PYTHON CONTROL PACKAGE FOR LIQUID ROCKET ENGINE TEST STAND

DEVON BURSON

Oregon State University

Propulsion Laboratory

School of Mechanical, Industrial, and Manufacturing Engineering

bursond@oregonstate.edu

PyIGN Package Final Report

June 12th, 2019

Contents

1 Introduction 5

1.1 Motivation . . . 5

1.2 System Architecture . . . 6

2 Methodology 7

2.1 Systems Analysis . . . 7

3 Implementation 10

3.1 Software Structure . . . 10

3.2 Python/LabVIEW API . . . 10

3.2.1 Functions . . . 12

4 Results 13

4.1 Control Panel GUI . . . 13

4.2 PyIGN Output . . . 13

5 Conclusion and Future Work 14

5.1 Package Overview . . . 14

6 Appendix 15

List of Tables

1 Valve Class . . . 7
2 Pressure Transducer Class . . . 7
3 Thermocouple Class . . . 8
4 Load Cell Class . . . 8
5 Ignitor Class . . . 8
6 Nanny/Abort Class . . . 8
7 GO/NOGO Class . . . 8

List of Figures

1 Liquid Rocket Engine Test Stand 5
2 Piping and Instrumentation Diagram 6
3 System Work Flow 10
4 Python/LabVIEW Function API 11
5 Python/LabVIEW Class Wrapper 11
6 Check Limits Structure 12
7 Fuel Control GUI 13
8 Command Line Output 14

Acronym Dictionary

ABV Actuated Ball Valve. 6, 7

DAQ Data Acquisition. 5, 6, 14

LC Load Cell. 6, 8

LOX Liquid Oxygen. 6, 9

LPS Liquid Propulsion System. 5

LRETS Liquid Rocket Engine Test Stand. 5, 6, 14

LS Limit Switch. 6

NI National Instruments. 6, 10

OSU Oregon State University. 5

PID Piping and Instrumentation Diagram. 6

PT Pressure Transducer. 6–8

PyIGN Python IGNITE. 5–7, 10–14

TC Thermocouple. 6, 8

1 Introduction

1.1 Motivation

To develop a proven reliable liquid rocket propulsion system, a robust testing program that verifies and validates the performance of the system in different operating conditions is required [[6],[8],[3]]. Through extensive testing and development, engineers can use gathered data to accurately model a Liquid Propulsion System (LPS)'s performance to improve control and improve the system design [8]. For these reasons, safely and accurately gathering sensor data during testing procedures is crucial to an LPS's development[4]. Oregon State University (OSU) is currently developing the infrastructure required to reliably test and safely develop LPS's. OSU's Liquid Rocket Engine Test Stand (LRETS), located in the Propulsion Laboratory, is illustrated in Figure 1.

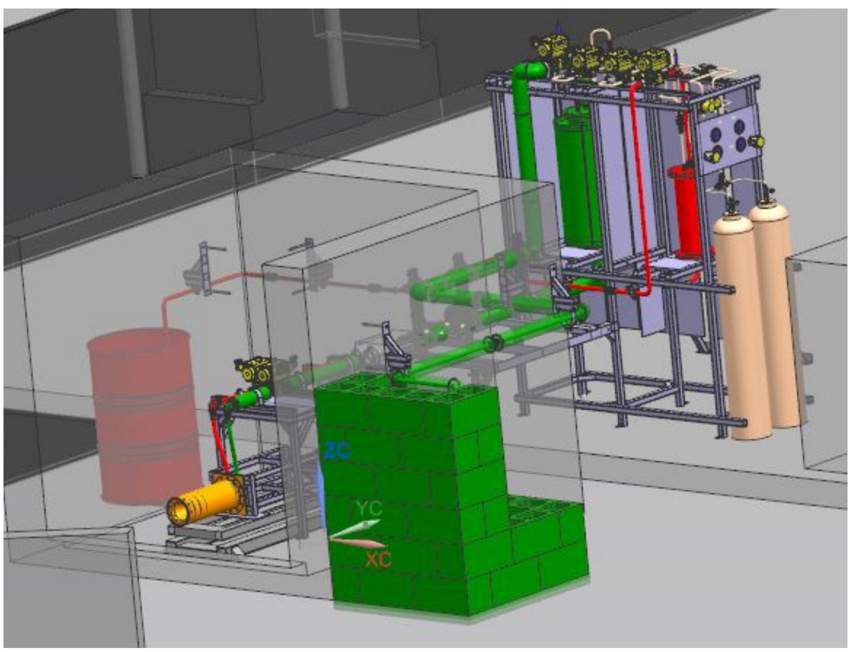


Figure 1: Liquid Rocket Engine Test Stand

A comprehensive testing programs system architecture can be broken into three main sub-categories. Systems include, testing facility hardware, testing Data Acquisition (DAQ) and control, and testing procedure [[8],[9],[5],[2]]. For the purpose of this design report, the testing facility hardware and testing procedure are ignored. However, it is important to note that the testing DAQ and control system architecture directly affect the design of both the testing facility hardware and testing procedure [7]. All three systems are closely related, but for the purpose of this report, the software package Python IGNITE (PyIGN) is primarily discussed.

1.2 System Architecture

Illustrated in Figure 2, the LRETS's Piping and Instrumentation Diagram (PID). The system consists of twelve Liquid Oxygen (LOX) compatible pneumatically Actuated Ball Valve (ABV)'s, eight Pressure Transducer (PT), sixteen Thermocouple (TC), three Load Cell (LC), and twelve Limit Switch (LS). During operation, all thirty nine sensors continuously output data to a National Instruments (NI) cDAQ Chassis system. While the DAQ system collects and records sensor inputs, it simultaneously sorts and formats data from a select number of sensor. The formatted data is then output to the PyIGN package. The python software processes the input data and determines the LRETS's valve states.

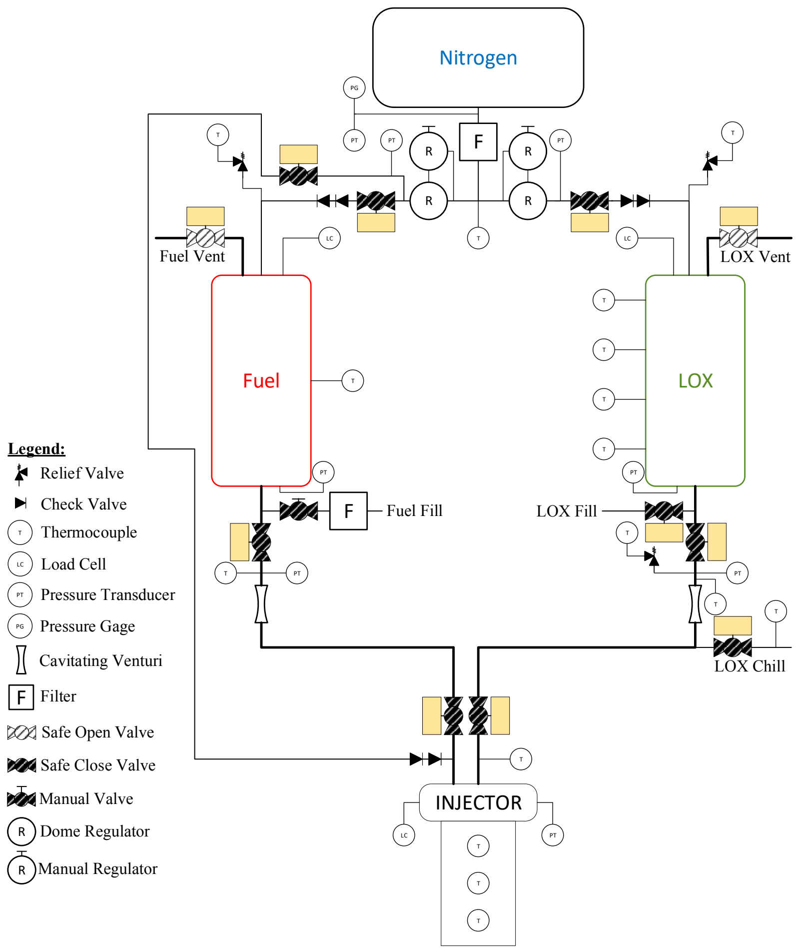


Figure 2: Piping and Instrumentation Diagram

2 Methodology

2.1 Systems Analysis

The development framework used to design the PyIGN software package focuses on intent specification. As defined in the NASA General Safety Program Requirements, intent specifications are based on research in human problem solving and on basic principles of system theory and system engineering[[6],[8]]. An intent specification differs from a standard system engineering specification primarily by prioritizes system-level requirements and design constraints[[6],[9]]. By focusing on intent information, prior design rational and assumptions are incorporated into the design of the safety software. During system testing, if the original assumptions are proven wrong, a new safety design analysis is triggered for the system-level task, without requiring a complete system redesign [6]. This is possible due to the development framework, which facilitates a modular package structure. For this project, each software module is developed and tested individually, and the final integrated version is tested once each module functions as designed. This method ensures that packages have fewer interdependence's, and simplifies the process of tracking errors once the system is integrated into the test stand.

[8]

Table 1: Valve Class

Class Name	Type	Valve ID
ValveState	Valve - a	ABV-PR-110
ValveState	Valve - b	ABV-PR-120
ValveState	Valve - c	ABV-OX-210
ValveState	Valve - d	ABV-FU-310
ValveState	Valve - e	ABV-OX-220
ValveState	Valve - f	ABV-FU-320
ValveState	Valve - g	ABV-OX-230
ValveState	Valve - h	ABV-FU-330
ValveState	Valve - i	ABV-OX-240
ValveState	Valve - j	ABV-FU-340
ValveState	Valve - k	ABV-OX-250

Table 2: Pressure Transducer Class

Class Name	Type	Pressure Transducer ID
PTLimits	Pressure Transducer - a	PT-OX-110
PTLimits	Pressure Transducer - b	PT-FU-120
PTLimits	Pressure Transducer - c	PT-OX-210
PTLimits	Pressure Transducer - d	PT-FU-310
PTLimits	Pressure Transducer - e	PT-OX-220
PTLimits	Pressure Transducer - f	PT-FU-320

Table 2 – continued from previous page

Event	**Action**	**Definition**
PTLimits	Pressure Transducer - g	PT-CC-410

Table 3: Thermocouple Class

Class Name	**Type**	**Thermocouple ID**
TCLimits	Thermocouple - a	TC-OX-210
TCLimits	Thermocouple - b	TC-FU-310
TCLimits	Thermocouple - c	TC-OX-220
TCLimits	Thermocouple - d	TC-OX-230
TCLimits	Thermocouple - e	TC-OX-240
TCLimits	Thermocouple - f	TC-OX-250
TCLimits	Thermocouple - g	TC-FU-320
TCLimits	Thermocouple - h	TC-OX-260
TCLimits	Thermocouple - i	TC-OX-270
TCLimits	Thermocouple - j	TC-CC-410
TCLimits	Thermocouple - k	TC-CC-420
TCLimits	Thermocouple - l	TC-CC-430

Table 4: Load Cell Class

Class Name	**Type**	**Load Cell ID**
LCLimits	Load Cell - a	LC-OX-210
LCLimits	Load Cell - b	LC-FU-310
LCLimits	Load Cell - c	LC-CC-410

Table 5: Ignitor Class

Class Name	**Type**	**Ignitor ID**
IgnitorState	Ignitor State	CC-Ignitor

Table 6: Nanny/Abort Class

Class Name	**Type**	**System Nanny/Abort**
AbortState	Abort State	SS-Abortr
NannyState	Nanny State	SS-Nanny

Table 7: GO/NOGO Class

Class Name	**Type**	**System GO/NOGO**
GoState	GO/NOGO - a	Control Panel

Table 7 – continued from previous page

Class Name	Type	System GO/NOGO
GoState	GO/NOGO - b	LOX Panel
GoState	GO/NOGO - c	Fuel Panel

3 Implementation

3.1 Software Structure

Figure 3 represents the software work flow during operation. Blue components represent sensors or micro-controllers, grey represents hardware valves or actuators, and red represents the PyIGN package. Similar NI cDAQ systems are used in industry setting [5].

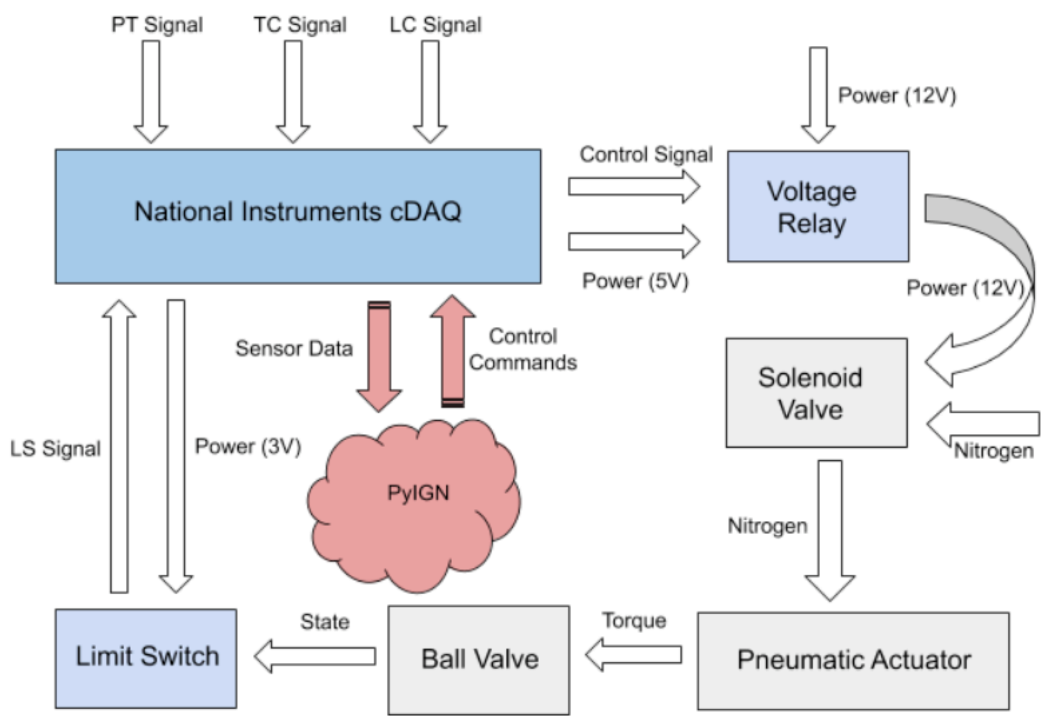
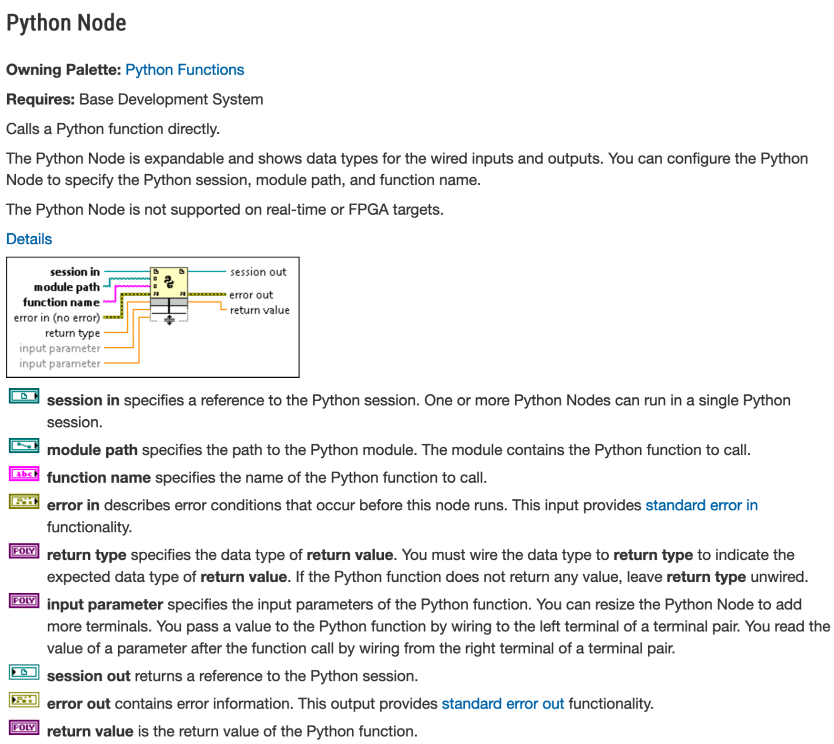


Figure 3: System Work Flow

3.2 Python/LabVIEW API

During operation the PyIGN package must continuously communicate with the NI cDAQ Chassis to ensure a system "ABORT" is not trip. The function used for two way communication between systems and and its corresponding API is displayed below in Figure 4. A Python specific node is used as a function wrapper by the LabVIEW software. This allows a .VI file to access and call the Python module. This single method of communication between the systems affects the flow of information between the systems by limiting or bottleneck's the information flow. In an attempt to mitigate this issue, nested functions are used to reduce the amount of time that the python wrapper node is called. This method allows a multiple system states to be evaluated from a single python wrapper call. Figure 6 represents the nesting technique used in the PyIGN pack.


Python Node

Owning Palette: Python Functions

Requires: Base Development System

Calls a Python function directly.

The Python Node is expandable and shows data types for the wired inputs and outputs. You can configure the Python Node to specify the Python session, module path, and function name.

The Python Node is not supported on real-time or FPGA targets.

Details

session in specifies a reference to the Python session. One or more Python Nodes can run in a single Python session.

module path specifies the path to the Python module. The module contains the Python function to call.

function name specifies the name of the Python function to call.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

return type specifies the data type of **return value**. You must wire the data type to **return type** to indicate the expected data type of **return value**. If the Python function does not return any value, leave **return type** unwired.

input parameter specifies the input parameters of the Python function. You can resize the Python Node to add more terminals. You pass a value to the Python function by wiring to the left terminal of a terminal pair. You read the value of a parameter after the function call by wiring from the right terminal of a terminal pair.

session out returns a reference to the Python session.

error out contains error information. This output provides standard error out functionality.

return value is the return value of the Python function.

Figure 4: Python/LabVIEW Function API

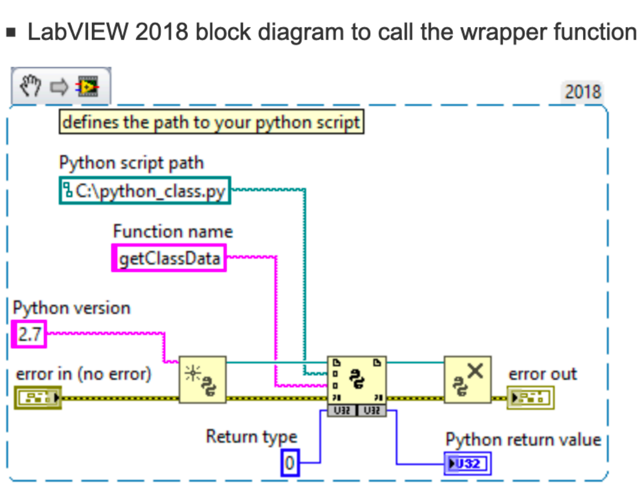


Figure 5: Python/LabVIEW Class Wrapper

A second software work flow issue stems from the use of the python class wrapper with the python function wrapper. When a python class instances is creating, data is sent between systems via the class wrapper. By passing class data between the LabVIEW and Python systems with the function wrapper, as well as calling the function wrapper to maintain systems states, traffic doubles which slowing the work flow. Below, Figure 5 illustrates the method to initialize a python class instance, but as mentioned above, communication issues can arise. Further testing is required to determine which method of communication between system is the most effective.

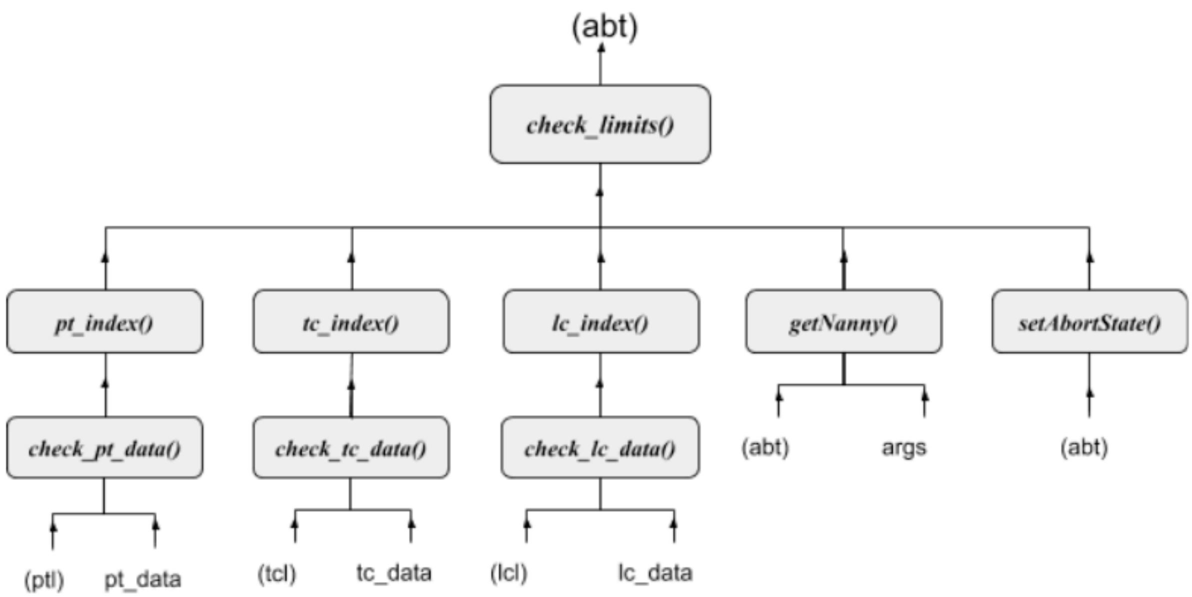


Figure 6: Check Limits Structure

3.2.1 Functions

Docstring commented function are located within the PyIGN package function file. PEP 484

4 Results

4.1 Control Panel GUI

In Figure 7 the results of initial system tests are displayed. During preliminary testing, data was read from a single sensor into the PyIGN package. Event though testing was limited when compared to command line tests that were preformed on the package, the results from initial testing looked promising.

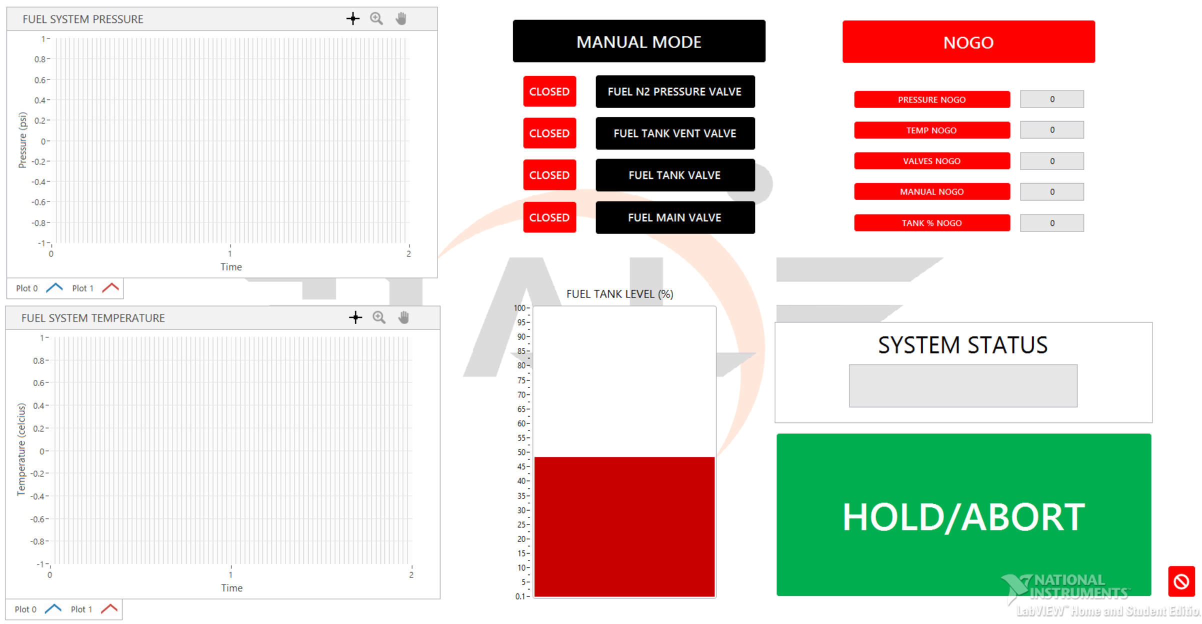
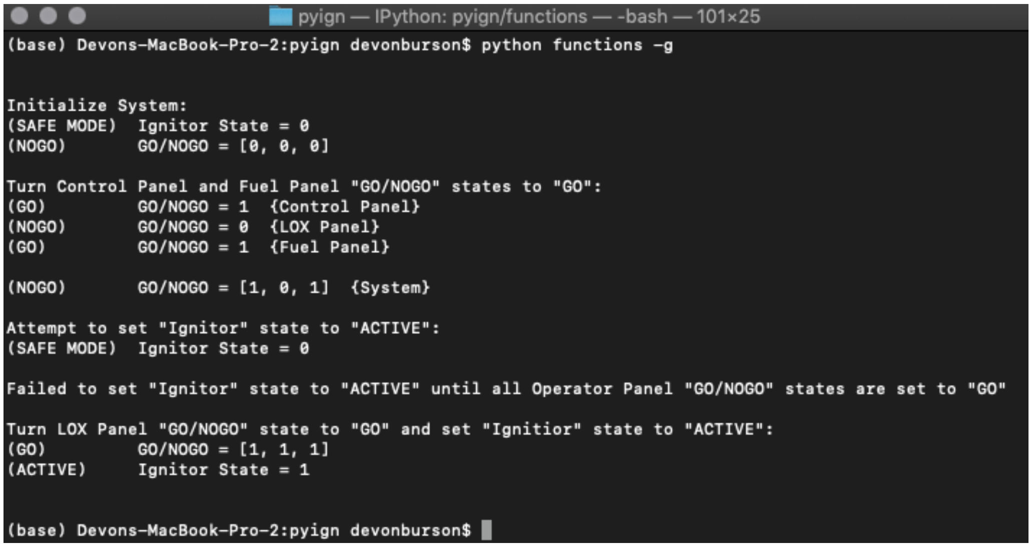


Figure 7: Fuel Control GUI

4.2 PyIGN Output

Results from command line testing is displayed in Figure 8. To decrease the chance of system errors during operation, a relatively extensive testing suite was created for the PyIGN package. The final test coverage value is 92 %, and is displayed on the README.md file.



```
pyign — IPython: pyign/functions — -bash — 101×25
(base) Devons-MacBook-Pro-2:pyign devonburson$ python functions -g


Initialize System:
(SAFE MODE)  Ignitor State = 0
(NOGO)       GO/NOGO = [0, 0, 0]

Turn Control Panel and Fuel Panel "GO/NOGO" states to "GO":
(GO)         GO/NOGO = 1  {Control Panel}
(NOGO)       GO/NOGO = 0  {LOX Panel}
(GO)         GO/NOGO = 1  {Fuel Panel}

(NOGO)       GO/NOGO = [1, 0, 1]  {System}

Attempt to set "Ignitor" state to "ACTIVE":
(SAFE MODE)  Ignitor State = 0

Failed to set "Ignitor" state to "ACTIVE" until all Operator Panel "GO/NOGO" states are set to "GO"

Turn LOX Panel "GO/NOGO" state to "GO" and set "Ignitior" state to "ACTIVE":
(GO)         GO/NOGO = [1, 1, 1]
(ACTIVE)     Ignitor State = 1


(base) Devons-MacBook-Pro-2:pyign devonburson$
```

Figure 8: Command Line Output

5 Conclusion and Future Work

5.1 Package Overview

A Python tool was developed and can interface with a LabVIEW DAQ system. The package, PyIGN is used to monitor sensor input data on a LRETS and is able to be installed from pip. The EXAMPLE.rst file in the docs folder of the package allows the user to run a limited number of the core.py package functions. When testing, the package can automatically "ABORT" during a test sequence if sensor data exceeds the preset system bounds. Further testing will continue over the summer term, once the DAQ and control electronics are installed and PyIGN is integrated into the system[1].

6 APPENDIX

References

[1] Burson E. D. *1.0.3*. 2019. DOI: 10.5281/zenodo.3244879. URL: https://github.com/devonburson/PyIGN.

[2] *Hardware-in-the-Loop:The Technology for Testing Electronic Controls in Vehicle Engineering*. URL: https://www.dspace.com/files/pdf1/dspace-paper_hil_overview_waeltermann_e_160405.pdf.

[3] *Reliability as an Independent Variable Applied to Liquid Rocket Engine Test Plans*. URL: https://pdfs.semanticscholar.org/9e89/d5309c008f3cd298233c56f125a61a83c6d3.pdf.

[4] *Reliability as an Independent Variable Applied to Liquid Rocket Engine Test Plans*. URL: https://pdfs.semanticscholar.org/9e89/d5309c008f3cd298233c56f125a61a83c6d3.pdf.

[5] *ROCKET COMBUSTOR EXPERIMENTS AND ANALYSES*. URL: https://tfaws.nasa.gov/TFAWS03/Data/Propulsion%5C%20Session/Anderson.pdf.

[6] *Safety-Driven Model-Based System Engineering Methodology Part I: Methodology Description*. URL: http://sunnyday.mit.edu/JPL-Part-1.pdf.

[7] *System Safety Engineering: Back To The Future*. URL: http://sunnyday.mit.edu/book2.pdf.

[8] *SYSTEMS TEST CONSIDERATIONS FOR HIGH PERFORMANCE LIQUID PROPELLANT ROCKET ENGINES*. URL: http://www.klabs.org/DEI/References/design_guidelines/test_series/1439msfc.pdf.

[9] *White Paper on Approaches to Safety Engineering*. URL: http://sunnyday.mit.edu/caib/concepts.pdf.