PyScan Developers Manual

Peter Timperman

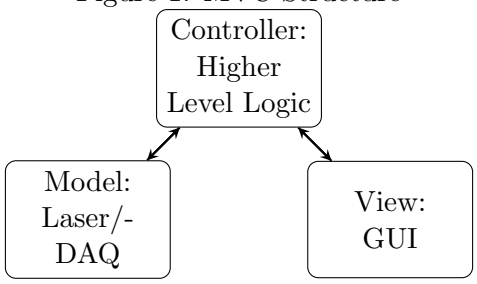
April 24, 2018

1 Introduction

PyScan is a program to control the HD-300 dye laser and aquire 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 program's architecture follows the Model-View-Controller structure (Figure 1).

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

Figure 1: MVC Structure



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 is 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.html`.

3 Version Control

3.1 Git

Git is used both for local version control and collaboration in the cloud. Changes should be committed after every incremental change, fix, or feature addition. Commit messages should be at most two sentences descriptions. A good rule of thumb is if your commit message is too long then you should of have committed earlier. The remote repository is hosted at `https://github.com/poliklab/PyScan/`

```
#Be in the working PyScan Directory
cd <PyScanDirectory>
#Track and stage changes
git stage <files to track>
#To make a local commit
git commit -m "message here"
#To push to repository at github.com
git push
```

3.1.1 `.gitignore`

A `.gitignore` file is maintained to filter out files types that should not be tracked using git. These type include `.pyc`, settings, and log file.

3.2 Directory Dumps

Directory dumps should occur at significant stages of the project, such as start of a semester or when a team member leaves the project. To create a directory copy the current working directory and rename using this pattern: `NameYearMonthDay`. Example, PyScan Directory dump for January 23, 2018: `PyScan180123`. The local git repository for the copied directory should be delinked from the remote repository at github.com using `git remote rm origin`. Git can be used either with the GIT GUI or GIT BASH.

4 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`

4.1 `board.py`

The data acquisition card be initially configured with Measurement Computing's InstaCal software. `board.py` implements logic for reading the data acquisition card 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 (Table 1).

Table 1: Gain Codes

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

4.2 `model.py`

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

4.2.1 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 a 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.

4.2.2 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. The current position is rounded so that position recorded by the program reflects the instrument's control panel.

5 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

6 Controller

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

6.1 `session.py`

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

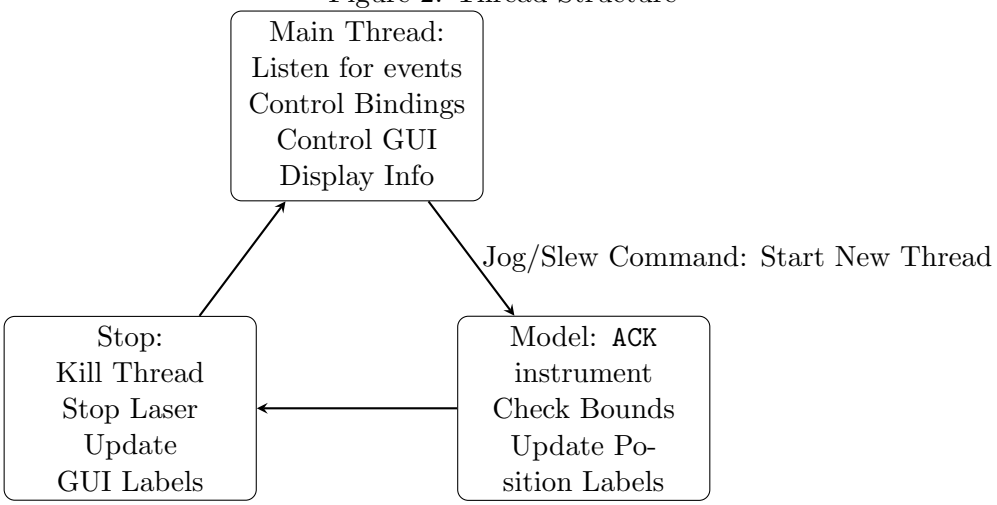
6.2 `controller.py`

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 controller uses the scanner to manipulate the laser's position and to step 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. The main program is divided into three threads: Main Thread, Jog/Slew Thread, and Scan Thread.

6.2.1 Jog/Slew Thread

When jogging or slewing the controller uses function, `monitorBounds()`, which creates a separate thread to ensure instrument is within bounds while slewing or jogging. When the jog or slew function is called via the keyboard or GUI, a new jogging/slewing thread is spun up. Then the jog or slew command is written to the serial port. The thread continually `ACK`s the instrument, check the bounds, updates the instruments recorded position, and rewrites the GUI position label. See Figure 2.

Figure 2: Thread Structure



6.2.2 Scan Thread

This thread takes a data point, redraws the plot, advances to the next point until the scan is stopped or the last data point is reached. This thread can also be paused.

7 Misc.

7.1 `commands.py`

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

7.2 `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 a `--debug` argument from the command line.

7.3 `serial_controller_raw.py`

This a simple serial terminal designed to work with the laser. This program is used for debugging serial communication issues. Enter commands as text given in the SCU manual.

8 Current Bugs

1. Convert ceiling and floor with unit change
2. Save scan file bug

9 Desired Features

1. Suppressing keyboard bindings for slew, jog forward, and jog backward when currently jogging and slewing.
2. Python 3 conversion
3. Implement Official MCC PyUL Library `https://github.com/mccdaq/mcculw`
4. Debug Mode
5. Improved Logging
6. Doubler
7. Monitor thread (basically a version of move thread) for slewing
8. Clear and more configurable export data