Python For The Lab Documentation

Release 1.0

Aquiles Carattino

Apr 27, 2020

Contents:

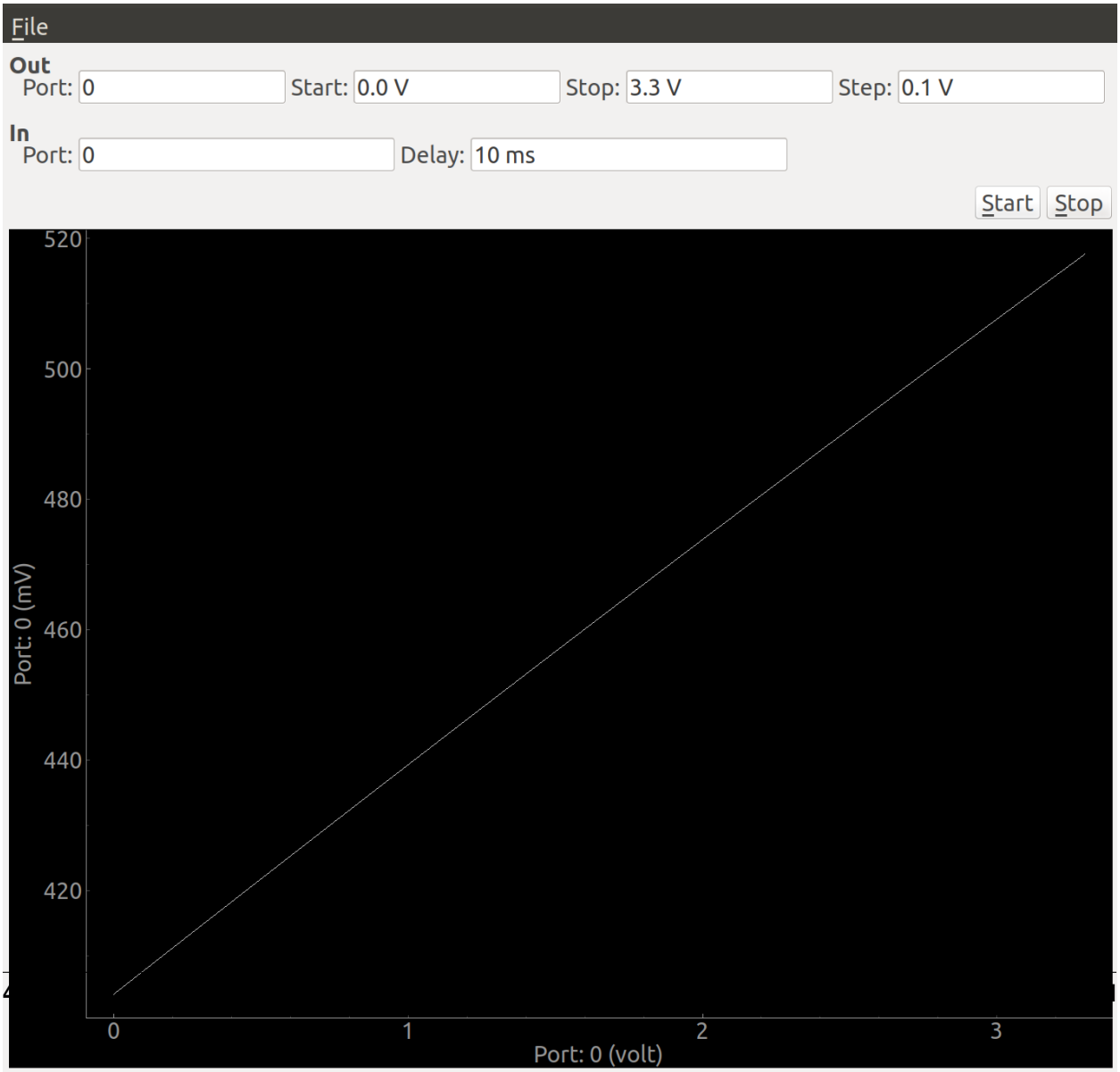
Python for the Lab (PFTL) is a simple program to acquire data from a DAQ device. It is designed following the MVC design pattern, splitting the code into Controllers for defining drivers, Models for specifying the logic on how to use devices and perform an experiment. The View is where all the GUI is developed.

PFTL was developed by Aquiles Carattino to explain to researchers, through simple examples, what can be achieved quickly with little programming knowledge. The ultimate goal of this project is to serve as a reference place for people interested in instrumentation written in Python.

You can find the code of this package at Github, the documentation is hosted at Read The Docs. If you are interested in learning more about Python For The Lab, you can check the courses or get a copy of the book.

CHAPTER 1

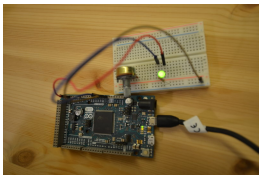
The GUI



If you follow the Python for the Lab course, the GUI is going to be the last step. You perform an analog output scan while acquiring the voltage on a different port. This will allow the users to acquire an I-V scan or any other voltage-dependent measurement.

CHAPTER 2

The Device



The objective of PFTL is to control a device to measure the IV curve of an LED. The device is built on an Arduino DUE which has two Digital-to-Analog channels. The program monitors the voltage across a resistance while increasing the voltage applied to an LED. We can change all the parameters of the scan, including the input and output channels, the range, time delay, etc.

2.1 PythonForTheLab package

2.1.1 Start Function

After installing Python for the Lab it is possible to start it directly from within the command line using *pftl.start*. It takes one argument that is the path to the configuration file.

```
$ pftl.start Config/experiment.yml
```

`PythonForTheLab.start.`**`start`**`()`

Starts the GUI for the experiment using the config file specified as system argument.

2.1.2 Subpackages

PythonForTheLab.Controller

One of the building blocks of the MVC design pattern. Controller hosts all the packages related to communication with devices. Each element should reflect exactly what a device is capable of doing and not the imposed logic from the experimentor. Loops, etc. should be placed within the Models.

Module contents

PFTL DAQ Controller

Python For The Lab revolves around controlling a simple DAQ device built on top of an Arduino. The DAQ device is capable of generating up to two analog outputs in the range 0-3.3V and to acquire several analog inputs.

Because of the pedagogy of the course Python for the Lab, it was assumed that the device can generate value by value and not a sequence. This forces the developer to think on how to implement a solution purely on Python.

class PythonForTheLab.Controller.pftl_daq.**Device**(*port*)

Controller for the serial devices that ships with Python for the Lab.

Parameters **port** (*str*) – The port where the device is connected. Something like COM3 on Windows, or /dev/ttyACM0 on Linux

rsc

The serial communication with the device

Type serial

port

The port where the device is connected, such as COM3 or /dev/ttyACM0

Type str

DEFAULTS = {'baudrate': 9600, 'encoding': 'ascii', 'read_termination': '\n', 'read_

finalize()

Closes the resource

get_analog_input(*channel*)

Get the Analog input in a channel

Parameters

- **channel** (*int*) – The channel
- **output_value** (*int*) – The output value in the range 0-4095

Returns *int* – The value

idn()

Get the serial number from the device.

Returns *str* – The serial number of the device

initialize()

Opens the serial port with the DEFAULTS.

query(*message*)

Wrapper around writing and reading to make the flow easier.

Parameters **message** (*str*) – The message to send to the device

Returns *str* – Whatever the message outputs

set_analog_output(*channel*, *output_value*)

Sets the analog output of a channel

Parameters

- **channel** (*int*) – The channel
- **output_value** (*int*) – The output value in the range 0-4095

Models

Models are where all the logic of the experimentor should be placed. In this case there are two models, one for the DAQ used and one for the Experiment itself. Models rely on Controllers to communicate with real devices and pass the information to the View in order to display it to the user.

Model for Devices

Module contents

Analog DAQ

Class for communicating with a real device. It implements the base for communicating with the device through a Controller. The experiment in mind is measuring the I-V curve of a diode, adding the logic into a separate Model for the experiment may seem redundant, but incredibly useful in bigger projects.

class PythonForTheLab.Model.analog_daq.**AnalogDaq**(*port*)
Bases: object

Simple Model that reflects the logic of the MVC pattern. This model relies on the real controller for communicating with an Arduino based DAQ.

Parameters **port** (*str*) – See *pftl_daq*

port
The port information

Type str

driver
The controller

Type *Device*

finalize()
Set the outputs to 0V and finalize the driver

get_voltage(*channel*)
Retrieve the voltage from the device

Parameters **channel** (*int*) – Channel number

Returns *Quantity* – The voltage read

initialize()
Initialize the driver and sets the voltage on the outputs to 0

set_voltage(*channel*, *volts*)
Set the voltage to a given value on a given channel

Parameters

- **channel** (*int*) – The channel number
- **volts** (*Quantity*) – The value to set, a quantity using Pint

Base DAQ

Base class for the DAQ objects. It keeps track of the functions that every new model should implement. This helps keeping the code organized and to maintain downstream compliancy.

class PythonForTheLab.Model.base_daq.**DAQBase**(*port*)

finalize()

get_voltage(*channel*)

initialize()

set_voltage(*channel*, *volts*)

Dummy DAQ Model

it only generates random values.

class PythonForTheLab.Model.dummy_daq.**DummyDaq**(*port*)
Bases: *PythonForTheLab.Model.base_daq.DAQBase*

get_voltage(*channel*)
Generates a randomg value

Returns *float* – Random value

PythonForTheLab.Model.dummy_daq.**random**() → x in the interval [0, 1).

Experiment Model

Experiment Model

Building a model for the experiment allows developers to have a clear picture of the logic of their experiments. It allows to build simple GUIs around them and to easily share the code with other users.

class PythonForTheLab.Model.experiment.**Experiment**(*config_file*)
Experiment to measure the IV curve of a diode

Parameters **config_file** (*str*) – Path to the config file. Should be a YAML file, later used by *load_daq()*

do_scan()
Does a scan. This method blocks. See *start_scan()* for threaded scans.

finalize()
Finalize the experiment, closing the communication with the device and stopping the scan

load_config()
Load the configuration file

load_daq()
Load the DAQ. Works with DummyDaq or AnalogDaq

save_data()
Save data to the folder specified in the config file.

start_scan()
Start a scan on a separate thread

stop_scan()
Stops the scan.

PythonForTheLab.View

All the files related to the GUI should be placed within the View package. This is the third leg of the MVC design pattern. If the Model is properly built, the Views are relatively simple PyQt objects. It is important to point out that if there is any logic of the experiment that goes into the view, the code is going to become harder to share, unless it is for the exact same purpose.

Start GUI

Convenience function to wrap the initialization of a window. The Experiment class should be created outside and passed as argument.

```
>>> experiment = Experiment()
>>> experiment.load_config('filename')
>>> experiment.load_daq()
>>> start_gui(experiment)
```

PythonForTheLab.View.start_gui.**start_gui**(*experiment*)
Starts a GUI for the ScanWindow using the provided experiment. :param Experiment experiment: Experiment object with a loaded config.

Main Window

This is the central code for the user interface of Python for the Lab. The design of the window is specifcied in its own .ui file, generated with Qt Designer.

class PythonForTheLab.View.main_window.**MainWindow**(*experiment=None*)
Bases: PyQt5.QtWidgets.QMainWindow

Main Window for the user interface

Parameters **experiment** (Experiment) – Experiment model, can be left empty just for testing. Should be instantiated and initialized before passing it.

experiment
The experiment object

Type *Experiment*

plot_widget
Widget to hold the plot

Type pg.PlotWidget

plot
The real plot that can be updated with new data

Type pg.PlotWidget.plotItem

start_button
The start button

Type QPushButton

start_scan()

stop_scan()

update_gui()

update_plot()

2.1.3 Module contents

Python Module Index

p

Index

M

P

Q

R

S

U