Plagiarism Declaration Here Please

Cover Sheet Here

# Abstract

Contents

[Abstract 3](#_Toc130736786)

[Introduction 5](#_Toc130736787)

[Aims & Objectives 5](#_Toc130736788)

[Background 5](#_Toc130736789)

[What is a DAQ System? 6](#_Toc130736790)

[What is LabVIEW? 6](#_Toc130736791)

[Literature Review 8](#_Toc130736792)

[Methodology 10](#_Toc130736793)

[Organisation 10](#_Toc130736794)

[Development 12](#_Toc130736795)

[Good Programming Practices 12](#_Toc130736796)

[Early Development 13](#_Toc130736797)

[Test Cases 13](#_Toc130736798)

[Mimic 13](#_Toc130736799)

[Producer Consumer Tests 13](#_Toc130736800)

[Main Branch Development 18](#_Toc130736801)

[V2.0 - Implementing a state machine for the Data Acquisition Loop 18](#_Toc130736802)

[V2.1/2.2/2.3/2.4 – Simulating Rig & Line Data 20](#_Toc130736803)

[V2.5/2.62 – Creation of a Consumer Loop 20](#_Toc130736804)

[V3.0/3.1/3.2 – Live Display Consumer Loop, Improvements to Export Loop 22](#_Toc130736805)

[V4.0/4.1 – Optimising the Export Loop 24](#_Toc130736806)

[V5.0/5.1/5.2/5.3 - Implementing EQ Ratio Calculations 25](#_Toc130736807)

[Version 5.4/5.41 – Rig Temperature Alarms & Improving EQ Ratio Calculation 27](#_Toc130736808)

[V5.5/5.51 – Improving EQ Ratio Calculations and Implementing Thermal Power 29](#_Toc130736809)

[V6.0 – Improving simulation of Flow Data 31](#_Toc130736810)

[V6.2a/6.2b – Using a formula node 33](#_Toc130736811)

[Version 6.3a – SubVI for Combustion Calculations & Finishing Touches 34](#_Toc130736812)

[SubVI’s 36](#_Toc130736813)

[Flow & Rig Data Simulation 36](#_Toc130736814)

[Flow Data to 1D Array 37](#_Toc130736815)

[Create Spreadsheet Headers 37](#_Toc130736816)

[Combustion Calculations SubVI for 6.3a 38](#_Toc130736817)

[Results 39](#_Toc130736818)

[Verification of Thermodynamics Calculations 39](#_Toc130736819)

[The final productDiscussion 41](#_Toc130736820)

[Criticism of my own code 42](#_Toc130736821)

[Going Forward 42](#_Toc130736822)

[Conclusion 43](#_Toc130736823)

[Appendices 44](#_Toc130736824)

[References 44](#_Toc130736825)

[Raw Data 46](#_Toc130736826)

[Nomenclature 46](#_Toc130736827)

[Dumping ground for stuff that doesn’t have a place 46](#_Toc130736828)

# Introduction

## Aims & Objectives

The aim of this project is to explore the suitability of LabVIEW as a replacement for the Data Acquisition and Display system in use at the School of Engineering’s Gas Turbine Research Centre (GTRC). To achieve this a new data acquisition system will be designed in LabVIEW, with the goal of demonstrating LabVIEW’s capabilities. The scope of the project does not include hardware integration.

A new DAQ system must be capable of capturing flow, pressure, and temperature data. Display this data in real time and save it to file. The new system must be more than a replacement, there is plenty of room for improvement. LabVIEW’s capabilities go beyond the functionality of the existing system. These additional capabilities should be explored.

## Background

GTRC is a multipurpose facility with a host of measurement techniques in use, carrying out a range of research into Combustion and Energy Systems. First opened in 2007 it is an experimental facility aimed at generation of original data for model validation or development (Bowen, et al., 2009). Some of the major projects currently undertaken at GTRC involve de-carbonisation research funded by FLEXIS (Cardiff University, n.d.) , Storage of Ammonia For Energy (SAFE) aims use Ammonia in the combustor to reduce NOx emissions (Valera-Medina, 2019), and multiple collaborations and grants focused on aviation caused pollution and emissions. (H2020 Raptor (863969)) (Hochgreb, 2020) (RAPTOR (863969)). The full list of GTRC publications can be found on the centre’s website <https://www.cu-gtrc.co.uk/content/27>. In addition to research GTRC also carry out combustion training.

Specific to this project is the High Pressure Optical Combustor (HPOC). On-going experiments with the HPOC are referred to as test-campaigns.

Measurements are taken within the input lines, at the inlet itself, within the combustion chamber, and in the water-cooled back end. These are then fed to an existing DAQ system in the control room at GTRC, where they are monitored live throughout the test and recorded. The conditions being monitored by the DAQ System are temperature, mass flow rate, and pressure.

A range of other diagnostic techniques are also in use such as Emissions Sampling, Flame Chemiluminescence, etc however these are not integrated directly into the existing DAQ system.

The existing system is a bespoke SCADA system built using Pro-Pack running on Windows 3.1. It’s robust and simple, with the capability to expand the number of measurements being taken to fit what’s required by a test-campaign. During a test, it displays a range of measurements within the control room at 1 Hz, whilst saving the data locally.

The system is functional but is outdated and limited in features. There is some concern about the age of hardware running Pro-Pack, memory and storage are limited. The transfer of data off the system is via floppy disk. A persistent issue is the live Air/Fuel Ratio and Equivalence Ratio Calculations are currently non-functional. This is not due to a technical fault, for these to work they’d need to be manually configured within the code each time the test conditions change. As a result they’re left non-functional on the UI. In additional a Thermal Power display does not exist.

Limitations and poor design of the existing system slows down research carried out at GTRC. During a test campaign EQ Ratio and Thermal Power are used as driving variables. Without indicators for these, the values are estimated based on the Flow conditions. It’s the hope that an improved Live Display and Data Acquisition system will increase the efficiency and speed of test-campaigns carried out on the HPOC.

Might talk about the tech specs of the actual instrumentation in use at GTRC? Difficult to do without a paper containing the info, going back or talking with Steve Morris. I’ve asked everyone I know if there’s a document somewhere containing the information/tech specs of the existing system and haven’t heard of anything

## What is LabVIEW?

LabVIEW was identified as a suitable replacement for Pro-Pack by the team at GTRC. Developed by National Instruments (NI) LabVIEW is a graphical programming language used to build Virtual Instrument Interfaces (VIs) and Data Acquisition Systems. It’s not a procedural language but is best described as a dataflow language. Officially LabVIEW is the development environment, and the code produced is known as ‘G’ or ‘G Dataflow’, although code produced in LabVIEW is colloquially referred to as ‘LabVIEW Code’ or as ‘Block-Diagram’. (LabVIEW Wiki, 2019)

LabVIEW is a good choice for this project. It is flexible and supports a wide range of hardware, as one of the industry standards it’s compatible with many existing DAQ solutions. Hardware integration in LabVIEW is quick and easy and it has a wide range of built-in user interfaces and real-time data processing tools. Lastly there is significant support and an existing codebase for data acquisition, there is no shortage of materials or resources to learn LabVIEW with.

The 2018 SP1 32bit version of LabVIEW was used for this project as it was the version available to students at the School of Engineering. Once completed the LabVIEW code can be compiled into a standalone executable.

# Literature Review

A review of self-teaching materials for LabVIEW & Data Acquisition.

LabVIEW is an industry standard for Data Acquisition.

LabVIEW Core 1.

The Participant guide for Core 1 is document meant to be worked through alongside the Core 1 course. The course itself is a mixture of online synchronous classrooms across three days totally 27.5 hours. The participant guide includes the fundamental ideas presented in the course, with example block-diagram code and VIs. It also includes simple exercises to help learn the fundamentals of LabVIEW programming. The target audience of the course are complete novices to programming in LabVIEW. I do not have access to the course itself. Only the Participant and Exercise booklets.

Core 1 covers the fundamental LabVIEW programming skills. This includes an overview of LabVIEW as a programming environment, how to create and troubleshoot Virtual Interfaces, using common programming structures like for-loops, while-loops, and case-structures (if-statements). It briefly touches on modularity and how to implement SubVIs. It covers how to acquire measurements from hardware using NI MAX and accessing files to read or log data. Lastly it introduces state machines.

This first part of the course goes into little depth or complexity with the concepts being taught. They serve as a simple introduction to LabVIEW programming, specifically navigating the UI and debugging simple programs. Many of the ideas in Core 1 are better taught by John Essick’s LabVIEW for Scientists and Engineers (Essick, Hands-On Introduction to LabVIEW for Scientist and Engineers, 2016).

LabVIEW Core 2.

Core 2 works to bridge the gap between the fundamental skills taught in Core 1 and the high-level system design focused on in Core 3. It introduces the asynchronous communication methods, implementation of basic design patterns, and basic file I/O techniques. These three concepts are key for data acquisition in LabVIEW. Large data acquisition programmes often use asynchronous design patterns like producer/consumer loops, as such asynchronous communication methods are required. Without input/output within a DAQ system there’s no mechanism to save the data.

The guide also includes sections on controlling the user interface; applicable to the project is Property Nodes. Property Nodes can read or change specific properties of an object on the front panel. An example would be dynamically changing the plot colour of a graph, or disabling/greying out a control when it shouldn’t be used. Similarly Invoke Nodes can be used to trigger specific methods or SubVIs within your programs.

It then introduces techniques for optimising/refactoring block diagrams and deploying your application as a stand-alone executable.

LabVIEW Core 3; Participant Guide

Hands on Introduction to LabVIEW for Scientists and Engineers – John Essick

A textbook with a

Symmarize,Analyse, spot gaps

LabVIEW for Data Acquisition – Bruce Mihura

# Methodology

## Organisation

All the work for this project was carried out on the authors personal laptop. To keep the project organised a Microsoft Teams was created. This served as a place to keep the documentation used to manage the project like the meeting form, but also a location to back-up LabVIEW code.

Even as a solo project scope creep can be an issue when developing software, to help combat this a Feature Tracker was created. This contained a list of requirements and features for the code, ranked by importance.

Once Main Branch Development began change control was implemented. Version numbers were formatted as v3.21, detailed notes about each iteration were kept in a notebook, documenting the objectives for each iteration, problems, bugs, potential solutions, and the eventual fix along with the reasoning behind it, lastly a list of minor changes to any existing code. The primary version number was incremented whenever a significant new feature or part of the block diagram was implemented.

There are 39 separate iterations of the software, beginning with the experimentation with Producer-Consumer Loops. Ending with version 6.3 in Mid-March. Table 1 shows the Feature Tracker at this stage. These iterations do not include failed attempts at implementing code, as an iteration would continue to be worked on until it met the objectives for that iteration, at which point it would be saved and a new version created.



Table 1 - Feature Tracker at completion of the Development Phase

All LabVIEW code can be found on the authors GitHub.

* Self Teaching
* GTRC Visit

# Development

## Good Programming Practices

* Overly Complicated Logic” applied to my iterative models, LabVIEW Core 2 5-6
* Timing, priorities, subroutine
* Memory usage
  + <https://www.ni.com/docs/en-US/bundle/labview/page/lvconcepts/vi_memory_usage.html>
* VI Execution Speed
  + <https://www.ni.com/docs/en-US/bundle/labview/page/lvconcepts/vi_execution_speed.html>
  + https://www.ni.com/docs/en-US/bundle/labview/page/lvconcepts/prioritizing\_parallel\_tasks.html
* Using profile performance window
  + <https://www.ni.com/docs/en-US/bundle/labview/page/lvconcepts/using_profile_window.html>
* Design patterns
  + <https://www.ni.com/docs/en-US/bundle/labview-real-time-module/page/lvrtbestpractices/rt_design_patterns.html>
* Subroutine priority
  + <https://www.ni.com/docs/en-US/bundle/labview/page/lvconcepts/prioritizing_parallel_tasks.html#subroutine_priority_level>
* Handling Large Projects section of LV for DAQ book
* Typedefs
* Front panel design
  + Greying out etc

## Early Development

Aaa

### Test Cases

Aaaa

### Mimic

Aaa

### Producer Consumer Tests

Producer Consumer Loops are a commonly used design pattern both in LabVIEW and other Data Acquisition applications. The producer loop is solely responsible for acquiring the data, it then shares the data with other loops running in parallel known as consumer loops. These consumer loops can then use the data without interfering with each other.

This design pattern is commonly implemented in LabVIEW using the queue operations palette, in simple terms this allows the VI to add data to a buffered list of elements. Which can then be accessed elsewhere in the block diagram. Queues operate chronologically using a first-in first-out principle. Allowing for asynchronous transfer of data between loops.

Channel wires were added to LabVIEW in 2016 (AristosQueue-(NI), 2015) and are advertised to achieve the same functionality as queues but with simpler implementation in the block diagram. Both can be a lossless communication method.

The Producer-Consumer design pattern has significant advantages for data acquisition. The producer loop can acquire the code at a fixed rate determined by the system, whilst the consumer loops can process the data at a variable speed. Decoupling data acquisition from data processing makes it possible for the producer loop to run at higher speeds than the consumer loops. It allows for simple implementation of separate processes like a live display.

Another advantage of this design pattern is that error handling and recovery mechanisms are easier to implement. If one part of the block diagram encounters an error, data acquisition can continue. Finally, by compartmentalising processes into their own loops it is easier to understand the block diagram and simplifies many of the design challenges in this project.

­To explore the feasibility of a Producer/Consumer design pattern a series of VI’s were built to understand this architecture.

The first VI built for this purpose was titled ‘Producer Consumer Test 1’ pictured in Figure 1

This VI has a simple DAQmx setup to measure data from simulated signals within NI MAX. This DAQmx read task is then fed into the queue. The queue is used to transfer the DAQmx task between the loops. The front panel consists of numerical indicators for each loop, and a stop button in the producer loop. When the producer loop is halted using the stop button, the Dequeue Element block outputs the error in the Consumer Loop causing it to stop.

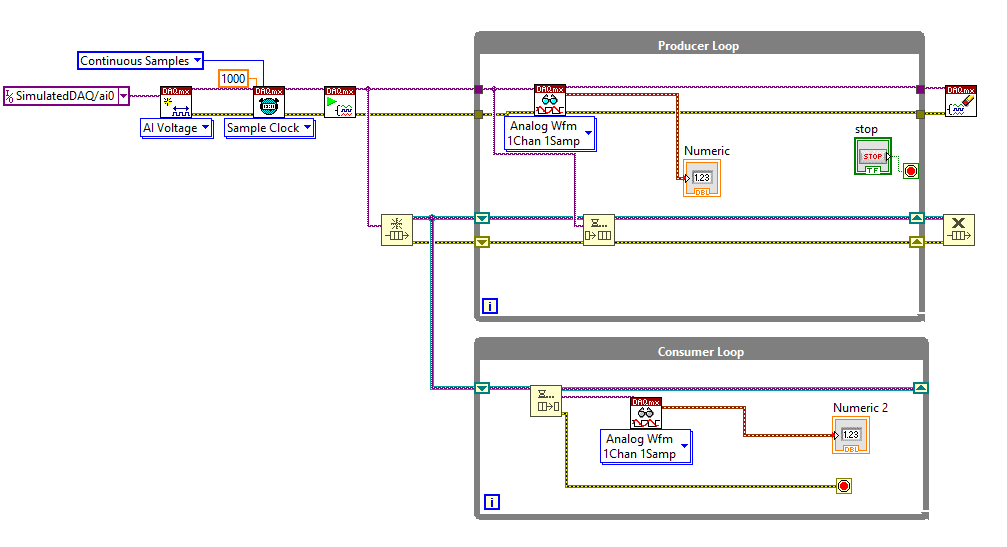


Figure 1- Producer Consumer Test 1

The VI demonstrates a misunderstanding of how the queue functionality should be used. In this first attempt both loops contain a DAQmx read task, although the data in the numerical indicators matches when ran. It’s running two separate acquisition tasks. The queue function should be used to transfer the data from the DAQmx read task, not the task itself.

‘Producer Consumer Test 2’ pictured below addressed these mistakes. Properly implementing a queue system for moving data from the Producer Loop to the Consumer Loop. In this iteration the queue is created outside the loop. Data acquired via the DAQmx Read Task inside the Producer Loop is fed into the Enqueue Element Function. This is then Dequeued in the Consumer Loop.

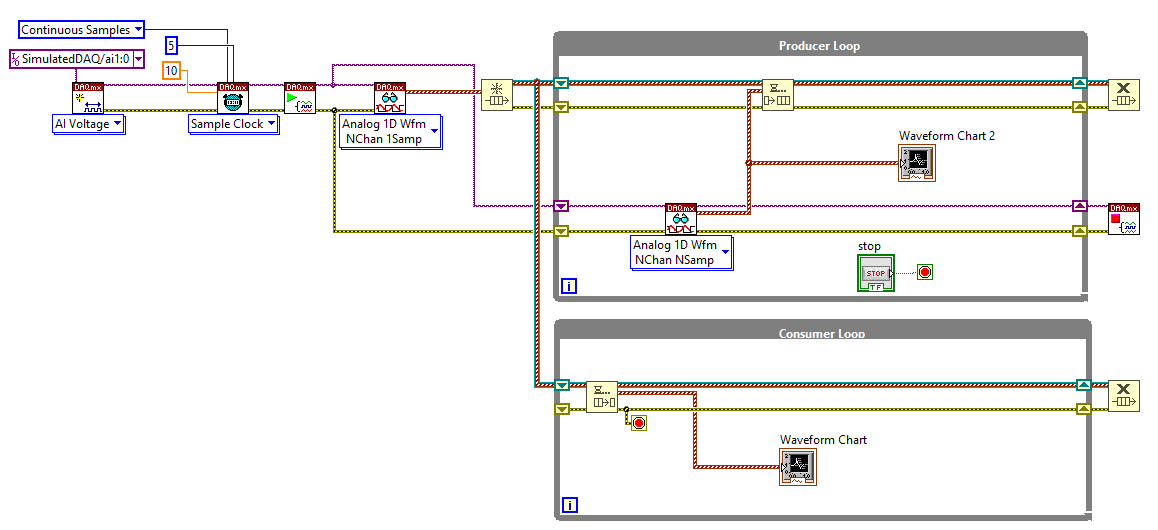


Figure 2- Producer Consumer Test 2

When the queue is created, in this case outside the loop, the element data type must be declared. Hence the use of a DAQmx read task outside the Producer Loop. This only executes once when the VI is ran. This VI includes a pair of waveform charts used to demonstrate the synchronous display across the separate loops.

The next iteration of this branch replaced queues with Channel Wires. Channel Wires were added to LabVIEW with the purpose of reducing the use of existing asynchronous data transfer, most notably queues. It also introduced a pair of nested case-structures used to take a rolling average of the signal. This is shown by Figure 3.

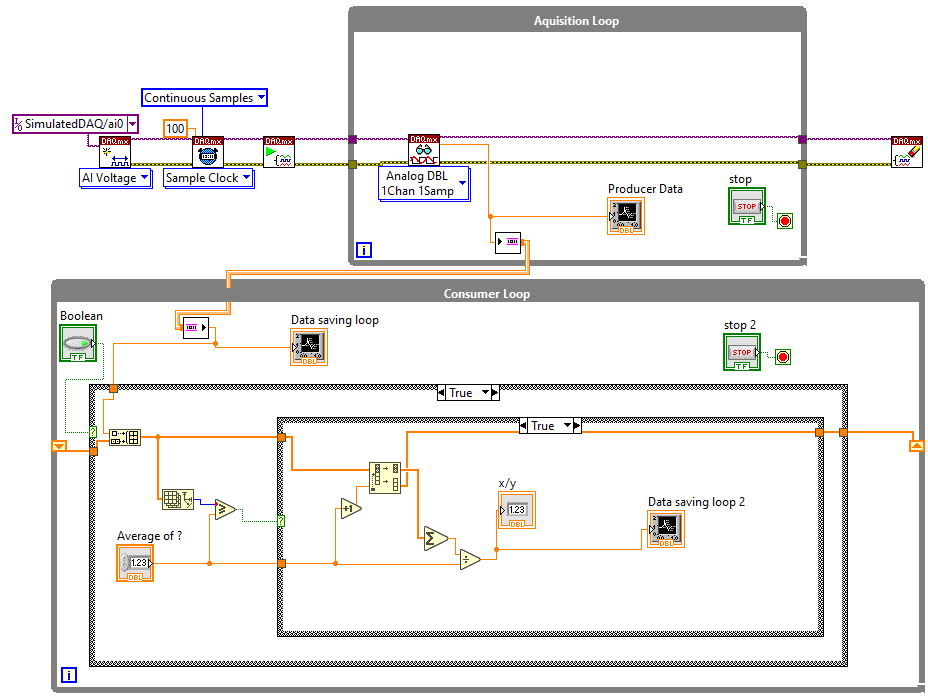


Figure 3 – Producer Consumer Test with Channel Wires

The way the average was being calculated uses overly complex block diagram logic, and only works for a 1-dimensional array of data. If two channels of data were being captured by DAQmx, this method would not work.

Diagram, schematic

Description automatically generated

Figure 4 - A better system for taking the average

Figure 4 demonstrates a better method for calculating the average of a chosen number of samples. It also features the addition of an extra channel of data to the DAQmx task, and a timing mechanism making use of the ‘High Resolution Relative Seconds’ VI. This SubVI provided with LabVIEW returns the current time in seconds. This can be subtracted from a previous value to get elapsed time. This is used in Figure 4 to calculate how long the acquisition has been running for each loop iteration.

This is a commonly used timing method in LabVIEW and was eventually implemented in the final version of the software, however documentation states this timing method may drift over long periods (National Instruments, 2023). For this reason a second absolute timing measurement was later implemented.

The last Producer/Consumer test build (Figure 5) further improved the method of averaging the signals from the producer loop. Making use the auto-indexing function of For-Loops, by placing the Channel Reader within a For-Loop, it creates an array of values. The size of this array is controlled by changing the number of iterations the For-Loop undergoes. This 2-D array of raw data is transposed and fed into another For-Loop containing LabVIEW’s standard ‘Mean’ SubVI.

Diagram

Description automatically generated

Figure 5 – Final Producer Consumer Test with Signal Averaging and Clusters.

As shown this method significantly reduces the complexity of the block diagram, at the cost of more nested loops.

## Main Branch Development

After building simple Producer/Consumer design patterns, developing the early test projects, and creating the mimic, development began on the VI that would become the finished product. The version number began at 2.0 as this code carried on from the Producer/Consumer tests.

### V2.0 - Implementing a state machine for the Data Acquisition Loop

The priority was building a robust method for Data Acquisition, John Essick’s Hands on Introduction to LabVIEW for Scientists and Engineers (Essick, Hands-On Introduction to LabVIEW for Scientists and Engineers, 2016) section 12.6 details how to build a data acquisition state machine using the DAQmx palette. To begin this was followed as a rough guide.

Using a state machine comes with many advantages, it gives better control over the behaviour of the VI. Overhead operations such as setup and shut-down actions are only executed when required. The Profiler Performance and Memory window shows that these actions can take significant CPU time, the data for this testing can be found in Appendix X. It can wait on user input or certain conditions to be met before progressing to the next state, data can be transferred between states using local variables or a shift register.

A state machine helps to keep the block diagram organised and compact, making it easier to understand, as it’s a common design pattern used in LabVIEW it can be easily recognised by an experienced LabVIEW user.

Version 2.0 implemented a state machine capable of data acquisition and display. The state machine is controlled by an Enumerated Type Control (Enum for short), this is a numbered list which can be fed to case-structure to control its state. The case-structure is controlled by passing the desired Enum via shift register to the next iteration of the loop. An example of this can be seen in the top right of Figure 6.

Beginning with a ‘Create Task’ state, this created the DAQmx read task, once this has happened it progresses onto ‘Change Settings’. During this state it idles waiting for the user to input the refresh rate and confirm by pressing the OK button.

Once the OK button has been pressed it moves onto ‘Check Settings’ this state is used to set the timing of the sample clock and acquire a datum start time using ‘High Resolution Relative Seconds’ VI. It also contains a Boolean check which can be used to compare the entered variables against any desired conditions, if this Boolean check is passed it moves onto the ‘Read Data’ state, if not it returns to the ‘Change Settings’ state.

Pictured in Figure 6 is an example of the ‘Read Data’ state. Once in this state the loop begins to record data. A property node is used to disable and grey out the DAQ properties control. This is important as once this state is entered the sample rate is fixed, therefore being able to change it in the UI could be confusing. Once data acquisition is finished the user can press the stop button, moving onto the ‘End Task’ state, where the DAQmx task is cleared, and any reserved resources are released. The Boolean logic at the bottom of the loop then halts the loop once the Enum reaches End Task.

This overall architecture went largely unchanged and was soon developed into the Producer Loop, although the specific behaviour and contents of each state was tweaked significantly through development.

Graphical user interface, text, application

Description automatically generated

Figure 6 - v2.4 'Read Data' State

### V2.1/2.2/2.3/2.4 – Simulating Rig & Line Data

To further demonstrate LabVIEW’s capabilities more signals would need to be simulated. This could be achieved with additional physical channels added to the NI MAX Simulated Device.

NI MAX simulated devices can only “create a noisy sine wave for all analog input channels. Configuration of other simulated data is not available” (National Instruments, 2021). Creation of more of these signals doesn’t demonstrate any further LabVIEW skills or functionality, and all the signals would be identical sin waves.

Simulate Signal Express VI was used to generate additional data. By using this Express VI it’s possible to change all aspects of the signal (shape, frequency, amplitude etc) using controls on the front panel. Signals were added to replicate temperature, pressure, flowrate for each of the five feed lines, and the Rig Temperatures. Later in the project this data is used to demonstrate combustion calculations and alarms. These were then bundled into a cluster to be used with a Channel Writer.

The block diagrams used to simulate this data was packed into SubVIs which are detailed in their own section of the report. Name and number section please bossman

There is more data displayed on the existing UI than this, however they are only live display. The simple addition of a dozen live indicators serves no purpose for this project.

### V2.5/2.62 – Creation of a Consumer Loop

The objective of v2.5 was to add a consumer loop responsible for exporting data. A lossless high-speed channel was used to move the data between the loops, and the ‘Write Delimited Spreadsheet’ VI was used to write the data. To do this the data was unbundled and organised into an array of strings. A SubVI was built to do this for the rig temperatures as it would need to be repeated elsewhere in the code and took up considerable space on the block diagram.

Like the producer loop, this consumer loop uses the State-Machine design pattern. It begins with an idle ‘Waiting to Record’ state, allowing the user to begin recording from the front panel. From here it creates the spreadsheet and writes in the headers.

Versions 2.6 (Figure 7) saw improvements to the way the data was converted into an array of strings, more control over the behaviour of front panel objects, the spreadsheet was changed to be .csv, and the addition of a general error handler. Importantly the state machine only progresses past the ‘Recording’ once the Channel Reader has finished receiving data and the buffer is empty. This prevents data loss when recording is finished.

Graphical user interface

Description automatically generated with medium confidence

Figure 7 - v2.62 Export Loop, Read State

### V3.0/3.1/3.2 – Live Display Consumer Loop, Improvements to Export Loop

Versions 3.0 – 3.2 saw the addition of a basic consumer loop for live display shown in Figure 8. Timeline

Description automatically generated

Figure 8 - v3.2 UI Consumer Loop

This loop contains displays for each of the clusters of data being captured, as well as the time-stamp recorded in the Producer Loop, and a timestamp generated and displayed in the UI loop. The loop is halted using the ‘done?’ terminal on the channel reader which returns true once the channel has cleared its buffer and is receiving no more data. The addition of separate timestamp displays for each loop allows the user to identify if the live display has fallen out of sync with the acquisition.

As well as a basic live display, these iterations saw significant improvements to the export loop, fixing several problems:

If the data acquisition ran for too long whilst the export loop wasn’t recording, the buffer on the channel would fill up, causing the VI to crash. This was because the producer loop would be writing data to the channel, which would then be trapped with nowhere to go as no Channel Reader had been created yet. Once the consumer loop moved to writing the data to file, it would first write all the saved data, emptying the buffer.

A potential fix for this, would be adding an additional state to the producer loop, where the data is acquired but is not sent to the channel writer. A simpler solution was used, by adding a channel reader to the ‘Waiting to Record’ state of the export loop which didn’t output to anything, the channel would be read, and the buffer kept empty, without needing additional states or block diagram logic.

The next issue was that Microsoft Excel was truncating the exported timestamps when opening the CSV. When opening the file in Notepad the digits existed in plain text format, but when opened in Excel the timestamps lacked the seconds measurement. Adding code to format the date/time stamp as two separate strings corrected this error.

If the exporting filename/location was already open in windows when the export loop tried to create the .csv, no error would occur initially. Only flagging an error when the export loop exits, losing all recorded data. Whilst the chance of a specific file being open during recording is unlikely, it showed that errors could occur causing all recorded data to be lost without the user knowing until after the test campaign would be finished.

Debugging showed that the error was generated within the ‘Creating Spreadsheet’ state but wasn’t addressed by the VI until the export loop exited. To remedy this an error handler was placed in this state connected in series to the Create Spreadsheet VI. If an error does occur the user is notified, and Boolean logic returns the loop returns to the ‘Waiting’ state. The loop only continues onto the ‘Recording’ state if there have been no errors with the creation of the spreadsheet. Adding an error handler to the ‘Recording’ state causes the UI to become unusable as it creates a new error dialogue with each loop iteration.

Another solution would be to concatenate the timestamp onto the end of the filename when the spreadsheet file is created resulting in a novel filename each time. Attempting to implement this prevented the file path dialogue from automatically appearing if a file path hadn’t been entered already. This was a key safeguard preventing the user from recording without entering a file path, so the idea was scrapped.

Lastly the creation of the array of strings used to generate the spreadsheet headers was moved into its own SubVI.

### V4.0/4.1 – Optimising the Export Loop

At sample rates higher than ~ 50Hz the data export loop was unable to keep up with the acquisition loop causing the software to error and the export loop to terminate. This is a significant issue, if this occurred during a test campaign all data beyond that point would be lost. The export loop needed to be improved to run faster.

Possible solutions were: pre-allocate the file size on the disk to increase write speed, the use of TDMS instead of the ‘Write to Speadsheet’ VI, using multiple export loops to alternate in writing the data, batch the data and write in batches, hold all the data in memory and write when the loop terminates. This last option was not seriously considered.

The first step was to streamline the block diagram within the loop. The SubVI use to unbundle the flowrate data was improved by replacing some Get Array Size operations with constants as these values never change. The execution priority of this SubVI was increased slightly.

Flowrate Data, the DAQmx Data, and the Rig Temperatures were combined into an array of doubles before being turned into an array of strings in one operation, rather than being done in three separate operations. Concatenating strings like this is advised by National Instruments (National Instruments, 2023).

Batching the data is simply done by putting the channel reader inside a for loop, this indexes the 1-D array of strings into a 2-D array. Which can be fed directly to ‘Write Delimited Spreadsheet’ VI. This is shown in Figure 9. The iterations on the for loop is set to match the sample rate of DAQmx, at 100Hz sample rate. For example the loop will collect an array with 100 rows of data, which will be written once per second. This sped up execution as file I/O calls each incur a significant overhead (National Instruments, 2023), by batching the data the rate at which an I/O call occurs is significantly lower.

Batches could be made larger, potentially making a file call every few seconds. The bigger the buffer the more data must be held in memory. This could be further improved by initializing the array created by the for loop to its known size, before filling it. Changing the size of the buffer is simple should the user want to and could even be controlled from the front panel by a simple numerical control. Initializing the array first is a minor improvement as it’s initialized only during the first loop iteration and is overwritten with each subsequent iteration.

Graphical user interface

Description automatically generated

Figure 9 - v4.1 Data Export Loop, Recording State

With these changes the VI can now run at 500 Hz for an extended period of time, providing a significant safety factor given the existing system runs at 1 Hz. Sampling rates this high and beyond are void given the time constant on a typical thermal couple is slower than this. (Placeholder1)

This version of the code was presented to Matt Allmark for a brief code review. The overall structure of the code was discussed. The use of Producer-Consumer and State-Machine design patterns were well received. Some key feedback was the consideration of loop priorities, timing mechanisms, and the unnecessary use of a lossless highspeed channel for the UI Loop.

The inclusion of a wait-milliseconds timer in the Export Loop was discussed as potentially useful, however later testing showed this to be unnecessary. The use of a for-loop to batch the data for periodic writing to file forced the loop to iterate at a slower pace (1 Hz), as the Export Loop would wait for the for-loop finish batching 1 second’s worth of data before writing to file and beginning its next iteration. The batching based on sample rate acts as a forced timing mechanism.

Lastly the lack of code comments was noted, up to this point the code did not contain any comments. As this is a solo design project it had not hampered development, however commenting is best practice. Comments were included in new code from version 5.0 onwards and retroactively added to old code in later versions.

### V5.0/5.1/5.2/5.3 - Implementing EQ Ratio Calculations

The existing DAQ system at GTRC can calculate the Air-Fuel-Ratio and the Equivalence Ratio from the Flow Data. The EQ ratio calculation on the existing DAQ does not function, and a Thermal Power calculation is not present. During a test campaign these variables are used as benchmarks throughout the experiment. Version 5 aimed to implement a simple system to handle these calculations and display them without the user needed to access the code.

To do this a look-up table was created containing some common fuels used at the GTRC, on this table was the AFR and Lower Heating Value for that substance. Controls were added to a tab on the front panel where the user can select the substance being fed in each line, or enter custom properties if the substance isn’t an available pre-set.

The look-up table is turned into an array of strings, a for-loop is used to index each of the lines. This loop reads in the substance for each line, searches the array containing the look-up table to get an index number for that substance, which is then used with a case-structure to obtain the line’s variables. An early version of the block diagram code for this is shown below in Figure 10.

Graphical user interface

Description automatically generated

Figure 10 – v5.0 Block Diagram for AFR Calcs

Version 5.1 saw the addition of live graphing of temperature and pressure.

The addition of this look-up table for AFR created a bug where the Flow displays were disabled and greyed when the VI was run and the graphs didn’t display properly, this would resolved once a substance was set and the AFR variable was initialised.

This was fixed in later versions with the addition of an ‘Empty’ Enum option for the lines, and a Local Variable to set and initialise these parameters. Using local variables is considered bad practice in LabVIEW as they are slower than signal wires. It is acceptable in this case as the loop is not required to iterate at a high rate.

In Versions 5.2/5.3a DAQmx timing node was added to the acquisition loop, measure the actual sample rate. Whilst this isn’t required when simulating signals, during real acquisition the actual sample rate can vary from what’s been set. Having a timing node to measure the real sample rate is considered best practice.

The high speed lossless channel to the live display loop was changed to a single element lossy channel. As a lossy channel it can still write at the speed of the acquisition loop, but can be read at any speed. If the buffer is full it overwrites the oldest element in the buffer. High speed live display is unnecessary and slows down any loop it is in. This change allows the control of the UI’s refresh rate using the code shown in Figure 11.

Diagram

Description automatically generated

Figure 11 - UI Refresh Rate Block Diagram Code

Changes in refresh rate can cause display issues with the live charts as they have a fixed chart history length. This chart history length cannot be changed programmatically. (National Instruments, 2020)

Another significant change in this version was combing the simulation SubVI’s for Rig and Flow into one SubVI. The execution priority of this SubVI was set to ‘Time Critical Data Acquisition’, which is one before ‘subroutine’, this was because SubVI’s containing simulated signals cannot have subroutine execution priority.

The acquisition loop for version 5.3 can be seen below in Figure 12

A picture containing schematic

Description automatically generated

Figure 12 - V5.3 Aquistion Loop, Read State

### Version 5.4/5.41 – Rig Temperature Alarms & Improving EQ Ratio Calculation

For any experimental setup involving high temperatures and pressures safety is a concern. The existing DAQ system has alarms built into the display. Flashing red when a temperature goes above a threshold value. This can be implemented in LabVIEW using a property node to cause the indicator to blink. Versions 5.4 sought to demonstrate this capability using the Rig temperatures.

A pair of tabs were created on the front panel for the Rig temperature indicators and their threshold values. Allowing the users to change the alarm thresholds from the front panel.

Property nodes in LabVIEW can only apply to one front panel object at a time. The initial implementation of these alarm thresholds was to take the Rig Temperatures, and their thresholds as arrays. These arrays were fed into a for-loop to be compared which creates an array of Booleans. This Boolean array is then used to control each front panel indicator separately.

There was concern that if the thresholds tab was open on top of the indicators, the user would be unaware a threshold had been exceeded. To combat this the VI programmatically changed the tab back to the Rig Temperature Indicators tab.

This solution caused issues, as when an alarm was activated, changing the threshold was very difficult as the VI would keep switching to the indicators tab. This solution was removed in favour of causing the threshold values in the thresholds tab to blink when they had been exceeded. Informing the user that the threshold had been exceeded even when looking at the wrong tab. Hence the existence of two columns of property nodes in the block diagram pictured below in Figure 13.

Diagram, schematic

Description automatically generated

Figure 13 - Early Block Diagram Code for Alarm Thresholds

I’m aware this style below does not belong in a dissertation, just getting the points down until I figure out how to word it.

I was unhappy with this implementation of alarms and wanted a more elegant solution. The LabVIEW discussion forums contained a method for extract the names of indicators from a cluster (Dayley, 2008). This makes use of a property node to extract the reference of an individual indicator.

This method was applied to the Rig Temperatures Cluster and Rig Alarm Thresholds to obtain arrays of their individual indicator’s references. Using a for-loop these are compared as before but the property nodes are assigned by reference for each iteration of the loop. Allowing for individual control of each indicator, without individual property nodes. This code is shown in Figure 14.

Diagram

Description automatically generated

Figure 14 - Improved Block Diagram Code for Alarm Threshold

As seen, this is a more elegant solution and simplifies the block diagram. It’s also scalable if more indicators are added to these clusters. One downside of this solution is the use of local variables as these take longer to transfer data than wires (National Instruments, 2023), however this code is placed the UI Loop, and is not required to iterate at high speed.

An alternative would be to install and use NI’s Datalogging Supervisory & Control Module (DSC). This module provides advanced functionality for shared variables in LabVIEW. This module can be used to manage alarms and events for a very large number of variables, it also provides tools for historical or real-time graphing of trends (National Instruments, 2012) (National Instruments, 2023). The module would be worth investigating for use with the GTRC.

### V5.5/5.51 – Improving EQ Ratio Calculations and Implementing Thermal Power

5.5 saw the final implementation of EQ Ratio Calculations and Thermal Power. These two processes were bundled into a single for-loop, and moved from their own while-loop to the Live Display while-loop. This reduced the total number of top level loops to 3. The block diagram code used to do this is shown in Figure 15.

Diagram

Description automatically generated

Figure 15 - Version 5.51 Block Diagram for Combustion Calculations

### V6.0 – Improving simulation of Flow Data

During a Test Campaign the flowrate of the individual lines is controlled separately, and Thermal Power and Equivalence Ratio are used as driving variables or benchmarks. To help with demonstration and testing of the VI, individual controls for flowrate, temperature, and pressure, were added to the front panel. These are shown below in Figure 16. The SubVI used to simulate the data was changed to be able to take these values input as a cluster.

Graphical user interface

Description automatically generated with medium confidence

Figure 16 - Front Panel Controls for Flowrate, Temperature, Pressure

Whilst the exact values simulated for the flow data are not relevant, it’s useful to be able to drive the VI and see how it responds to changing values. The last tab ‘Species List’ contains the look-up table for AFR and Thermal Power.

In this version the overall front panel was organised, and comments were added to the block diagram. The front panel at this time is shown in Figure 17.

A picture containing text, indoor

Description automatically generated

Figure 17 - Front Panel for Version 6.0/6.1l

### V6.2a/6.2b – Using a formula node

LabVIEW’s formula node is a structure on the block diagram which allows the user to enter traditional written code, commonly used for mathematical formulas. The syntax is based on C. (National Instruments, 2023).

There was a mistake in the code used to calculate EQ Ratio, in version 6.2a this was corrected using wires. This solution is untidy and unintuitive. As an alternative v6.2b uses a formula node to handle these calculations. As this formula node uses text-based programming, it’s easier to understand at a glance. This section of the block diagram for versions 6.2a and 6.2b are shown in Figure 18 and Figure 19 respectively.

Diagram

Description automatically generated

Figure 18 - Version 6.2a Combustion Calculations Block Diagram

Graphical user interface, text, application, email

Description automatically generated

Figure 19 - Version 6.2b Combustion Calculations Block Diagram (using formula nodes)

Formula nodes can contain if statements and loops, in theory the whole section of code used to assign AFR values and calculate EQ Ratio could be put into formula node, however wires are the fastest most efficient data transfer in LabVIEW. Formula nodes can be slow in comparison. (Placeholder2)

The formula node here is used within the UI Loop, which isn’t required to update more than a few times a second. Therefore it shouldn’t have a significant impact on execution.

### Version 6.3a – SubVI for Combustion Calculations & Finishing Touches

Version 6.2a was carried forward. To simplify the block diagram the combustion calculations were combined into a SubVI. Figure 20 shows the improvement in legibility. The contents of this SubVI are covered in section X of the report. To reduce overhead when calling this VI, it’s set as an inline function. Its execution priority is unchanged as the UI display loop is the least time critical loop.

Diagram

Description automatically generated with medium confidence

Figure 20 - Version 6.3a Block Diagram (Implemented SubVI)

In this final version minor improvements were made to the block diagram; code comments were added, and cosmetic improvements were made to the front panel. The full breakdown of the final code can be found in the Results section.

## SubVI’s

Modularity is considered good practice in programming, LabVIEW is no different. SubVI’s are the LabVIEW equivalent of functions from traditional text-based programming languages. They allow the creation of modular self-contained code, which can be called by a larger VI to perform a certain task. Use of SubVI’s is encouraged as it helps keep code manageable. (National Instruments, 2022)

Many of the block diagram objects in LabVIEW are pre-made SubVI’s, an example of this is the DAQmx Read object. This can be opened with a double click to view its block diagram code.

User made SubVIs can be customised to have input and output terminals of any datatype, and can even have a custom logo. SubVIs get included in a LabVIEW project, and saved with the rest of the code when an executable is created.

For this project several SubVIs were made.

### Flow & Rig Data Simulation

The first SubVI created was used to simulate some of the data used by the existing DAQ system, this is called within the data acquisition loop. The final iteration simulated Flow and Rig temperatures, using the simulate signal express VI. The Flow Data is controlled by front panel objects. This SubVI is shown by Figure 21.

The auto indexing function of the for-loop is used to create an array of five clusters, one for each feed line at GTRC. Each cluster contains temperature, pressure, and flowrate. As this SubVI is called within the acquisition loop execution priority of this SubVI is set to ‘time-critical’ which is the second highest behind subroutine. The subroutine priority is incompatible with the simulate signal express VI. The preferred execution style is set to ‘data acquisition’.

Diagram

Description automatically generated

Figure 21 - Combined Simulation SubVI Final Version

### Flow Data to 1D Array

Figure 22 is a SubVI used to unbundle the array of clusters containing the flow data into a 1D array. This is required by the export loop as the ‘Write to Delimited Spreadsheet VI’ cannot receive an array of clusters. By unbundling the clusters into one array it can be combined into an array with the rest of the data for writing to file.

Diagram

Description automatically generated

Figure 22 - Flow Data to 1D Array SubVI

This is set to be a high priority inline SubVI. Subroutine priority could be used here. As this code is called within the data export loop, it make sense for it to have a lower execution priority than the Simulation SubVI called within the acquisition loop. Setting this as an inline function increases execution speed by compiling the SubVI code into the calling VI code, reducing overhead. (National Instruments, 2023)

### Create Spreadsheet Headers

The code used to generate the spreadsheet headers was moved into it’s own SubVI. This was done only to improve readability of the ‘Create Spreadsheet’ state of the Export Loop. As this VI only executes once the priority is not important. The SubVI works by creating an array of strings and then concatenating them into one long string. Which is then fed to the ‘Write Delimited Spreadsheet VI’. Due to the size and simplicity of this VI, it’s figure is contained in the Appendices.

An improve to this VI would be to extract the cluster names of the signals with a property node using their reference. This would be scalable, as currently if more signals are acquired additional spreadsheet headers would need to be added manually to the code.

### Combustion Calculations SubVI for 6.3a

Diagram

Description automatically generated

Figure 23 - Combustion Calculations SubVI

Above in Figure 23 is the contents of this SubVI.

# Results

## Verification of Thermodynamics Calculations

After introducing AFR, EQ Ratio, and Thermal Power displays into the VI testing was required to verify the code was correctly setup. Table 2 shows the input values used to setup the experiment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Line** | **Substance** | **AFR** | **LHV [kJ/kg]** | **Flowrate (F) [g/s]** |
| 1 | Air | 0 | 0 | 9 |
| 2 | Ethanol | 9 | 26.7 | 8 |
| 3 | Butane | 14.98 | 45.75 | 3 |
| 4 | Custom | 7 | 5 | 4 |
| 5 | Empty | 0 | 0 | 0 |

Table 2 - Input Values for EQ Ratio, AFR, and Thermal Power Calculations

This setup was chosen as it uses each of the four available options for a line (Air, Fuel, Custom, Empty), and uses two different fuels to verify the look-up tables functionality. The VI calculates and displays: Total Fuel Flow, Total Air Flow, Stoichiometric AFR, Mixture AFR, EQ Ratio, and Thermal Power. Equations for these values are shown below.

[Eq 1.]

[Eq 2.]

[Eq 3.]

[Eq 4.]

Using the input values shown in Table X and the equations above theoretical values were calculated. Versions 6.2a which uses wires for the calculations and 6.2b which uses a formula node were tested. In both cases the values came out identically to four decimal places. Confirming the code used for the calculations is correct. The results of this test and Excel formulas used can be found in Appendix X. It should be noted that it’s possible some of the values for AFR and Thermal Power in the Look-up table are inaccurate as they’re merely for demonstration purposes. This doesn’t impact this testing as the values used by the software and in the theory kept consistent.

# The final productDiscussion

## Criticism of my own code

Problems, bugs, solutions, improvements I’d like to make

Addeding CSV to end of filepath automatically without not detecting when it’s not filled

Replace the line display number with the substance currently in the line

Dynamic and static analysis

Desktop execution standalone tool

Toms labview adventure

## Going Forward

Changes, improvements, what would need to be done to bring this code to productions, implications of these changes, how it would be integrated at GTRC, adding of a configuration file

Things to look into, questions

Networked DAQ

DSC and other modules

Improve the graphics

Compare formula node thing Vis with profile tool

# Conclusion

# Appendices

## References

AristosQueue-(NI). (2015, February 4). Getting Started With Channel Wires. Retrieved Feb 27, 2023, from https://forums.ni.com/t5/LabVIEW-Channel-Wires/Getting-Started-With-Channel-Wires/td-p/3505658

Bowen, P., Syred, N., Roberts, P. A., March, R., Griffiths, Griffiths, A. J., . . . Morris, S. (2009). GTRC First Year Contribution to Progress in Combustion and Energy Systems. *International Flame Research Foundation 16th International Members' Conference.* Boston.

Cardiff University. (n.d.). *Flexis*. Retrieved from Gas Turbine Research Centre: https://www.cu-gtrc.co.uk/flexis

Dayley, T. (2008, May 3). Is there a "Get Cluster Names" VI? . 1. Retrieved March 22, 2023, from https://forums.ni.com/t5/LabVIEW/Is-there-a-quot-Get-Cluster-Names-quot-VI/td-p/666555

Essick, J. (2016). *Hands-On Introduction to LabVIEW for Scientist and Engineers* (3rd ed.). Oxford: Oxford University Press.

Essick, J. (2016). Hands-On Introduction to LabVIEW for Scientists and Engineers. In J. Essick, *Hands-On Introduction to LabVIEW for Scientists and Engineers* (Third ed., pp. 518-530). Oxford: Oxford University Press.

H2020 Raptor (863969). (2019, November 1). *Research of Aviation PM Technologies, Modelling and Regulation*. doi:10.3030/863969

Hochgreb, S. (2020, April 7). *Details of Grant for Tracer-free, non-intrusive, time- and space-resolved temperature and scalar measurements*. Retrieved March 2023, 24, from Engineering and Physical Sciences Research Council: https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/T030801/1

National Instruments. (2012, June). *Getting Started with LabVIEW Datalogging and Supervisory Control Module.* Retrieved from National Instruments: https://www.ni.com/docs/en-US/bundle/labview-datalogging-and-supervisory-control-module-getting-started/resource/372946d.pdf

National Instruments. (2020, December 03). *Knowledge.NI*. Retrieved March 21, 2023, from Change Waveform Chart History Length in LabVIEW: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YI6UCAW&l=en-GB

National Instruments. (2021, August 31). *Considerations With NI-DAQmx Simulated Devices*. Retrieved March 19, 19, from Knowledge.NI: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019Nw0SAE&l=en-GB

National Instruments. (2022, December 8). *Create and Configure a LabVIEW SubVI*. Retrieved from LabVIEW Support: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YK4VCAW&l=en-GB

National Instruments. (2023, 02 21). *Datalogging and Supervisory Control Module*. Retrieved from National Instruments Product Documentation: https://www.ni.com/docs/en-US/bundle/labview-datalogging-and-supervisory-control-module/page/lvdscconcepts/dsc\_module\_defa.html

National Instruments. (2023, 02 23). *Formula Node Syntax*. Retrieved from LabVIEW Product Documentation: https://www.ni.com/docs/en-US/bundle/labview/page/lvhowto/formula\_node\_syntax.html

National Instruments. (2023, 02 23). *High Resolution Relative Seconds VI*. Retrieved 03 15, 20, from NI Product Documentation: https://www.ni.com/docs/en-US/bundle/labview/page/glang/high\_res\_rel\_sec.html

National Instruments. (2023, 03 15). *Minimizing SubVI Overhead*. Retrieved from National Instruments: https://www.ni.com/docs/en-US/bundle/labview-nxg-creating-first-application/page/minimize-subvi-overhead.html

National Instruments. (2023, February 23). *VI Execution Speed*. Retrieved from LabVIEW Product Documentation: https://www.ni.com/docs/en-US/bundle/labview/page/lvconcepts/vi\_execution\_speed.html

NI. (2023, March 15). *Strategies for Improving VI Execution Speed*. Retrieved March 02, 2023, from National Instruments: https://www.ni.com/docs/en-US/bundle/daqexpress/page/strategies-for-improving-vi-execution-speed.html

RAPTOR (863969). (2019, June 1). *Details of Grant for Assessing aViation emission Impact on local Air quality at airports: TOwards Regulation*. doi:10.3030/814801

Valera-Medina, A. (2019). Storage of Ammonia For Energy (SAFE) - AGT Pilot Grant. Retrieved March 24, 2023, from Engineering and Physical Sciences Research Council: https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/T009314/1

## Raw Data

## Nomenclature

DAQ

# Dumping ground for stuff that doesn’t have a place

* + Accuracy of timing apps in LabVIEW
    - <https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P9QiSAK&l=en->