Summer Student Report
Mallard: DAQ for CRIS

Kieran R Campbell
Cern Summer Student 2013
kieranrcampbell@gmail.com

August 14, 2013

Contents

1 Online Resources

Github Repository: https://github.com/kieranrcampbell/mallard

Documentation: http://kieranrcampbell.github.io/mallard/

Sourceforge page: https://sourceforge.net/projects/mallardforcris/

2 Introduction

2.1 Etymology

Two things frequently occur at CERN: everything is an acronym (CERN, LHC, ATLAS, ALICE, ISOLDE, CRIS, ...)[1], and the DAQ/duck joke is made (a yellow rubber duck sits on the data acquisition desk in the ATLAS control room). The obvious challenge then for any self respecting summer student is to combine both of these, and so came Mallard: Multi Analog LossLess Acquistion of Resonance Data[2].

2.2 Purpose

Mallard is designed for efficient data acquisition and experimental control on the CRIS experiment at ISOLDE, CERN. The software interfaces a National Instruments USB-6211 card that connects to the lasers, counters and trigger. The experiment requires the software to scan across a range of voltages set on an analogue output, and at each voltage measure counts from the MCP (Micro-Channel Plate) and an analogue input (relative intensity of the laser). The user selects the voltage range across which to scan, how many volts should be set in a given interval, and how many scans across the full voltage range should be performed. The timing of the voltage change comes from an external trigger that is also connected to the scanning laser.

During the data acquisition the software continuously graphs the current count rate and analogue input voltage measured, as well as the average so far in order to provide the user with an idea of whether the measurements are drifting from the expected value. The raw data of counts and analogue input measurements is collected and saved to disk, as well as an integrated file showing the measurements at each voltage averaged over a number of scans.

3 Installation

3.1 Dependencies

Mallard has the following dependencies:

- SciPy http://www.scipy.org/
- PyDAQmx http://pythonhosted.org/PyDAQmx/
- wxPython http://www.wxpython.org/

Installing SciPy on Windows can be tedious, so use Anaconda (https://store.continuum.io/) which comes with it pre-installed. Don't install wxPython using `easy_install` or `pip` as this installs some bizarre empty package. Instead, download the windows installer directly from their website (see above).

3.2 NIDAQmx

NIDAQmx (http://www.ni.com/dataacquisition/nidaqmx) from National Instruments provides the base set of drivers to interface the card. The full set of functionality is only available on Windows. At time of writing the latest version was 9.7 which can be downloaded from http://joule.ni.com/nidu/

[1] There are even super-acronyms such as CHIS - CERN Health Insurance Scheme - a service with which most summer students are all too familiar

[2] Many thanks to various other summer students for help on the acronym

cds/view/p/id/3811/lang/en. NIDAQmx installs a header file containing all the C library function definitions. The location of this file must be noted and an installation file in PyDAQmx modified accordingly (see PyDAQmx documentation for details).

3.3 Mallard

3.3.1 From source

The source code can be downloaded (as zip or tar.gz) from the github repository https://github.com/kieranrcampbell/mallard. Extract it to a convenient folder then, from the top level `mallard` directory it can be installed by `python setup.py install`. Then mallard can be launched in the usual way, or if you prefer to use it within a python script itself it can be called using `import mallard` etc.

3.3.2 Windows executable

A windows installer for the python package can be downloaded from the sourceforge page https://sourceforge.net/projects/mallardforcris/ which will install it as a python package.

3.3.3 System Requirements

Due to the availability of `NI-DAQmx` Mallard only runs on windows. Despite all data being held in RAM, during system tests it was found that 500 scans with 50 intervals per scan only takes up about 150MB, so this is certainly not a limiting factor.

4 Use

4.1 Basic Capture

To begin Mallard, navigate to the root `mallard` folder, and start by

```
python -m mallard.main
```

Due to a multiprocessing bug under python on windows we can't have the usual __main__.py folder to begin the module as `python mallard`.

Each acquisition session of data corresponding to a particular set of settings is known as a capture. Each capture is represented by a tab in the GUI, and a new capture can be opened by File → New Capture. Captures can be saved (see below), but cannot be reloaded into Mallard as there is currently no data analysis functionality. However, the settings for a new capture can be loaded from an old capture by File → Load Capture then selecting the capture you wish to open.

The capture begins by pressing 'Start Capture', and can be stopped by selecting Capture → Kill Capture. The current voltage and MCP count is displayed on the screen, as well as the scan average superimposed on the graph. The graph can be saved at any time using the Matplotlib functionality on screen.

The graph style can be changed at any time by going to File → Preferences. The graph style can be set separately for the count graph and for the voltage graph. There are three different types:

1. Step (histogram): Essentially bins the data exactly like a histogram
2. Line: joins consecutive data points using lines
3. Points: Each data point is represented by a separate point. For the count graph this has error bars, which are given by $\sqrt{N}$ for N counts.

In these preferences you can also set the style of the 'mean'. Selecting 'normalised' graphs each count divided by the number of scans, whereas 'cumulative' gives the sum of all counts across all scans.

Plots can be saved using the controls on the plot itself. To move the plot around, select the 'Pan' symbol and drag with the left mouse click. To zoom in and out, both horizontally and vertically, select the 'Pan' symbol and drag with the right mouse button.

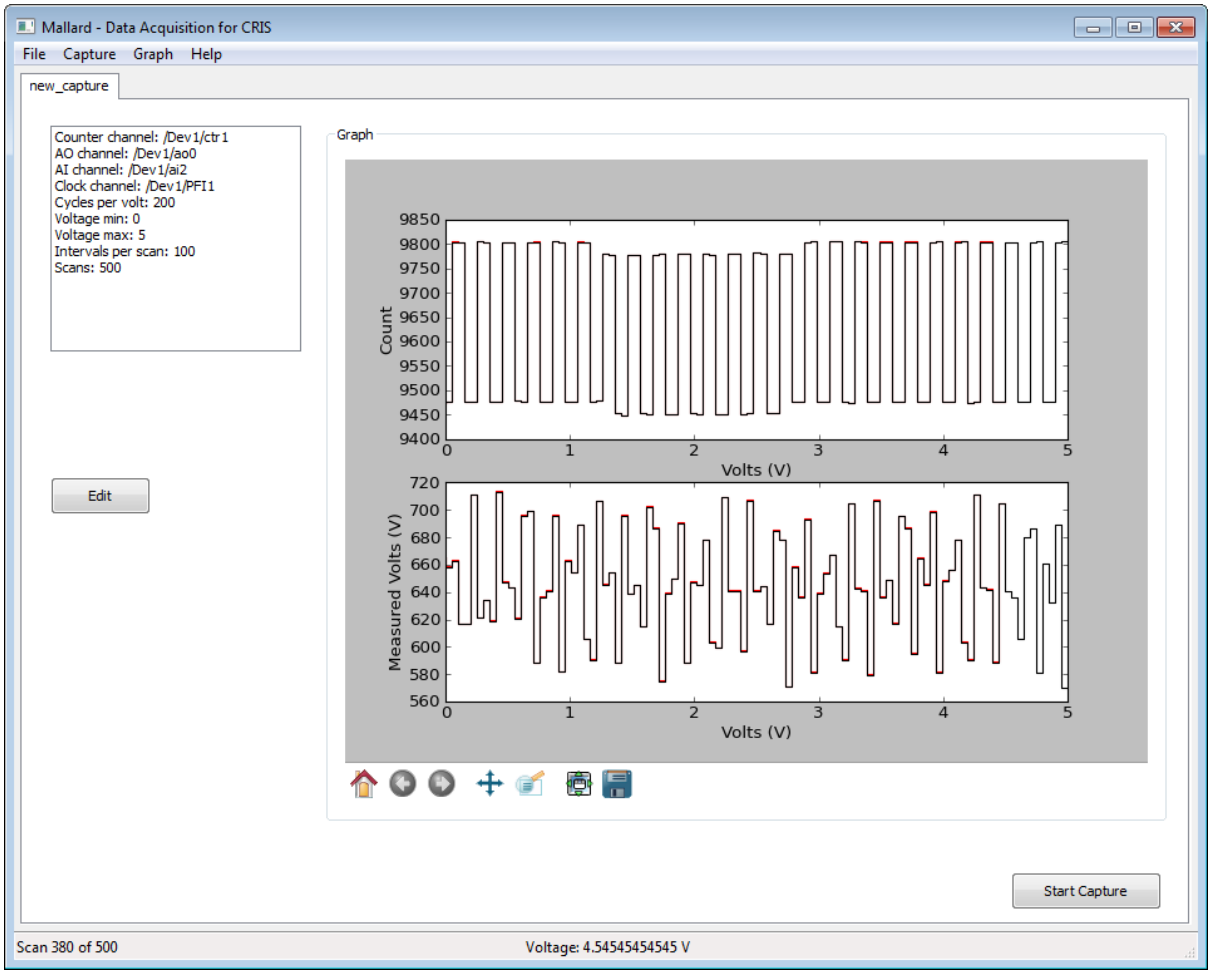

Figure 1: Screenshot of Mallard capturing data.

4.2 Key Concepts

- Voltage minimum: The voltage from which the laser starts scanning
- Voltage maximum: The voltage to which the laser will scan
- Intervals per scan: The number of intervals to scan in the voltage range, defined by $(V_{max} - V_{min})/(N-1)$ for N intervals
- Scans: The number of full scans to do (from minimum to maximum across the range)
- Clock cycles per interval: The number of onboard clock cycles to wait per scan (known as the 'rate' in NI terminology, 100 is a good number)
- Analog input: Analog input channel (default /Dev1/ai2)
- Counter input: Input channel for counting events. This is a 'virtual' channel, which requires looking at the NI handbook, but Dev1/ctr1 corresponds to Dev1/PFI1 on the 6211
- Analog output: Analog output channel (default /Dev1/ao0)
- Clock input: Input channel for the pseudo-trigger clock

Note: if we scan from V_{min} to V_{max} in N intervals, then the volts per interval is $(V_{max} - V_{min})/(N-1)$, **not** $(V_{max} - V_{min})/N$, to accommodate the fact that we would like to measure at the voltage endpoints.

4.3 Data Output

When mallard runs it creates two datasets: every count measurement at every interval across every scan, and likewise for the analogue input readings. These are then averaged over the scans to give a third 'integrated' dataset. As such, mallard outputs 3 data files. If the capture name is 'mycapture' then these files are:

- mycapture.integrated.csv: contains the integrated (final) dataset across all scans
- mycapture.raw.counts.csv: contains the raw count data for all intervals and scans
- mycapture.raw.ai.csv: contains the raw analog input information for all intervals and scans

Each of these files is in comma separated value format and contains a header, denoted by '#' at the beginning of each line, containing the settings listed in key concepts above. As such, any file can be used to recover capture settings for a new capture. The header format comes from numpy's file output functions, as any line starting with '#' is ignored as a comment, so the data can be read in again by python.

Mallard does not yet have any in built analysis, but with the data in this format it should be easy enough. The data can be opened directly in excel, or using numpy in python as follows:

```
import numpy as np
data = np.loadtxt('mycapture.integrated.csv', delimiter=',')
voltage_intervals = data.T[0] # need to transpose matrix
counts = data.T[1]
ai_readings = data.T[2]
```

and voltage, counts and ai_voltage can be analysed using any of the standard tools in SciPy.

5 Development

Mallard is written in python and is under git source control.

5.1 Package Layout

The mallard package is organised into several sub-packages:

5.1.1 mallard.core

This provides base functionality for the package including data and file handling. `CaptureSession` represents an abstract (ie has no attached daq interface or gui), and manages calls for file management and data acquisition. `SessionSettings` represents the settings for a given capture session, including input, output and counter channels. `CaptureSession` holds the 'master' copy, though since everything in python is a pointer all copies (held by all `mallard.core` modules). However, after settings are changed some things must be recalculated (such as voltage intervals), so if a module uses `SessionSettings` it's not okay to assume changes elsewhere will be safely reflected in the current location. `DataManager` holds runtime data and has the `queue.get()` method to receive data from the interface. It also calculates the averages across scans, and calls the `GraphManager` (which provides the interface between `mallard.core` and the gui package). FileManager deals with file input and output. It can generate the header from `SessionSettings` and can read in and parse settings and filenames.

5.1.2 mallard.daq

This contains only one module called `Acquire` that interfaces the USB-6211 card. If the card needs changed in future then only this module needs rewritten. It is the only module in the entire package that is just a function, as spawning processes that are class methods using the `multiprocessing` module is very difficult (maybe impossible?). All this module needs are the session settings (for channel names and rates) and a `multiprocessing.Queue` to report the data back to `DataManager`.

5.1.3 mallard.gui

This provides the graphical user interface using wxPython. `MFrame` is the base `wx.Frame` used, which owns a `CaptureNotebook`. This `CaptureNotebook` holds `CapturePanes`, each of which corresponds to a `CaptureSession`. The `GraphManager` provides a link between the `DataManager` and the gui, updating the graph on screen with the data coming in. `SettingsDialog` simply displays a dialog that lets the user change all the different settings.

5.1.4 mallard.test

Holds all the obsolete and test scripts created along the way, in case they come in handy at a later point. May be changed to 'old' in future.

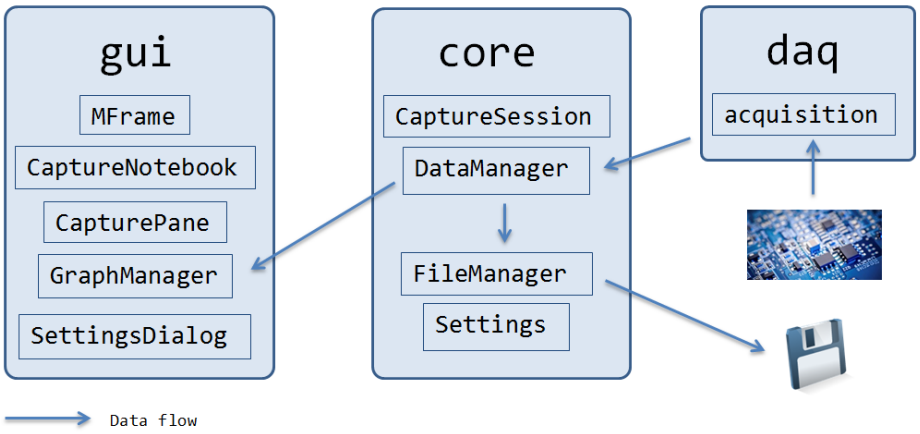

Figure 2: Diagram of data passing between modules.

5.2 Triggering

In order to change the analogue input and count particles in each voltage interval, mallard uses a special type of triggering called *re-triggering*. However, virtually no cards (including the NI-6211) support re-triggering. To get round this we simply use an external clock as a sort of pseudo-trigger, whereby each measurement is performed on the clock cycle. In practice there seems to be absolutely no difference between a trigger and an external clock.

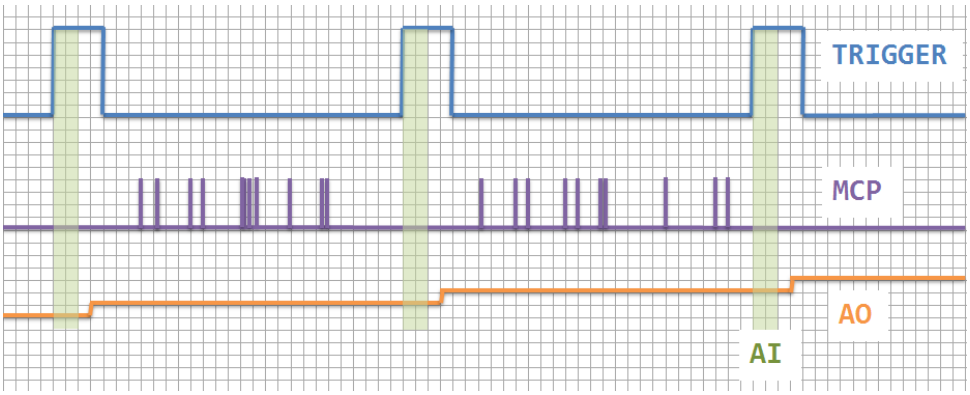

Figure 3: Trigger timing.

On the 6211 certain operations we need can't use an external clock, such as externally timed analogue output. To get round this we slot the analogue output voltage update just after the digital count read (which *can* be externally timed), and in doing so we create virtually externally timed analogue output. Note that the time this software loop takes to update forms the minimum trigger rate, which is found

to be around 40ms. If the CPU is particularly loaded (or a different machine is used) then this may increase.

5.3 Data Handling

6 Troubleshooting

6.1 Common Errors

ImportError: Matplotlib backend_wx and backend_wxagg require wxversion, which was not found.

Sometimes the wxversion module is missing. Download it from http://svn.wxwidgets.org/viewvc/wx/wxPython/trunk/wxversion/wxversion.py?view=co and install in your `site-packages` directory (for Anaconda this is usually `C:/Anaconda/lib/site-packages`).

Second
The second item ts

6.2 NIDAQmx Errors

The specified resource is reserved. The operation could not be completed as specified.
This means a task is still running on the device. To end the task and recover, open the NIDAQmx utility from Start → National Instruments → NiMAX. Select Devices and Interfaces → NI USB 6211 then select 'Reset Device', and await confirmation.