preexisting code from previous teams will be very helpful to our needs and the low overhead over something like Unity will help with data transfer speeds and keep our read times in the projected requirements for the project. Additionally we feel like the downsides of this platform will be manageable

.

F. Visualization Tool Kit to Display Data in UI

There is a near infinite amount of ways to display data. We could use graphs, charts, tables, sliders, gifs, images, the list goes on and one. There are also a few different underlying technologies that can be used to organize the UI. This can be done with frames, objects, pre-rendered images with changing data sets, or a number of other systems in place. The three options we will be discussing to organize the UI for our project are Semantic UI, HTML/CSS, and the IPython with Project Jupyter.

Semantic UI is a pre-packaged program in place to design and implement project ideas and user interfaces for data driven UI systems. It works with multiple different languages to combine elements of a user interface into a simple straightforward design. The benefits of this system include an easy to navigate UI that has a consistent feel and layout. It would also be easy to manage different projects and different layouts in order to test different layouts to see one we like without a lot of extra testing or coding. However, the system seems fairly difficult to get up and running and managing a large project like this with multiple different sub groups might be challenging.

HTML/CSS would be a local web layout that we could customize to look however we want, we could either use preexisting CSS packages or create our own fairly easily. This type of system would give us the most control over layout style and we could tailor it to look exactly how we need for the project. Another benefit is that it is pretty easy to implement HTML/CSS with other languages that we might use like C++, JavaScript, or Python. The downside of this system would be that in order to make any changes to the layout once we made it would require a large overhaul and re-coding the majority of the system.

IPython is a way to use Python code to create an interactive system which can be used as a customizable user interface that would serve our needs for this project. The main benefit of this system would be that it interacts well with Python and VPython natively and would save a lot of time in trying to get cross systems to work seamlessly in the final integrated system. The problem with IPython is that there is limited support documentation, there is not a whole lot of microcontroller support, and some of the team members have never used it, so we would lose development time learning how to implement the system to make sure it is compatible with the other software systems or microcontroller hardware.

For the final implementation for this project we are going to go with an HTML/CSS layout for our user interface. The capabilities of higher customization and layout control are very desirable for this project and will benefit the end system. In order to get around the drawbacks of an HTML/CSS system we will do smaller mockups of the final project before deciding on the final design. That way we can test a few different types of layouts before committing additional resources that we might not end up using.

G. The organization of the UI

The way that the user interface is organized is an important consideration to the usability of the system being designed. There are many ways that a UI can be organized for ease of use, functionality, or density of information being displayed. There is a fine line between these three styles and we need to carefully consider the pros and cons of each and how they will fit into the project goals and requirements for the overall system.

The first type of UI layout we considered is one designed for ease of use. This is a way to make the UI very user friendly so they know what exactly is going on without much clutter or over exposure to large quantities of data. This type will have simple graphs and buttons with minimal data being displayed for the user. The majority of the data will be archived into a file structure and stored in logs to be viewed later instead of being on the screen. This allows for a less cluttered interface for the user. The downside is that the user might be missing out on a specific type of data they want at a given time, or they might need to issue a command or task to the rocket mid-flight that a simple UI would prevent access to that command, in the case of an emergency or unplanned event for example.

The second type of UI is one designed for high functionality. This is a more complex UI with additional features such as scripts or commands that would send instructions to the rocket that may not ever be needed, but are included in the UI in the rare case that something goes wrong and they are needed after all. It will show about the same amount of data as the first type, but will have many more options available for executing commands. The downsides to this type of UI are that we would be creating and implementing code for obstacles and situations that may never arise, we might use a lot of time developing something that would never be used. The upside, obviously, would be that in the rare case we DID need that functionality, we would have it.

The final type of UI we discussed is a data dense UI which shows much more data on the interface then the other two methods. We would have almost all of the gathered data in very dense graphs and tables that would update in real time while the rocket is in flight. This would leave less room for functionality commands, but at a glance we would have more visibility to what is happening to the rocket during flight. The benefit of this type of system would be that the user would be able to know exactly what is happening at any given time without having to rely on the logs being analyzed after the flight. A major downside of this design is the cluttering of the data being displayed. Having access to so much data all at once might be distracting to the user and would not look clean or streamlined.

None of these systems are going to work specifically for our needs, so we are designing a new type that is a combination of these existing UI types. We want the system to have high functionality while maintaining good data density. We will still be outputting the data to a log for future analysis, but most of the gathered data will be available on the screen in real time during launch. The system does not need to be super simple as the layout is for gathering complex data, but we want the users to be able to navigate the system without needing extensive training with the system beforehand.

H. Interaction Modes

Interaction modes are important in order to create a natural and flowing user-experience when attempting to derive meaning from the data displayed. Without a good interaction method, data is just miles of spreadsheets and characters. Interaction modes allow the user to get to their desired viewports and data processing methods with minimal effort or headache. Our first option for interaction methods is the gold standard, mouse and keyboard. This interaction mode is the one that requires minimal user training, as most users are already more than well versed with it. Because of its prevalence in today's world of computers, the mouse and keyboard is already second nature for 99.9 percent of potential users of our system. This is a

major advantage for us because all we have to do is create a control sceme that compliments this input method in an intuitive way and users will spend minimal time having to learn our UI control scheme. Second interaction mode would be a touch screen on a tablet or smartphone. This method is a very powerful way of navigating the UI. Mobile touchscreen interaction is very efficient and very intuitive. whats more intuitive that using your own fingers to navigate a UI? While it can be argued that this mode is very intuitive, the issue with this interaction mode is that our project requires real time interaction with components that just dont mesh well with a mobile device. So while it may be nice as a side project, or secondary method to view data post-launch, it does not suit our needs for data collection and processing (and viewing) within the scope of our project.

Finally, UI interaction could be supported by way of virtual reality. This method is obviously beyond the scope of our project, designing a virtual reality program for a user to step into and use various "VR wand" inputs to move about a virtual space and interact with data is a rather inefficient and complex process. The concept sounds very cool from a presentation standpoint. But when looking at this idea relative to the requirements of our project, it is just not feasible.

Our goals for interaction modes in the scope of this problem are to create a system that is intuitive, simple, and powerful. Taking a closer look at these goals; We want to create a system that is not going to take a user a long time to become capable of using it effectively. We want to create a system that involves as few "hurdles" as possible, meaning a user can get to their desired data visualization or graphic with minimal steps. And finally we want a system that offers the full selection of data and processed data visualizations that the user may desire, there should be no case that the user wishes they could see but it is not supported by the system. For this portion of the solution, mouse and keyboard will be the best input method to be paired with the rest of the system.

I. UI Interaction Model

The UI interaction model, as it pertains to our project, is the marriage of our UI capabilities and our input mode. As discussed prior, our best known interaction mode is via the mouse and keyboard. The UI interaction model itself is a set of decisions that will define how our UI will look and feel for any user trying to make sense of the data collected during our project, it defines what the user can and can't do when interacting with our system.

One option is to make use of menus, in a tree-like structure, so that a user can navigate through a series of menus that present various data manipulation options as the user progresses, eventually resulting in the desired data being processed and presented on screen.

Another option is a more graphical approach, displaying data on screen in graphical format right off the bat then letting the user change x or y axis with available data metrics to create graphical representations of the data they are interested in real time. For example a user could want to see the flow rate versus time on a graph, but after viewing this data they may want to see rocket velocity versus time and would interact with the y axis to produce this result.

Finally we could implement a system of preset data viewing options. So the user would do less navigation or selection of how they personally would like to see the data, and instead they would just be viewing or choosing from a set of available options. This is a much less hands-on approach to presenting useful information to the user, but accomplishes the task based on what we know the user will want to see.

Within the scope of our project, and because of the incredibly complex formulas that will need to be applied to the data. It makes the most sense for us to go with the last option. We will decide what the user gets to see based on communicating with the user during development of our project and we will present only that pertinent information on the UI. Giving the users total freedom to manipulate data within our system is just not a practical approach, it will be clunky to use and incredibly difficult to create bug-free. A nice dashboard of known data outputs will be all our user will need.

J. Mapping of Data onto Display Model

The method for mapping the data onto the display model is a critical component for making sense of our projects data output and improving the project in future iterations. There is really only two ways for us to map data into our display model, and both are graphical. It is discussed in this document that we will be using an HTML customized web layout to represent all our UI elements, and beyond that, all data that will be displayed on our UI will be in "graphical" or "gauge" format. An example of the former would be a generic x y plot of velocity over time. An example of the later would be like the engine temp guage on your car dashboard. There is no real comparison required here because both methods will be used and applied to the two different visual data representations desired by our user. The only other method would be to view raw data in table format and that is already fully available to our user, our goal is to make that data more easily understood and/or manipulated.

III. Conclusion

Through are careful consideration of each section we have concluded that we will use these techgnologies for each of our sections. This may change as the project progresses, but this is what we have decided so far. Generating and Capturing of

Data on a Micro Controller: Beagle Bone Black with Pyhton, Generating and Capturing of Data on a Traditional Computer: Pyhton with PyUSB, Handling and Storage of Data: Buffer and record to text file, Processing Data to make it Suitable for Visualization and Storage: Object Oriented approach, the toolkit to generate the UI: VPython, Visualization Tool Kit to Display Data in UI: HTML/CSS, The organization of the UI: Combination of research items, Interaction Modes: Mouse and Keyboard, UI Interaction Model: Dashboard of Kown Outputs, Mapping of Data onto Display Model: Graphs and Gauges. We will go into our design document with these changes unless a bump in the road occurs and we need to alter our plan.

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