PyScan Manual

Peter Timperman

July 28, 2017

1 Introduction

PyScan is a program to control the HD-300 dye laser and quire from a Measurement Computing PCI-DAS1002 data acquisition card. PyScan is divided into seven files: `model.py`, `debug.py`, `controller.py`, `board.py`, `session.py`, `commands.py`, and `view.py`. There is also an `__init.py__` file is used to simplify imports. The programs architecture follows the Model-View-Controller structure.

To run PyScan, navigate to its local directory and run `python controller.py` from the command line.

2 Important Modules

2.1 PyUniversalLibrary

Measurement Computing provides the Universal Library for C, C, and Visual Basic that interfaces with their data acquisition hardware. PyScan uses PyUniversalLibrary, a Python wrapper for the Universal Library's C implementation. PyUniversalLibrary only works in Python 2. Source code, documentation, and installation instructions are available at https://github.com/astraw/PyUniversalLibrary. Documentation for the Universal Library is available at http://www.mccdaq.com/PDFs/manuals/Universal-Library-Help.pdf.

2.2 PySerial

Serial communication in Windows XP are controlled by Win32 binaries. The PySerial module interfaces with these binaries. The most current version of PySerial, 3.3, does not work with Python 2 so PySerial 2.7 must be used.
Documentation is available at https://pythonhosted.org/pyserial/.

2.3 Tkinter

Tkinter is Python's default user interface module. It is used to implement all of PyScan's GUI. Helpful tutorials and guides are available at http://effbot.org/tkinterbook/tkinter-index.htm.

3 Model

The model is intended to represent the logic and attributes of data acquisition and the scanning unit and is defined in `board.py` and `model.py`

Table 1: Gain Codes

Gain	Code
$\pm 5V$	0
$\pm 10V$	1
$\pm 0 - 10V$	100

3.1 `board.py`

The data acquisition card board be initially configured with Measurement Computing's InstaCal software. `board.py` implements logic for reading the data acquisition hard using the PyUniversalLibrary module. The Universal Library method `cbAIn(board, channel, gain)` reads a analog voltage and coverts it to a unsigned 12 bit integer. This integer is then converted to a voltage with `cbToEngUnits(board, gain, dataValue)`. Gain is controlled by integer codes defined in the Universal Library.

3.2 `model.py`

The `model.py` consists of two classes: SerialInterface and Scanner.

3.2.1 Serial Interface

The Serial Interface implements the low level serial communication logic. The instrument using a complex protocol to receive commands and send responses. The instrument will only receive a command after it sends a an `<ENQ>`. After an `<ENQ>` is received a message can be written to the instrument. After command is written, the serial port is read from and received characters are buffered. Each command must have a check sum computed and appended to the command. The algorithm for computing the check sum is defined both in instruments manual and PCScan38. The buffered message is parsed into status and current position.

3.2.2 Scanner

The Scanner class implements the specific commands defined in the instrument's manual. The commands implemented are:

1. `Scan`
2. `Slew`
3. `JogReverse`
4. `JogForward`
5. `Stop`

6. `Pause`
7. `ACK`

Scanner uses an instance of the Serial Interface class to write the specific commands. The instruments position boundaries are hard-coded as class attributes. `monitorBounds()` uses a separate thread if the instrument is within bounds while slewing or jogging. The current position is rounded so that position recorded by the program reflects the instrument's control panel.

4 View

The view implements all of the GUI components using Tkinter. The main frames defined are:

1. Start Menu
2. Main Menu
3. Scan Menu
4. Configure Menu
5. Control Menu

Other special frames used are:

1. Entry Box: Used to record input from the user
2. Scan Start Box: Asks users permission to start plot
3. Reading Window: Parent frame for the plot
4. Scan Plot: Displays the plot

5 Controller

The controller uses two files: `session.py` and `controller.py`.

5.1 `session.py`

The session class is helper class that buffers position and acquired data. It can also format, save, and print buffered data.

5.2 `controller.py`

The controller class is the most complex potion of the entire program. It binds the view's button and controls with the models functions and logic. It controls window and view management, saving and loading settings, and plotting the data. It also defines a dictionary used to manage keyboard bindings. The scanning logic is the most critical component of the program. The Scanner manipulates the laser's position and steps from based on predetermined parameters: start position, and end position, and position increment. The Session class buffers the data which is based to the ScanPlot frame and plotted.

6 Misc.

6.1 `commands.py`

This is helper file that binds the command codes such as `ENQ` or `NUll` to their respective ASCII integer codes.

6.2 `debug.py`

This is a helper class that logs messages to and from the program and instruments in a file called `scan_config.set` . It can be enabled by calling the program with an `--debug` argument from the command line.