CS Capstone Technology Review

November 20, 2017

30k Rocket Spaceport America

Prepared for

Mechanical Engineering, Oregon State University

Nancy Squires ________ *Signature* ________ *Date*

Prepared by

Group41

30k CS Avionics

Joshua Novak ________ *Signature* ________ *Date*

Allison Sladek ________ *Signature* ________ *Date*

Levi Willmeth ________ *Signature* ________ *Date*

Abstract

This document describes three technology choices that our team needs to make to complete our project, with three proposed options for each choice.

Contents

1 Graphical User Interface Language

1.1 Overview

The GUI will be in the form of either a web-hosted or local app that connects to an intranet database. It will display graphs of telemetry data, either live or recorded.

1.2 Criteria

The criteria for evaluation is based on whether there are pre-existing libraries that can accomplish the task, the ease with which it can be implemented alongside a database and in a web-hosted app, whether it runs server or client side, and how flexible it will be for adjusting the visual design of the application. Pre-existing libraries and ease of implementation are given a priority because any of the three should be able to be used to accomplish the task and efficiency of the code is not of significant importance since it will all be run on client computers such as laptops that should be more than well-equipped to handle the task, though performance issues may become significant if the script needs to run on the computer running the server.

1.3 Potential Choices

1.3.1 Python

Members of the team have used Python to create GUIs before, so the team is already aware of several libraries to use with this. Several are options are described in a blog on python libraries named 5 Python Libraries for Creating Interactive Plots[1] which also has links to the relevant libraries. They appear to be fairly easy to use and likely will fit our purposes, with some example plots being visually quite similar to our own end goal. In particular, HoloViews features a map similar to one we will need to generate from GPS data. Several of the libraries mentioned in the source have functions for outputting to HTML, so we should be able to use them to create our graphs. However, it is unlikely that these libraries will be able to dynamically alter the display as new data is made available since they generally output to HTML files, though this should be possible by overwriting HTML files when a library does not have the functionality to dynamically update the graphs and having the site automatically refresh some of its contents. In general, Python libraries will run on the server computer rather than the client, which may cause performance problems.

1.3.2 JavaScript

There are many libraries for JavaScript that the team should be able to use to generate or manipulate graphical content, including the CanvasJS[2]. This library accomplishes most of our goals for graphing data, explicitly describes methods for live updates to graphs, and has methods for generating graphs from csv files. This library is extremely suitable to our tasks. JavaScript generally runs primarily on the client side, which is true of this library, so performance should not be an issue. The only potential issue will be the map, for which either another library can be used, or the map image can simply be edited using standard JavaScript methods to place red dots or other markers over the map at the correct locations.

1.3.3 R

R is specialized for mathematical programming, which suits our needs. There are libraries for R which generate graphs that fit the desired output, such as Plotly[3]. These libraries will either be able to perform live updates (with the implementation of a Plotly server), or overwrite the currently existing HTML file to mimic this. Visually, the plots are very functional, and they include graphs that resemble the GPS map that needs to be created. However, it will need to run on the server computer rather than on the client computer, which may cause performance issues.

1.4 Compare

R (and the related libraries) has all of the graphs we will need and is specialized for much of this task, while also being very easy to use and code in. However, it will need to run on the server, and therefore may introduce performance issues. Python is not as specialized for this task or as easy to use, but is more flexible, and has a larger number of related libraries. It will also likely perform better. However, it also will run server side, and therefore may introduce performance issues. JavaScript has libraries that explicitly describe how to create live updates to dynamic graphs. It also runs client side, giving it the best performance.

1.5 Conclusion

JavaScript is likely the best candidate. It will be able dynamically generate our graphs, and runs client side to deal with performance issues. As it is the only option that accomplishes all of these goals, it is what will be used to generate the GUI.

2 Avionics Code - Language

2.1 Overview

A portion of the Avionics code for the rocket and payload will be written by the CS team, with some of the code also being written by the ECE team in order to satisfy the requirements of their Senior Capstone. This code will need to run on a Blackberry Pi.

2.2 Criteria

The criteria for selecting the language of the Avionics code is based on three major factors, the efficiency of the language, the compatibility of the language with the devices that are likely to be used, and the suitability/usability of the language for accomplishing the required tasks. As a minimum requirement to be used at all, the code must be compatible with the Blackberry Pi, narrowing the range for selection. So far, the only device that is known to be used in the rocket is BigRedBee's BeeLine GPS[4], which will not heavily interact with the avionics code and does not appear to have a related library. Looking to previous years, most sensors have C libraries for interacting with them, so languages that are compatible with C functions or can call C programs are more valuable. Therefore the secondary criteria breaks down to the language's compatibility with C.

2.3 Potential Choices

2.3.1 C

C easily satisfies all three criteria. As a low-level language, C is highly efficient. This makes it excellent for criteria one. It is obviously compatible with itself, satisfying the second criteria. As for suitability/usability for the task, C is the standard language for embedded code such as avionics code. Issues may be encountered due to the difficulty of coding in C compared to higher level languages but these should be able to be addressed. When creating the test suite, a higher level language would normally be preferred due to the ease of use and the compatible libraries, but the test should still be able to be written in C.

2.3.2 C++

C++ is a high-level object-oriented language, making it inefficient in many cases. While C functions and avoiding the use of objects can address this problem, additional inefficiencies will be introduced by using C++. C functions can be used in C++, so there is unlikely to be any compatibility issues. All of the necessary code should be able to be written in C++ without significant issue, and it's status as a high-level language will make it easier to use. Test suites will be easier to create and organize with the option to use objects.

2.3.3 Python

Python is a high-level object-oriented language with particularly issues with efficiency. For this reason, it largely fails criteria one, and may cause performance issues. Python can call C programs and collect their outputs, so it should be able to satisfy criteria two with minimal effort. Finally, it should be very easy to write code and tests in Python, and there are a wealth of libraries that exist to bridge any gaps we may encounter. However, such libraries are likely to be unnecessary. Test cases will be relatively easy to write in python.

2.4 Compare

None of the three languages will have significant compatibility issues, though Python may encounter minor ones. Python is by far the most inefficient of the three languages, and only brings minor benefits in usability and suitability. C++ is significantly more efficient and has similar benefits to Python in usability and suitability, as we are unlikely to need to use third-party libraries that may exist for Python. Finally, C is vastly more efficient than either of the other options, and while it may be somewhat more difficult to use, it is not a significant downgrade in this case.

2.5 Conclusion

C will be used for the avionics code due to the greater efficiency of the language with only minimal losses in usability and suitability. Tests may be written using C++, as the two languages are compatible with one another, but the avionics code itself will be written solely in C.

3 Avionics Code - Testing Methods

3.1 Overview

The client has requested that the CS team create a thorough suite of tests for the Avionics code. This is due to the many failed recoveries of rockets in the past, which may have been due to poorly written/tested code.

3.2 Criteria

The criteria for evaluation is whether the testing method will allow us to test against likely errors that we may encounter from the sensors. This will vary based on the sensor in question, but of particular importance is a sensor throwing a value that is outside of expectations, not an acceptable value, or failing to send a value at all. Of lesser but still significant importance is creating tests that ensure that the return values for a flight without sensor errors are accurate.

3.3 Potential Choices

3.3.1 Real Data

Testing against real data includes test launches/flights, testing against the sensors at rest, and testing against data recorded from previous flights, such as those carried out by earlier years or data found online. There are several limiting factors on these tests. Firstly, test launches/flights will not occur often, making their utility extremely limited. Also, we will most likely want our avionics code to be completely before any test launch. Second, previous flights and flights found online may have their data fixed to remove outliers or not contain errors. This makes this method less useful for satisfying our core criteria. However, they remain very useful for testing against the secondary criteria, as we should be able to test against flights/launches that reflect our own. Finally, tests against the sensors at rest will only be able to allow us to test within an extremely narrow range of possibilities that do not reflect the likely outcomes during a launch.

3.3.2 Simulated Launch w/ Robustness

Testing against a simulation tool or a collection of simulated data has a variety of advantages. It satisfies the second criteria very well, by testing against what should be the outputs of a typical launch or a launch within certain parameters. Robustness testing can be inserted in manner similar to how robustness testing is added to model-based testing, as outlined in the paper Model-based robustness testing for avionics-embedded software[5]. The article describes a manner of adding random changes to the outputs of different test objects representing sensors during a simulated launch. This ensures that the avionics code can handle incorrect outputs or failings in the hardware. This robustness testing would include randomly having the sensor throw a value outside of the expected bound, a value that does not fit it's standard form, or having it shut down and turn on at random intervals. By setting this to occur randomly at some varying rate, the robustness of the suite can be dramatically increased, satisfying the second criteria.

3.3.3 Unit Testing

Unit Testing will ensure that the basic functionality of the code is there. It can be used to ensure that for very specific inputs the correct outputs are reached, and that every line of code and logic is checked. This helps satisfy the first and second criteria, and ensures the code is functional. It is also fairly straightforward to implement.

3.4 Compare

All three have particular advantages and disadvantages. Testing against real data is perhaps the best way to satisfy the second criteria, as it is definitely a test against a likely launch. However, it does not satisfy the first criteria to any significant degree, if at all. Unit Testing will assist in satisfying both criteria, but not as well as Simulated Launch with Robustness. However, it is much easier to implement than either other method.

3.5 Conclusion

Simulated Launch with Robustness is the best was to ensure that both criteria are satisfied, and therefore will be given the highest priority to complete. Unit Testing, due to the ease of implementing it as well as its significant returns, will be used alongside it. Testing against real data will be given the lowest priority, as it only satisfies the second criteria, and not significantly better than a Simulated Launch.

REFERENCES

[1] M. Bierly, "5 python libraries for creating interactive plots," October 2016. [Online]. Available: https://blog.modeanalytics.com/python-interactive-plot-libraries/

[2] "Beautiful html5 javascript charts." [Online]. Available: https://canvasjs.com/

[3] "R graphing library — plotly." [Online]. Available: https://plot.ly/r/

[4] "Beeline gps." [Online]. Available: http://www.bigredbee.com/BeeLineGPS.htm

[5] W. S. Yang Shunkunab, Liu Binab, "Model-based robustness testing for avionics-embedded software," June 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1000936113001179