Technical User's Manual for *Analog Data Recorder*

Edited by Noah Gula

Propellant Tank Research Team
Ohio State University

This manual corresponds with version 1.0 of Analog Data Recorder.

The Propellant Tank Research team is composed of Shreyas Doejode, Noah Gula, Jordan Lombardo, and Tyler Schell. The team is advised by Dr. John Horack.

PROPELLANT TANK RESEARCH

Revision History

Version	Comments	Date
A	Initial release	2020/1/1

Analog Data Recorder

Technical User's Manual

Noah Gula

Contents

Introduction

Do you leave instrumentation setup and validation until the last minute of the morning of test day? Are you clueless on using LabVIEW? Do you want to process your data in MATLAB? If you answered yes to any of those questions, this app may be right for you. Consult your testing specialist for more information.

The Analog Data Recorder app was designed by the Propellant Tank Research (PTR) team in order to streamline the data acquisition activities when performing tests on the bipropellant piston tank. The software is now being shared with you.

In the OSU AAE capstone course, the students are provided a crash course on using LabVIEW. However, from the PTR team's experience, the available hardware was not compatible with the available software, and so we were unable to complete our testing activities. Therefore, we investigated other ways of acquiring our data and stumbled upon MATLAB's *Data Acquisition Toolbox*. This toolbox provides resources to pragmatically interface with National Instruments hardware (and several other hardware vendors).

While this toolbox may seem overwhelming, and perhaps the documentation isn't very helpful per MATLAB's standards, it still offers very powerful tools which the ADR app makes use of. Getting started with the toolbox is fairly straightforward, but there are a few steps which are imperative to follow. Those are discussed in the Quick Start Guide.

The app was designed with somewhat limited functionality. If all you need is to record data off a few sensors, but don't need to output data or control hardware, then this app should be compatible with your needs. Currently, analog devices are supported, but digital device support will be added in a future release.

This documentation serves as the manual for the ADR app. It's usage will be described here, as well as how to set it up and tips for troubleshooting problems. On that note, it is important to mention that this software is provided as-is and has no warranty. If there are bugs and issues with the software, well, we're aerospace engineering students. Not computer science students. That said, we welcome suggestions and encourage you to modify the app to fit your specific needs. For technical support, we refer you to this document.

Known Limitations

While the goal was to develop a comprehensive, generalized piece of software which would accomplish everyone's need, realistically, this was not achieved. Hence, there are some limitations to the capabilities of ADR. These are:

- Only NI devices are compatible. Or at least, only one vendor is supported at a time. The software is currently designed for NI devices, but to switch over to a different vendor would require a rewrite of the configuration code, since each vendor configures their devices differently.
- The original intentions were to allow for the recording of digital input signals, however, these devices fundamentally work in a different way than their analog counterparts. It was decided to not pursue their development because in its current form, ADR would not meaningfully be able to interface with them.
- Only one's imagination

Quick Start Guide

The Analog Data Recorder app allows for the recording on analog data acquisition devices. These include the NI USB-6009, or the NI compact DAQ chassis, amongst others. This quick start guide will teach the user how to briefly setup, record, and save data.

1.1 Installation

Go to https://github.com/gulanr/adr and download the repository. Extract the downloaded files, and double click on the filed named **Analog Data Recoder.mlappinstall**. In MATLAB, the app will then be available on the **APPS** tab.

1.2 Requirements

The following are dependencies of ADR:

- Data Acquisition Toolbox
- National Instruments Hardware Support Package
- Supported data acquisition hardware

Appendix B provides details on the installation process.

1.3 Using the App

The interface of the app is broken down into four tabs:

Configure This tab handles the device discovery and allows the creation of a configuration file to save channel setup.

Record This tab handles the recording activities of the software. A channel configuration file can be loaded which is used to create the `daq` session and record data.

Save This tab saves the most recently recorded data. Meta information can be added to identify the run.

Process This tab is left blank for the user to design and implement. Possible use cases could be pre-processing activities such as converting voltage measurements to other physical units, or to combine data sets from multiple runs into one.

To begin using the software, first, plug in the hardware devices you wish to record from. Then, open the app. When opened, the app will look like this:

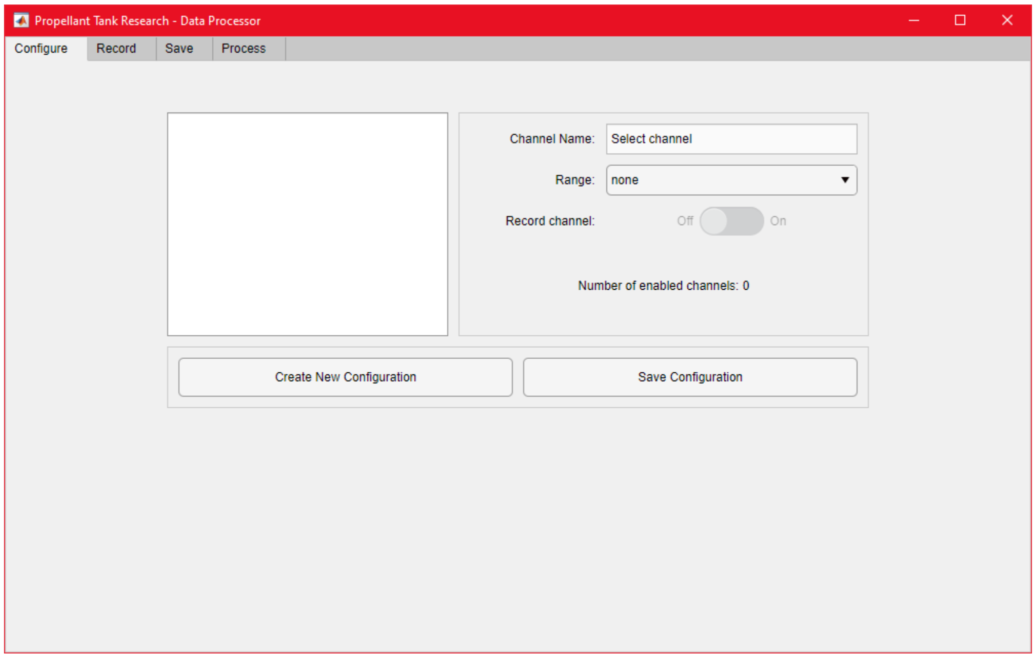


Figure 1.1: Initial screen

To configure channels

1. Click the **Create New Configuration** button.
2. Navigate the list of displayed devices and select a channel that you want to configure.
3. Change the display name, range, and terminal configuration of the channel using the right panel.
4. Enable the channel by using the **Record channel** switch.
5. Once all appropriate channels have been configured, click the **Save Configuration** button to finalize channel configuration.

To record data

1. Navigate to the **Record** tab.
2. Click the **Load Configuration File** button, and select the appropriate configuration file.
3. Select at least one channel from the list to live view data. Multiple channels may be displayed by pressing *ctrl* or **shift** while selecting. All enabled channels in the configuration file will be recorded, regardless if they are displayed live.
4. Adjust the live and recorded measrement rates, and the buffer length as necessary. The buffer length determines the duration of data plotted in the live view and does not affect the duration data is recorded.
5. The live view may be enabled by using the switch labeled **Enable live view**.
6. Data may be recorded by pressing the green **Start** button. The red lamp will be illuminated to indicate that data is being saved.
7. When finished, click the red **Stop** button.

To save data

1. After finishing recording, the app will automatically navigate to the **Save** tab.
2. Enter the save folder location, or press **Browse Folder** button and select a directory.
3. Enter in desirable meta data. See section 2.5 for more info.
4. Click the **Save Data** button. A dialog box will indicate success.

2 Usage

2.1 Overview

The Analog Data Recorder app allows the user to record analog data from National Instruments devices via MATLAB. The advantages are that it provides a comfortable and native environment to work with recorded data directly in MATLAB. Additionally, MATLAB enables programmatic control over every element of recording data. Additionally, the source code for the app may be easily modified to accomplish specific tasks that differ from user to user based on the respective requirements.

The operation of the app was designed to be as “plug-and-play” as possible. In that effort, a simple channel configuration tool has been developed inside the app to eliminate the need of typing several complex commands for each channel for each session. Instead, the user simply clicks a switch to add a channel to the recording session. This configuration is then saved as a `.mat` file that may be loaded each time the app is opened. This allows the configuration and validation of instrumentation to be performed ahead of actual test activities in order to identify and resolve any apparent issues with the setup.

Recording data was also designed to be as effortless as possible. A configuration file is simply loaded into the app, then a single parameter is selected (the record rate), and the **Start** button is pressed. The app also allows the user to monitor live data without actually recording it. Once ready, the user simply presses the **Stop** button to end the recording session and save the data.

The saving operation was designed to give additional flexibility and control to the user. The user is able to specify meta data that is saved with each data file. This meta data may include observation notes on the how the trial ran, or information on the testing conditions, for example. The format of the notes can also be saved in a `.mat` file so that it doesn’t need to be retyped during each save.

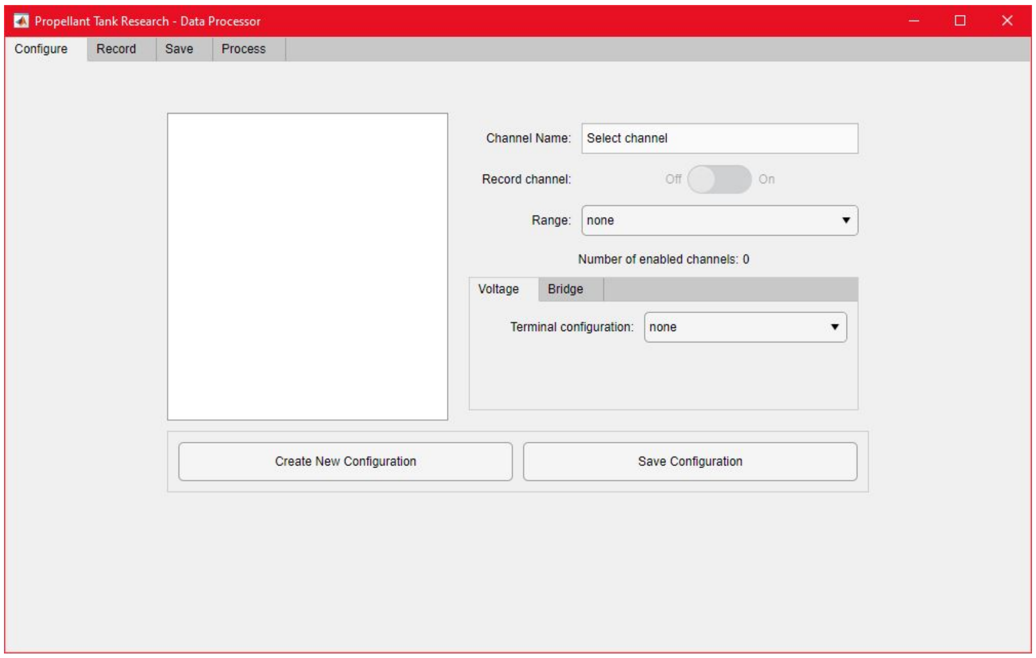
The app also creates an environment for the user to perform processing operations within the app. However, this does require editing the source code. It is, though, already setup to allow the user to select a data file and have it loaded into the workspace. Otherwise, the user may simply copy the code in appendix A to get started with parsing the data and performing actions on it.

2.2 Installation

The Analog Data Recorder app uses a few MATLAB dependencies to operate. The most important is the Data Acquisition Toolbox. The installation of this toolbox is demonstrated in section B.1. Additionally, the National Instruments Hardware Support Package must be installed. Details for this package may also be found in section B.1.

To install the app, navigate to https://github.com/gulanr/adr. To install the app alone, download the file called **Analog Data Recorder.mlappinstall**. Then, run this file to install the app. MATLAB will prompt the user for confirmation to continue. To view and edit the source code, download the entire repository. Editing the source code may be accomplished through MATLAB's App Designer. Do this by typing `appdesigner` into the command window, then by opening the file called **ADR.mlapp** that was downloaded in the repository.

2.3 Channel Configuration

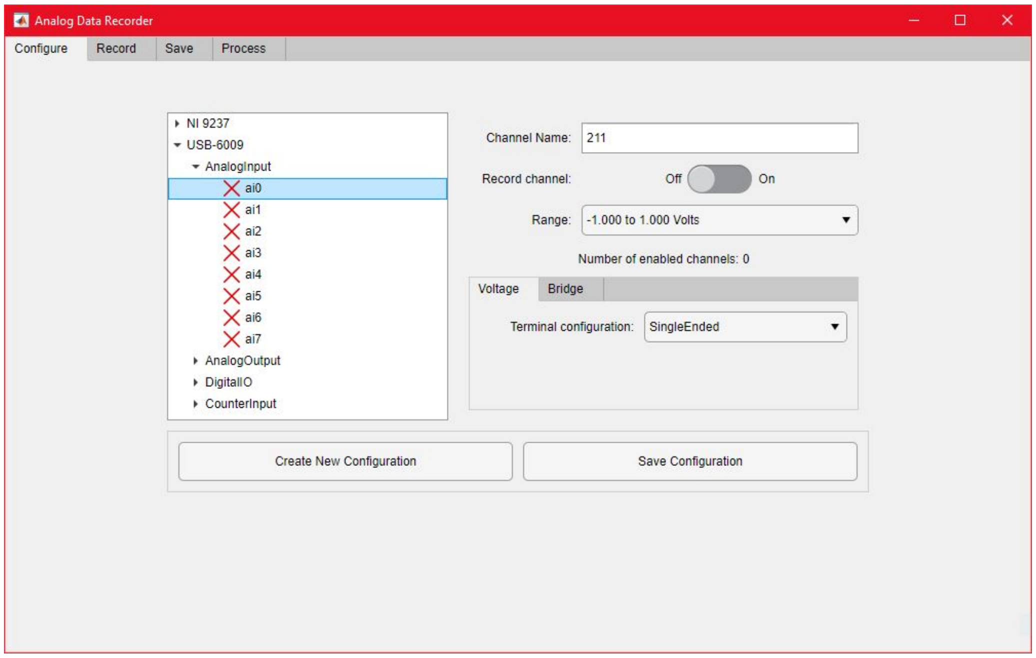


Channel Name: Select channel
Record channel: Off On
Range: none
Number of enabled channels: 0
Terminal configuration: none

Figure 2.1: Channel configuration tab

The Analog Data Recorder is designed to only be compatible with `AnalogInput` type devices. This type of device refers to a specific subsystem type of device. Some devices such as the NI-9237 only have a single subsystem, which is called `AnalogInput`. Other devices, such as the NI-USB-6009 may have as many as four, including `AnalogInput`, `AnalogOutput`, `DigitalIO`, and `CounterInput`.

To configure a device to be recorded, first click the **Create New Configuration** button. This will then list all connected NI devices. To add a channel, expand each device and corresponding `AnalogInput` subsystem. Depending on the device, a number of channels will appear. For `AnalogInput` type channels, they will typically be called `ai0`, `ai1`, `ai2`, etc. When done, the app will look something like this for a `Voltage` measurement type channel:



Channel Name: 211

Record channel: Off On

Range: -1.000 to 1.000 Volts

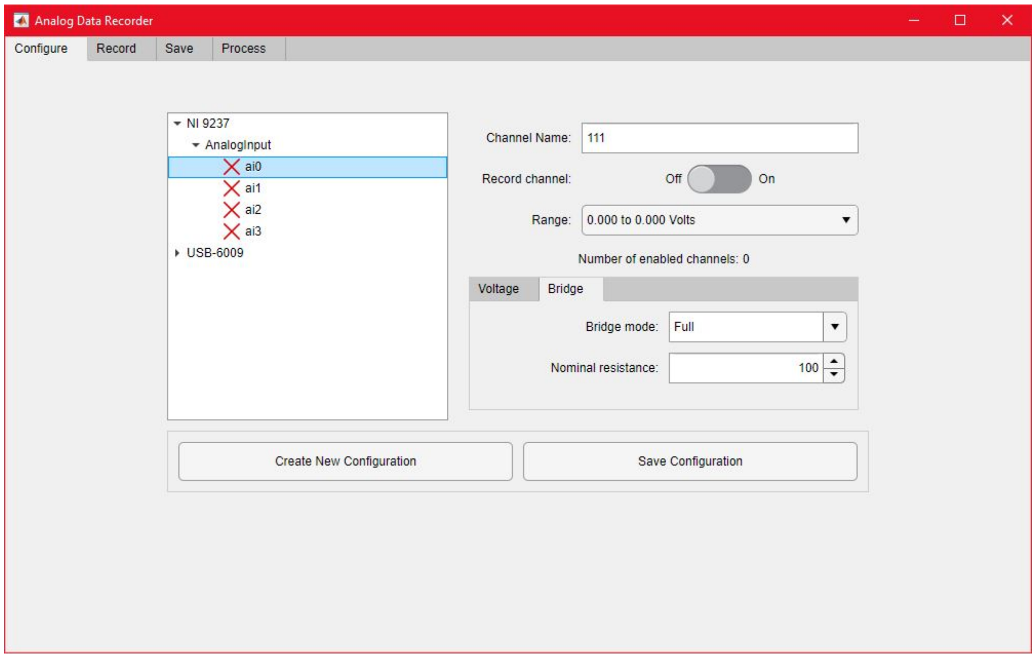
Number of enabled channels: 0

Terminal configuration: SingleEnded

Figure 2.2: Configure Voltage measurement channels

By default, the app will give each channel a unique channel name. The user may edit this to be more descriptive and memorable. The user may also adjust channel specific settings, such as its range and terminal configuration for `Voltage` measurement channels. Once configured, slide the **Record Channel** switch to the “On” position. A green check will appear in the channel list to confirm the channel has been enabled.

The process is similar for other `AnalogInput` devices, such as `Bridge` measurement channels. Here though, the nominal resistance and bridge mode need to be specified instead of the terminal configuration.



Channel Name: 111

Record channel: Off On

Range: 0.000 to 0.000 Volts

Number of enabled channels: 0

Bridge mode: Full

Nominal resistance: 100

Figure 2.3: Configure Bridge

Once all desired channels have been configured, click **Save Configuration** to save the channel setup. This file may then be loaded to quickly setup a DAQ session without having to re-configure each device each time.

2.4 Recording Data

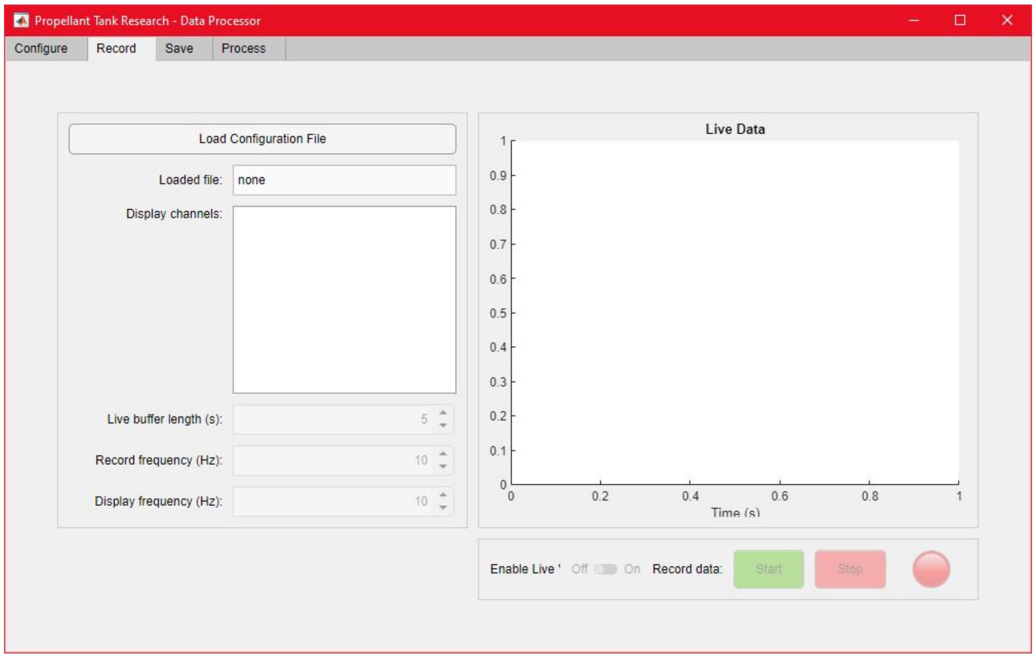


Figure 2.4: Initial record tab

As mentioned before, the process to record data was designed to be very simple. To start, press the **Load Configuration File** button, and select a configuration file that was previously created. A progress bar will then appear while the app is loading and adding the channels to the session. Then, the user must select at least one channel to be plotted in the axes to the right. Multiple channels may be selected by pressing **ctrl** or **shift**.

Then, the user need only configure the record rate and press the **Start** button to begin. The **Stop** button may be pressed after the required data has been collected. In addition, the app also allows the user to monitor the selected channels and plot their data live without recording it. This is accomplished by switching the **Enable Live View** switch to the **On** position. The user may also specify the rate for live monitoring, as well as the duration of data to keep in the plot.

Once the **Stop** button has been pressed, the app will automatically navigate to the **Save** tab.

2.5 Saving Data

The save environment is the third tab.

This environment allows the user to specify unique meta data to save along with the rest of the data. This meta data allows users to record identifying information (besides the date and time of recording) such as trial number, or testing parameters. This information is entered into a text box in the middle of the screen. The formatting of this textbox is important. Information must be entered as:

```
TagName1=Values1
TagName2=Values2
TagName3=Values3
...
```

Here, the elements are:

TagName The name of the tag must be a valid MATLAB variable. So, it can contain alphanumeric characters, but can't start with a number. The tag name could be something like "TestAngle", or "RunNumber".

Equal sign (=) The equal sign separates the tag name from the values of the tag. There must be no spaces on either side of the equal sign.

Values The values portion specifies the data for each tag. The values will be stored as `string` objects, and any space after the equal sign will be included in this string.

It is also important to note that a single "return" must be entered before each new line. Having multiple spaces between each tag name is not allowed. The app will parse the data and prompt the user to confirm that the information is correct before saving.

In the save file, the meta data will be stored in a structure called `META`. Using the example above, `META` would be a structure like this:

```
USRMETA =

  struct with fields:

    TagName1: "Values1"
    TagName2: "Values2"
    TagName3: "Values3"
```

So, when processing the the meta information, it may be necessary to use the `str2double()` functions, unless a `string` object is desirable.

2.6 Processing

The data processing features of the app are left to the user to devise. Because every situation is inherently different, providing comprehensive and meaningful data processing solutions built-in would be difficult to implement. However, an environment is prepared to allow the the user to implement their own solution directly into the app but this requires modifying the source code. This allows the user to directly perform processing activities in the app, such as converting analog measurements to other physical units, or combining multiple data sets into one file. However, a script can just as easily be written to perform these activities outside of the app. This is more ideal if data processing will occur after all testing is finished. That said, it may be useful to perform preliminary processing in order to verify that the most recent test yielded acceptable results, or if that trial should be repeated.

The following section outlines how to access and modify the source code. If processing activities will not be performed in-app, skip to section 2.6.

To modify the source code

The source code for the app can be downloaded from https://github.com/gulanr/adr. To modify the source code, download the entire repository from GitHub. Then in MATLAB, open the App Designer either by clicking **Design App** on the apps or by entering `appdesigner` in the command window. Next, open the file named "ADR.mlapp". Using the **Code View** environment, the source code may be edited.

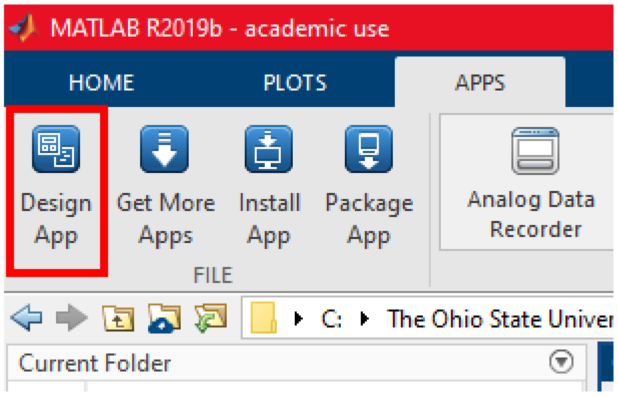


Figure 2.5: Open App Designer

In the source code, the function of interest is named `ProcessData`. It is repeated here for convenience:

```
function ProcessData(app, event)
    % Prompt user to select file
    [f,p] = uigetfile('*.mat');

    % Load the file
    if f == 0 % If the user cancelled before selecting the file
        return
    else % File was selected
        load([p f])
        if ~exist('META') == 1 % If the file does not have a
            variable named 'META'
            uialert(app.Main,'Invalid save file.','Error');
            return
        elseif ~strcmp(META.Type,'DATA') % If .mat file is not
            of type 'DATA'
            uialert(app.Main,'Invalid save file.','Error')
            return
        end

        % Save variables to base workspace
        assignin('base','DATA',DATA)
        assignin('base','USRMETA',USRMETA)
        assignin('base','META',META)
        assignin('base','TIMESTAMPS',TIMESTAMPS)
    end

    % ADD CODE BELOW HERE

end
```

This function acts as a callback function for the **Browse File** button on the **Process** tab, as illustrated in Figure 2.6. Once a data file has been successfully loaded, the code created by the user will be executed. Graphical elements may also be created and manipulated.

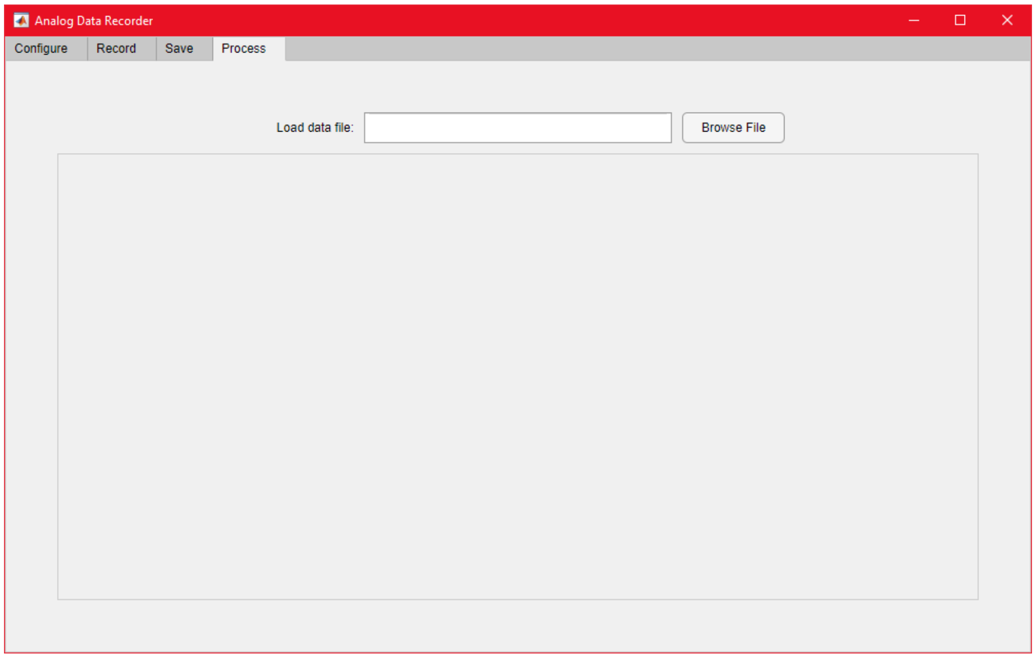


Figure 2.6: Process Tab

To parse the saved data

To load the data from a save file into the workspace, use the syntax

```
>> load(filename)
```

where `filename` is the `.mat` file which contains the data of interest. Once loaded, four variables will appear in the workspace:

DATA The `DATA` variable is an $m \times n$ array, where m is the number of samples and n is the number of channels.

META The `META` variable is a structure which provides identifying and utility information. `META` has the form:

- `Timestamp`: time that test was performed. To convert to datetime, use the `datestr()` function.
- `ChannelOrder`: string array object containing the list of channel names
- `ConfigFile`: filename and path of configuration file.
- `SessionRate`: record frequency in Hz.

USRMETA Provided by user when saving the data. More info, see 2.5.

TIMESTAMPS Timestamps of recorded data as a $m \times 1$ array.

From here, the data is available to process and manipulate as required. Appendix A provides an example which processes measurements taken from pressure transducers.

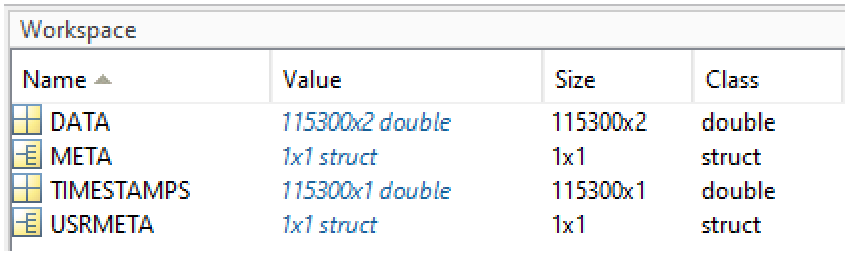
A

Data Processing Example

This example will demonstrate how to parse the saved data files and convert an analog voltage measurement into pressure units. The data file that will be used is called “20200105_203526_S001.mat”. This file was created using a simulated NI-USB-6009 device, though as far as MATLAB can tell, it is the real deal. In the configuration file, two analog input channels were enabled, called “P1” and “P2”. These channels were recorded at 1000 Hz for approximately two minutes. To begin, the data file should be loaded:

```
load("20200105_203526_S001.mat")
```

Once done, four variables will appear in the workspace as shown in Figure A.1.



Workspace

Name	Value	Size	Class
DATA	115300x2 double	115300x2	double
META	1x1 struct	1x1	struct
TIMESTAMPS	115300x1 double	115300x1	double
USRMETA	1x1 struct	1x1	struct

Figure A.1: Loaded variables in the workspace

The data may now be reviewed in order to verify the test by inspecting `META` and `USRMETA`. The `META` variable was automatically generated by ADR and is

```
META =

  struct with fields:

        Timestamp: 7.3780e+05
      SessionRate: 1000
```

```
            Type: 'DATA'
    ChannelOrder: ["P1"    "P2"]
      ConfigFile: 'C:\The Ohio State University\Propellant Tank
         Research - Documents\Phase 2\Testing\MATLAB\
         Pressure_Test.mat'
```

Here, the timestamp can be confirmed by using the `datestr()` command. For example:

```
>> datestr(META.Timestamp)

ans =

    '05-Jan-2020 20:35:26'
```

Additionally, the `USRMETA` data for this case is:

```
USRMETA =

  struct with fields:

          run: '1'
    simulated: 'true'
        notes: 'this data will be used in an example'
        array: '[1 2 3 4]'
    bad_array: '1 2 3'
```

Here, a few tags were created to demonstrate the parsing process. By default, the values are stored as `char` objects. To convert to the intended variable class, the `eval()` function may be used. For example, to convert the `run` field from `char` to `double`, do

```
>> eval(USRMETA.run)

ans =

     1
```

or to convert the `simulated` field from `char` to `logical`, do

```
>> eval(USRMETA.simulated)

ans =

  logical

   1
```

This can be repeated on the other variables, however, the contents of the `char` object must be a valid MATLAB expression. For example, `eval(USRMETA.array)` will return a successful result, but `eval(USRMETA.bad_array)` will not.

Now, the data will be processed. Before this, the calibration of the pressure transducer will be considered. In this case, `P1` has the following calibration:

Pressure	Unit Data
PSIA	Vdc
0.00	0.002
150.00	2.501
300.00	5.003
150.00	2.501
0.00	0.001

and `P2` has this calibration:

Pressure	Unit Data
PSIA	Vdc
0.00	0.002
150.00	2.501
300.00	5.004
150.00	2.501
0.00	0.002

With this data, the goal is to create a linear trend line in order to correlate the unit data with the pressure. There are a variety of ways to accomplish this, but here, the MATLAB function `fit()` will be used.

```
P1.CalP = [0 150 300 150 0]';
P1.CalV = [0.002 2.501 5.003 2.501 0.001]';

P2.CalP = P1.CalP;
P2.CalV = [0.002 2.501 5.004 2.501 0.002]';

P1.fit = fit(P1.CalV,P1.CalP,'poly1');
P2.fit = fit(P2.CalV,P2.CalP,'poly1');
```

Then, the `fit` object for pressure transducer `P1` is

```
>> P1.fit

ans =

     Linear model Poly1:
     ans(x) = p1*x + p2
     Coefficients (with .95 confidence bounds):
       p1 =       59.99  (59.95, 60.03)
       p2 =    -0.06854  (-0.1777, 0.04058)
```

Then, the voltage data may be converted to pressure measurements by doing the following:

```
P1.Voltage = DATA(:,1);
P2.Voltage = DATA(:,2);

P1.Pressure = P1.Voltage*P1.fit.p1 + P1.fit.p2;
P2.Pressure = P2.Voltage*P2.fit.p1 + P2.fit.p2;
```

The plotted data is shown in Figure A.2. Visual inspection suggests the calibration of the pressure transducers are correct because the pressure measurements match the trend of the voltage measurements.

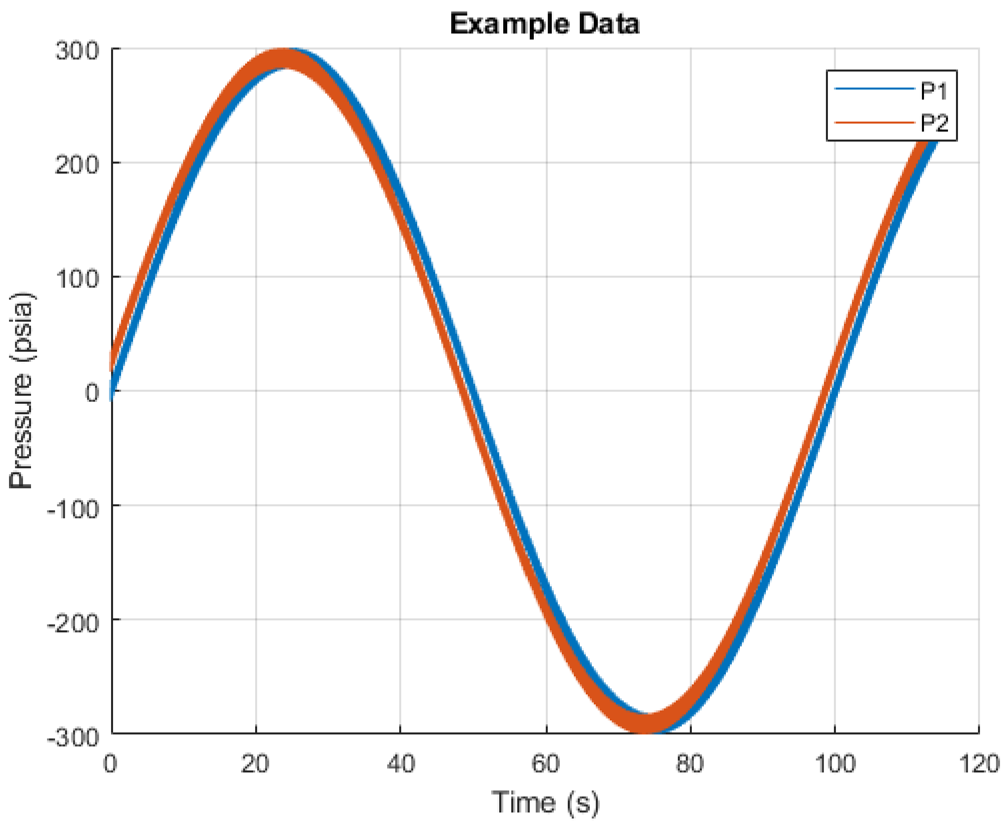


Figure A.2: Example data

The final script to acquire the above plot is then

```
load("20200105_203526_S001.mat")

P1.CalP = [0 150 300 150 0]';
P1.CalV = [0.002 2.501 5.003 2.501 0.001]';
```

```
P2.CalP = P1.CalP;
P2.CalV = [0.002 2.501 5.004 2.501 0.002]';

P1.fit = fit(P1.CalV,P1.CalP,'poly1');
P2.fit = fit(P2.CalV,P2.CalP,'poly1');

P1.Voltage = DATA(:,1);
P2.Voltage = DATA(:,2);

P1.Pressure = P1.Voltage*P1.fit.p1 + P1.fit.p2;
P2.Pressure = P2.Voltage*P2.fit.p1 + P2.fit.p2;

figure;
hold on
plot(TIMESTAMPS,P1.Pressure)
plot(TIMESTAMPS,P2.Pressure)
xlabel('Time (s)')
ylabel('Pressure (psia)')
legend("P1","P2")
grid on
title('Example Data')
```

B

Data Acquisition Toolbox Tutorial

The Data Acquisition Toolbox is a Mathworks created toolbox which enables the interface between MATLAB and data acquisition hardware. This sections demonstrates how to get started with the DAT and how to create a minimum working example. After reading, the user will be able to acquire data from MATLAB's command window.

B.1 Instalation

In order to use the Analog Data Recorder app, additional software is necessary. The Data Acquisition Toolbox is the first one. It can be installed in MATLAB by pressing on the button outlined in Figure B.1

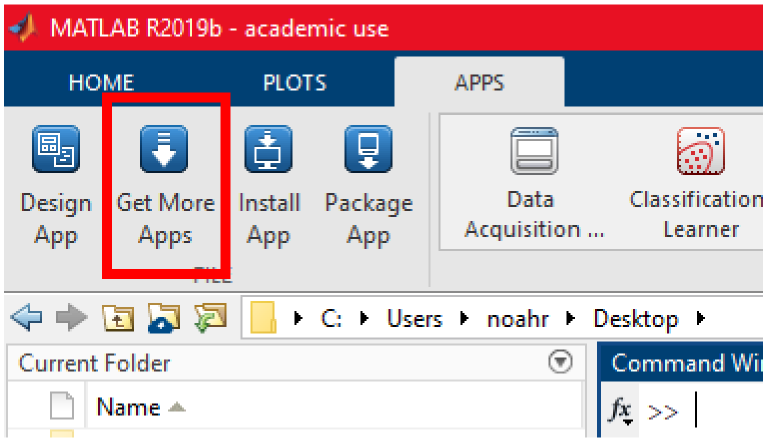
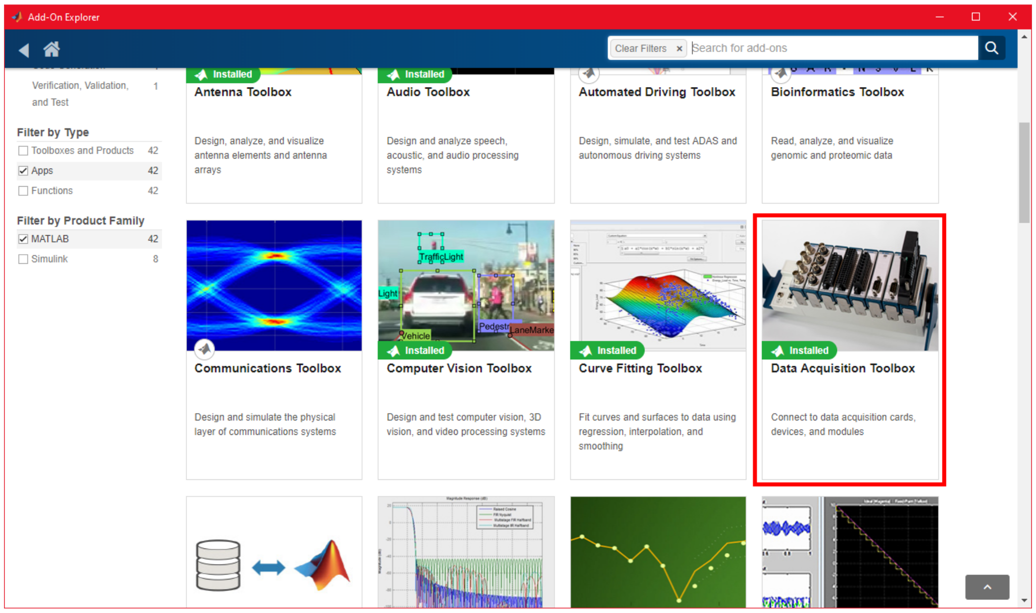


Figure B.1: Get more apps!

Then either searching for the toolbox or filtering by Mathworks authored apps allows one to quickly find the Data Acquisition Toolbox. Just follow the prompts to install it:


Add-On Explorer

Clear Filters × Search for add-ons

Verification, Validation, and Test 1

Filter by Type
- Toolboxes and Products 42
- [✓] Apps 42
- Functions 42

Filter by Product Family
- [✓] MATLAB 42
- Simulink 8

Installed
Antenna Toolbox
Design, analyze, and visualize antenna elements and antenna arrays

Installed
Audio Toolbox
Design and analyze speech, acoustic, and audio processing systems

Automated Driving Toolbox
Design, simulate, and test ADAS and autonomous driving systems

Bioinformatics Toolbox
Read, analyze, and visualize genomic and proteomic data

Communications Toolbox
Design and simulate the physical layer of communications systems

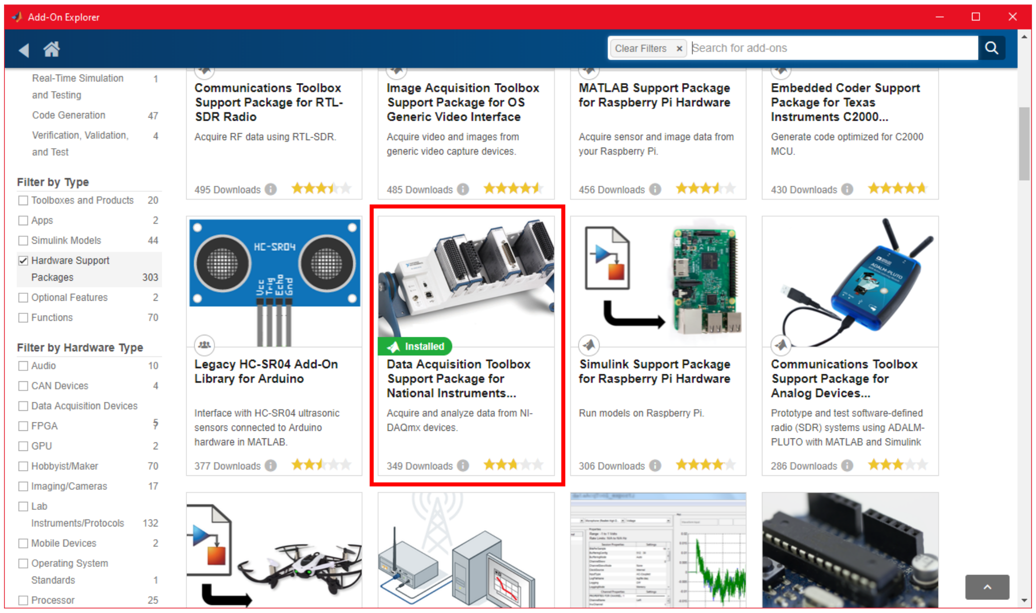
Installed
Computer Vision Toolbox
Design and test computer vision, 3D vision, and video processing systems

Installed
Curve Fitting Toolbox
Fit curves and surfaces to data using regression, interpolation, and smoothing

Installed
Data Acquisition Toolbox
Connect to data acquisition cards, devices, and modules

Figure B.2: Data Acquisition Toolbox search

Next, the appropriate Hardware Support Package (HSP) needs to be installed. For each hardware vendor, there is a corresponding driver package. For National Instruments (NI) it looks like this:



Communications Toolbox Support Package for RTL-SDR Radio
Acquire RF data using RTL-SDR.
495 Downloads

Image Acquisition Toolbox Support Package for OS Generic Video Interface
Acquire video and images from generic video capture devices.
485 Downloads

MATLAB Support Package for Raspberry Pi Hardware
Acquire sensor and image data from your Raspberry Pi.
456 Downloads

Embedded Coder Support Package for Texas Instruments C2000...
Generate code optimized for C2000 MCU.
430 Downloads

Legacy HC-SR04 Add-On Library for Arduino
Interface with HC-SR04 ultrasonic sensors connected to Arduino hardware in MATLAB.
377 Downloads

Data Acquisition Toolbox Support Package for National Instruments...
Acquire and analyze data from NI-DAQmx devices.
349 Downloads

Simulink Support Package for Raspberry Pi Hardware
Run models on Raspberry Pi.
306 Downloads

Communications Toolbox Support Package for Analog Devices...
Prototype and test software-defined radio (SDR) systems using ADALM-PLUTO with MATLAB and Simulink
286 Downloads

Figure B.3: National Instruments HSP

The National Instruments package can be found by either searching for filtering in the left hand menu. Follow the prompts to fully install the package - additional 3rd party NI drivers will be installed automatically. This process may take a few minutes, but once completed, you're ready to get working with MATLAB.

In MATLAB, you can check that everything is working soundly by typing:

```
>> daq.getVendors()
```

if the HSP is installed correctly, you should see something like this:

```
ans =

Number of vendors: 2

index     ID        Operational          Comment
----- ----------- ----------- ------------------------
1      ni          true         National Instruments(TM)
2      directsound true         DirectSound

Properties, Methods, Events

Additional data acquisition vendors may be available as
    downloadable support packages.
Open the Support Package Installer to install additional vendors
    .
```

As you can see, in this instance of MATLAB, two HSP's are installed; one for NI, and another for `directsound` (which enables recording and creation of audio data). Of other importance is that the `Operational` property is set to `true`.

B.2 Setup & Configuration

Now, we can make sure that all of our devices are properly talking to MATLAB. This is accomplished with the following command:

```
>> D = daq.getDevices()
```

As an output, MATLAB will either list every available device (if multiple are connected) or provide details on a single device. `D` is now a `device` object. The output for the previous command may be:

```
>> D = daq.getDevices()

D =

Data acquisition devices:

index   Vendor     Device ID                           Description
----- ----------- ---------
   ------------------------------------------------------------
1     ni          cDAQ2Mod1  National Instruments(TM) NI 9237
2     ni          Dev1       National Instruments(TM) USB-6009
```

As can be seen, two devices were connected at the time. One device was the NI USB-6009 which is a general purpose analog/digital IO device. It has 8 analog input channels, 2 analog output channels, 12 digital IO channels, and a single counter input. Other pertinent information such as the rate and measurement types are listed. Now, let's dig in to what all properties the `device` object has.

The full properties associated with `device` objects can be seen by clicking on the link in the previous output, or by inspecting the variable in the workspace. Of interest to us are the following properties:

- `ID` This is the device ID that MATLAB will refer to. For the USB-6009, the ID is `'Dev1'`.
- `Subsystems` This is a `struct` which contains even more good information:
  - `RangesAvailable` These are the available ranges for the instrument. Typically, calibration errors are provided based on the full scale, so by setting the range to the minimum possible, the error is reduced.
  - `RateLimit` This is the measurement frequency limits in Hertz.
  - `SubsystemType` This represents the *type* of device it is (analog input, analog output, audio, etc.).

  - `ChannelNames` This provides the actual name of each channel (especially useful when the device has many channels).

Of course, in order to become a master at the DAT, one should dive in and investigate each and every property. But, moving on to setting up the session. A DAQ session allocates resources and sets up the data acquisition environment. It can be created with command

```
>> S = daq.createSession('ni')
```

where `'ni'` is the ID of the vendor. By default the settings of the session are configured as

```
                      AutoSyncDSA: false
                    NumberOfScans: 1000
                DurationInSeconds: 1
                             Rate: 1000
                     IsContinuous: false
   NotifyWhenDataAvailableExceeds: 100
IsNotifyWhenDataAvailableExceedsAuto: true
       NotifyWhenScansQueuedBelow: 500
   IsNotifyWhenScansQueuedBelowAuto: true
           ExternalTriggerTimeout: 10
                   TriggersPerRun: 1
                         UserData: ''
                           Vendor: National Instruments(TM)
                         Channels: ''
                      Connections: ''
                        IsRunning: false
                        IsLogging: false
                           IsDone: false
      IsWaitingForExternalTrigger: false
                TriggersRemaining: 1
                        RateLimit: ''
                      ScansQueued: 0
            ScansOutputByHardware: 0
                    ScansAcquired: 0
```

Here, the properties of the `Session` object are pretty self-explanatory. The ones that are most interesting are `DurationiInSeconds`, `Rate`, and `Channels`. Of course, all of them are useful and necessary but these are the ones we will mess with as an introduction to the DAT. For example, we can set the `DurationInSeconds` to `5` with

```
>> S.DurationInSeconds = 5
```

and the measurement frequency with

```
>> S.Rate = 10
```

Now, our session would say something like this:

```
S =
Data acquisition session using National Instruments(TM) hardware
    : Will run for 5 seconds (50 scans) at 10 scans/second. No
    channels have been added.
```

As you can see, initially there are no channels added to the session. Let's say we had a pressure transducer plugged into channel `'ai0'` of our USB-6009 that we wanted to measure. We could use the `addAnalogInputChannel()` function to add this channel to the session. Now, when we eventually record data, MATLAB will measure this channel. For example:

```
>> addAnalogInputChannel(S,'Dev1','ai0','Voltage')

ans =

    'Data acquisition session using National Instruments(TM)
        hardware:
         Will run for 5 seconds (50 scans) at 10 scans/second.
         Number of channels: 1
            index Type Device Channel MeasurementType      Range
                       Name
            ----- ---- ------ ------- ----------------
                ---------------- ----
            1      ai   Dev1   ai0     Voltage (Diff)  -20 to +20
                Volts

     Methods, Events .'
```

Detailed information about this command is provided in MATLAB's documentation.

B.3 Data Collection

Once the channel is added, you are ready to begin recording data. The DAT has two ways of recording data:

Foreground Records data for `DurationInSeconds` at `Rate` samples per second. Other commands cannot be used while data is being recorded. Usefule when the duration length is known ahead of time.

Background Continuously records data and creates an event every time there is new data collected. Does not interfere with using MATLAB. Useful when the record duration is unknown.

Arguably, the foreground data collect is much easier; simply use the command:

```
>> Data = startForeground(S)
```

and Data will then be a $m \times n$ array of measurements, where m is the number of samples collected (DurationInSeconds×Rate) and n is the number of channels attached to the session.

Background data collection allows MATLAB to continuously collect data and have the user specify the treatment of the data (discard, save, use as controller input, etc.). To do this, two commands are neccessary to begin:

```
>> S.IsContinuous = true;
>> startBackground(S)
```

In order to access the data being collected, the user must create an event listener of the 'DataAvailable' type. The documentation for this function provides great detail. In the example of interest, an anonymous callback function will be created which calls a function to handle the data. This function may, for example, save the data to a mat file, plot the data, or perform some kind of analysis. Now, let's create an example:

```
fig = figure;
ax = axes;
plot(ax,NaN,NaN);
Data = [];
TimeStamps = [];

DataAvailableListener = addlistener(S,'DataAvailable', ...
    @(src,event) dataAvailableCallback(src,event));

function dataAvailableCallback(src,event)
    TimeStamps = [TimeStamps; event.TimeStamps];
    Data = [Data; event.Data];
    set(ax,'XData',TimeStamps,'YData',Data);
end
```

These few lines simply take the data from the event listener, append it to the end of the complete data, and then plot it. Of course, depending on the duration and measurement frequency, very quickly the Data and TimeStamps arrays may become quite large. To mitigate this, the user may specify the "buffer" size of the arrays, then add the new data to the beginning of the array and discard the elements whose indices are greater than the buffer. Note that the listener does not trigger every time new data is available. By default, the event occurs once one tenth of a second worth of data has been collected. Hence, the "refresh" rate of the callback function is 10 Hz by default. Google NotifyWhenDataAvailableExceeds to learn more.

C

Troubleshooting

As mentioned previously, the software is delivered as-is and has no warranty. If there are technical issues (i.e., bugs), the authors make no guarantee of fixing them. Instead, here are some tips on fixing commonly encountered issues:

C.1 General Tips

The Analog Data Recorder app was created and tested using version R2019b of MATLAB. While it is expected that the app works in future versions, it is not guaranteed.

If there are any issues encountered using the software, try creating a session

C.2 Reference Documentation

MATLAB references:

- Analog input tutorial
- Acquire bridge measurements
- Data available
- Data Acquisition Toolbox

C.3 Specific Error Messages

Vendor is not recognized

Sometimes, when initially installing the National Instruments Hardware Support Package, MATLAB may not recognize NI as being an operational vendor when creating a session. If this error is encountered, in the command window enter

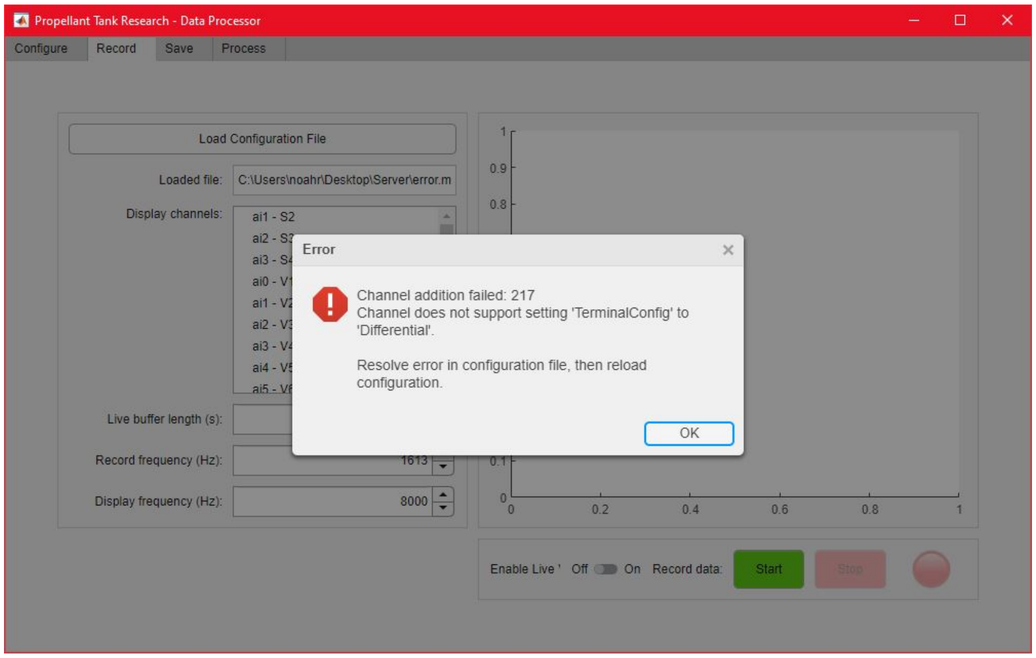
```
V = daq.getVendors()
```

Check to make sure the NI vendor has `Operational` as `true`. If so, next try creating a session with

```
S = daq.createSession('ni')
```

If a similar error message is encountered, uninstall and reinstall the HSP via MATLAB. If that doesn't work, restart the computer and ensure the NI Device Monitor is running.

Channel does not support setting 'TerminalConfig' to 'Differential'



Error

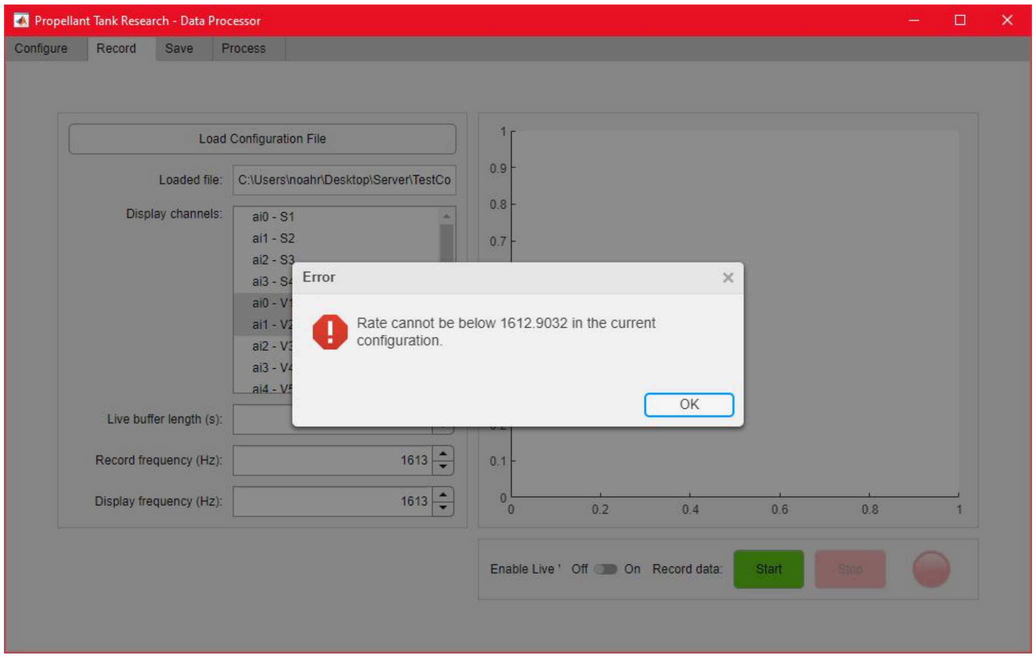
Channel addition failed: 217
Channel does not support setting 'TerminalConfig' to 'Differential'.

Resolve error in configuration file, then reload configuration.

Figure C.1: Channel terminal configuration error

To resolve this error, change the `TerminalConfig` to `SingleEnded` for the problematic channel. This error occurs with specific devices such as the NI USB-6009. Interestingly, NI configures this device's `AnalogInput` subsystem for all analog input channels, even though not all of these channels behave the same. According to the manual for the device, not all channels may be configured as `Differential`, even if the option is available in the DAT. More information in the manual, on page 10.

Rate cannot be below 1612.9032 in the current configuration



Loaded file: C:\Users\noahr\Desktop\Server\TestCo

Record frequency (Hz): 1613

Display frequency (Hz): 1613

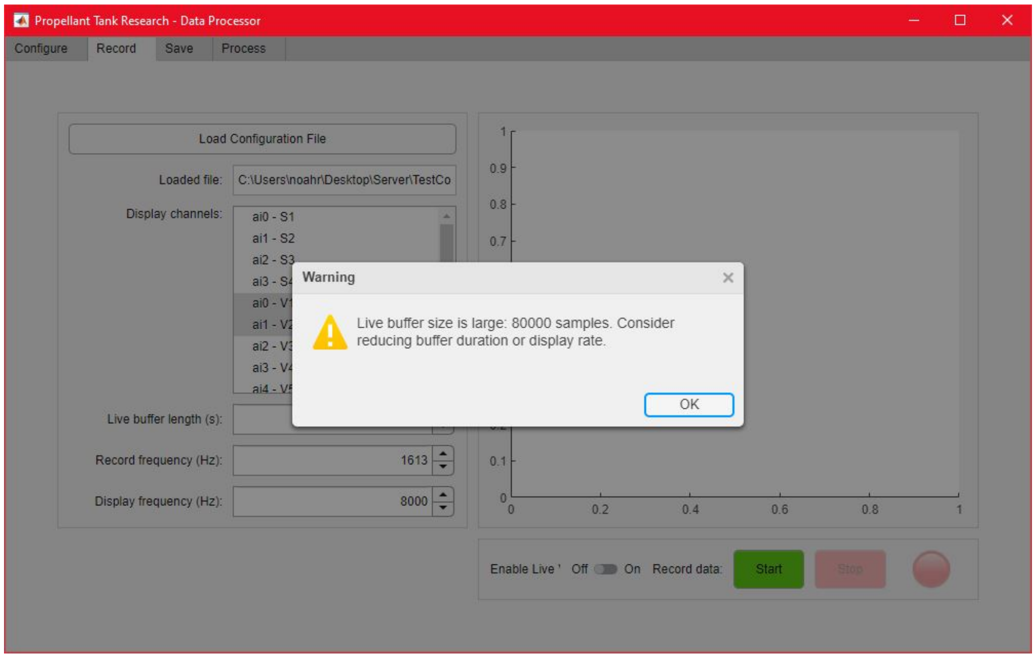
Rate cannot be below 1612.9032 in the current configuration.

Figure C.2: Rate error message

For some devices, only certain discrete rates are permitted. For example, the lowest rate allowed by the NI-9237 strain gage module is 1612.9032 Hz. However, the NI-USB-6009 allows for a more fine control over the rate. Because the session must record at a single rate for all devices, the NI-9237 will limit the NI-USB-6009 to certain allowable rates.

When this error is encountered, the app will automatically select the next higher allowable rate.

Warning: Large buffer size



Warning

Live buffer size is large: 80000 samples. Consider reducing buffer duration or display rate.

Loaded file: C:\Users\noahr\Desktop\Server\TestCo

Record frequency (Hz): 1613

Display frequency (Hz): 8000

Figure C.3: Large buffer size warning

When plotting live data, the app only save the n most recent measurements, where n is the buffer length (in seconds) multiplied by the display rate. If the buffer is larger than 1000 samples, then performance of the app may be impacted. Typically, MATLAB delivers data to the event listener 10 times per second, so the size of each event will be large. As a result, calculations to manipulate, store, and plot the data may take longer than 0.1 seconds, and the app may appear to lag.

Acronyms

AAE Aeronautical and Astronautical Engineering. iii

ADR Analog Data Recorder. iii, iv, 1, 3, 5, 6, 15, 21, 29

DAQ Data Acquisition. 8, 25

DAT Data Acquisition Toolbox. 6, 21, 25, 26, 30

HSP Hardware Support Package. 6, 22–24, 29, 30

IO Input/Output. 24

NI National Instruments. iv, 1, 5–7, 22, 24, 29, 30

OSU Ohio State University. iii

PTR Propellant Tank Research. iii