Controlling laboratory equipment with Python

Alexey Shkarin

Max Planck Institute for the Science of Light, Erlangen

Outline

- Python overview
- Basics of device communication
  - Message-based communication
  - Library-based communication
- Pylablib introduction
- (Live) demo

Basic Python facts

- Created in early 90’s, 1.0 in 1994, 2.0 in 2000, 3.0 in 2008
- Scripting language, but a lot of standard library code is written in C, so the performance is usually not an issue
- Fairly minimalistic: full language specification is ~100 pages (C is 700, C++ is 1300)
- Open-source and completely free, including implementations and libraries
- “Batteries included”: standard library already has a lot of what you might want; the rest is available as packages, which are installed with a single command
- Popular in many scientific fields, e.g., de-facto standard in ML community

Image sources: official Python logo

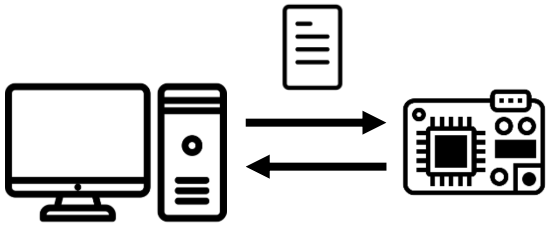
Basic packages

- Scientific data analysis:
  - *Numpy*: data arrays, efficient element-wise operations, basic math and linear algebra, FFT, random number generation
  - *Pandas*: heterogeneous data tables, CSV files
  - *Scipy*: advanced data analysis, optimization, interpolation, basic image processing, special functions
  - *Scikit-learn*: collection of machine learning algorithms
  - *Scikit-image*: collection of image processing and analysis algorithms
  - *OpenCV*: there's a Python wrapper for it, which is regularly updated
  - *Numba*: JIT-compilation of high-performance code (including strong optimization, parallelization, and GPU compilation)
  - *Cython*: C-compiler which takes Python-compatible syntax
  - *Matplotlib/seaborn*: publication-quality plotting
- GUI: *PyQt/PySide* (Qt wrapper), *wxPython* (wxWidgets wrapper), *pyqtgraph* (fast plotting)
- Web tools: *Django*, *Dash*, *Flask*
- Basic language stuff: testing (*pytest*, *nose*), linting (*pylint*, *pep8*, *flake8*), IDEs (*Spyder*, *PyCharm*, *Jupyter*), documentation (*Sphinx*)

Image sources: official Python packages logos

Device communication

- Two basic approaches to device control (from programmer's perspective):
  - Sending and receiving **messages** via some predefined protocol. Language-independent, as long as the physical protocol (serial port, USB, TCP/IP) is supported by the language.



```
instr=serial.Serial("COM5")
instr.write(msg)
reply=instr.readline()
```

  - Using manufacturer-supplied **libraries** (usually .dll for Windows or .so for Linux), which contain precompiled code to directly communicate with custom device drivers.
    C is de-facto standard there, but sometimes wrappers for other languages (LabView, C#, Python) could be provided by the manufacturer or developed by the community.


```
lib=ctypes.cdll.LoadLibrary("devlib.dll")
handle=lib.dlConnectDevice(0)
value=lib.dlReadValue(handle,0x3F)
```

- Sometimes both are combined: libraries provide methods for sending and receiving messages, which do most of the work

Message-based devices

- Typically used for simple devices, where throughput and latency are less of an issue, and complicated synchronization is not required
  - Most HP/Agilent/Keysight electronics (for historical reasons) and corresponding device types (oscilloscopes, AWGs/MWGs, VNAs, MW spectrum analyzers). Examples: Keysight, Tektronix, Rigol.
  - Most simple sensors: pressure gauges, temperature sensors, level meters, power meters. Examples: Lakeshore, Pfeiffer, Leybold, MKS.
  - Many actuators and motion controllers. Examples: Thorlabs, Newport, Attocube, Trinamic.
  - Simple single-purpose devices (lasers, simple motorized components, microcontrollers). Examples: Toptica, M Squared, OZOptics, Arduino.

Message interfaces (how to send)

- Serial port (aka RS-232, COM-port, tty, UART)
  - Very simple, very slow (<2 kByte/s for many devices, <13kByte/s for almost all)
  - Require specifying some parameters (most notably, baud rate), which can be found in device manuals
  - Physically often implemented as a USB connection with a USB<->Serial chip inside
  - Caveat (at least on Windows): COM-ports are system-wide resources, so only one process can be connected to the device at a given time
- TCP/IP (usually via Ethernet)
  - Faster, simple to program, in theory much more extendable
  - Usually only need to know address and port
  - Protocol can handle many connections to the same device; the extent to which devices comply varies
- USB
  - Fairly complex and universal protocol, so you usually rely on manufacturer’s drivers
  - VISA is a common message interface for USB devices

Message protocols (what to send)

- Text protocols
  - Vary widely, many companies and devices will have their own
  - Somewhat common standard is SCPI (created by HP)
  - Usually pretty easy to use (in harder cases might use regular expressions to parse replies)
- Binary protocols
  - Vary even more, everyone comes up with their own
  - Somewhat harder to work with and debug problems
- Sometimes can see a combination (e.g., SCPI): commands in text, large data in binary

Text message protocols

Typically consist of a command/query name, a list of arguments, and a terminator

Tektronix TDS2000

HORizontal:DELay:POSition

Syntax
`HORizontal:DELay:POSition <NR3>`

`HORizontal:DELay:POSition?`

Examples
`HORizontal:DELay:POSition 2.0E–6`

Picomotor 8742

PA

Description	Target position move command.
Syntax	xx**PA**nn
Example	1PA+200000

(Move motor 1 to target position +200000)

Attocube ANC300

stepu <AID> [<C>]	Move <C> steps or continuously upwards (out-wards). An error occurs when the axis is not in "stp" mode.
stepd <AID> <C>	Move number of steps or continuously downwards (inwards).
stop <AID>	Stop any motion.

Some important parameters to consider:

- Termination character for sending or receiving (<CR>, <LF>, or both)
- Command format: number representation, parameter separators, case sensitivity
- Reply format: separators, error messages, echo
- Large data transfer: list of numbers, binary representation, base64

In some cases different approaches might be used (e.g., JSON)

Image sources: documentation for the corresponding devices

Binary message protocols

Typically have fixed length and rigid byte-level structure

Thorlabs APT

MGMSG_MOT_SET_MOVEABSPARAMS **0x0450**

SET:

Command structure (12 bytes)

6 byte header followed by 6 byte data packet as follows:

0	1	2	3	4	5	6	7	8	9	10	11
header						*Data*					
50	04	06	00	d|	s	Chan Ident		Absolute Position			

Some important parameters to consider:

- Arguments size and format, endianness
- Variable message length format (usually length is declared)

Image sources: documentation for the corresponding devices

Related Python libraries

Accessing interfaces

- *pySerial* (https://pyserial.readthedocs.io/) for serial interface devices
- *PyVISA* (https://pyvisa.readthedocs.io/) for VISA devices; also includes some support for other interfaces
- *socket* (built-in) for TCP/IP communications
- *pyft232* (https://pypi.org/project/pyft232/) for some particular serial devices
- *pyusb* (https://pyusb.github.io/pyusb/) for non-VISA USB devices

Dealing with protocols

- *re* (built-in) is sometimes useful to parse text messages
- *struct* (built-in) for generating and parsing fixed-format binary data
- *Numpy* (https://numpy.org/) to parse large binary arrays

Library-based devices

- Typical for more complicated devices: complicated memory management, low latency, high throughput
  - Cameras and frame grabbers. Examples: Andor, PCO, IDS, Hamamatsu, SiliconSoftware, Teledyne Photometrics.
  - DAQs. Examples: National Instruments.
  - Sometimes used to provide high-level device-independent interfaces. Examples: Thorlabs Kinetix, SmarAct.
  - In some cases, required for hardware with specific drivers which are not supported by standard libraries (serial, VISA). Examples: Arcus Technology, HighFinesse.

Related Python libraries

- The main workhorse is the built-in *ctypes* library
  - Allows calling the functions contained within the library (.dll or .so)
  - Provides all necessary C-related functionality: creating variables of a specific type, handling structures, memory allocation, data pointers, function pointers (callbacks)
  - In all but the simplest cases, basic knowledge of C (working with memory, pointers, structures) is highly recommended
  - Unfortunately, does not parse C header files, so all of the definitions (functions, types, structures) need to be done in Python

niimaq.h

```c
//==============================================================================
//  Functions
//==============================================================================
USER_FUNC imgInterfaceOpen(const Int8* interface_name, INTERFACE_ID* ifid);
USER_FUNC imgSessionOpen(INTERFACE_ID ifid, SESSION_ID* sid);
USER_FUNC imgClose(uInt32 void_id, uInt32 freeResources);
USER_FUNC imgSnap(SESSION_ID sid, void **bufAddr);
```

```python
lib.imgSessionOpen.restype=ctypes.c_int32
lib.imgSessionOpen.argtypes=[ctypes.c_uint32,ctypes.POINTER(ctypes.c_uint32)]
sid=ctypes.c_uint32()
lib.imgSessionOpen(0,ctypes.byref(sid))
```


Image sources: corresponding NIIMAQ header file

Pylablib

- The main goal is to encapsulate different device communication methods in a simple object-based package
- Aims to provide predictable interfaces which are consistent across different devices of the same kind
- Supports about 50 different device interfaces from more than 30 different manufacturers

```python
from pylablib.devices import Thorlabs, Andor  # import the device libraries
import numpy as np  # import numpy for saving

# connect to the devices
with Thorlabs.KinesisMotor("27000000") as stage, Andor.AndorSDK2Camera() as cam:
    # change some camera parameters
    cam.set_exposure(50E-3)
    cam.set_roi(0, 128, 0, 128, hbin=2, vbin=2)
    # start the stepping loop
    images = []
    for _ in range(10):
        stage.move_by(10000)  # initiate a move
        stage.wait_move()  # wait until it's done
        img = cam.snap()  # grab a single frame
        images.append(img)

np.array(images).astype("<u2").tofile("frames.bin")  # save frames as raw binary
```

Currently supported devices

Cameras

Andor (iXon, Luca, and Zyla), **Hamamatsu** (Orca Flash and ImagEM), **NI IMAQdx** camera interface (PhotonFocus HD1-D1312), **PCO** (pco.edge), **Thorlabs Scientific Cameras** (Kiralux), **Thorlabs uc480** (Thorlabs DCC1545M), **IDS uEye** (IDS SC2592R12M), **Princeton Instruments** (PIXIS 400), **Photometric** (Prime 95B and KINETIX), **PhotonFocus** frame grabber cameras (MV-D1024E with NI and SiSo frame grabbers), **NI IMAQ** frame grabbers (NI PCI-1430 and PCI-1433), **Silicon Software** frame grabbers (microEnable IV AD4-CL)

Stages and motors

Attocube (ANC300 and ANC350), **Thorlabs APT/Kinesis** (KDC101, K10CR1, BSC201, KIM101), **Newport Picomotor** (8742), **Arcus Performax** (PMX-4EX, PMX-2EX, DMX-J-SA), **Trinamic** (TMCM-1110), **SmarAct** (open-loop SCU-controller)

Sensors

HighFinesse (WS6 and WS7 wavemeters), **Lakeshore** (Lakeshore 218 temperature controller), **Cryocon** (CryoCon 14C temperature sensor), **Pfeiffer** (TPG261 and DPG202), **Leybold** (ITR90), **Kurt J. Lesker** (KJL300), **Ophir** (Vega power meter), **Thorlabs** (TPA101 quadrature detector)

Lasers

Toptica (iBeam Smart), **Lighthouse Photonics** (SproutG), **Laser Quantum** (Finesse), **M2** (Solstis)

Electronics

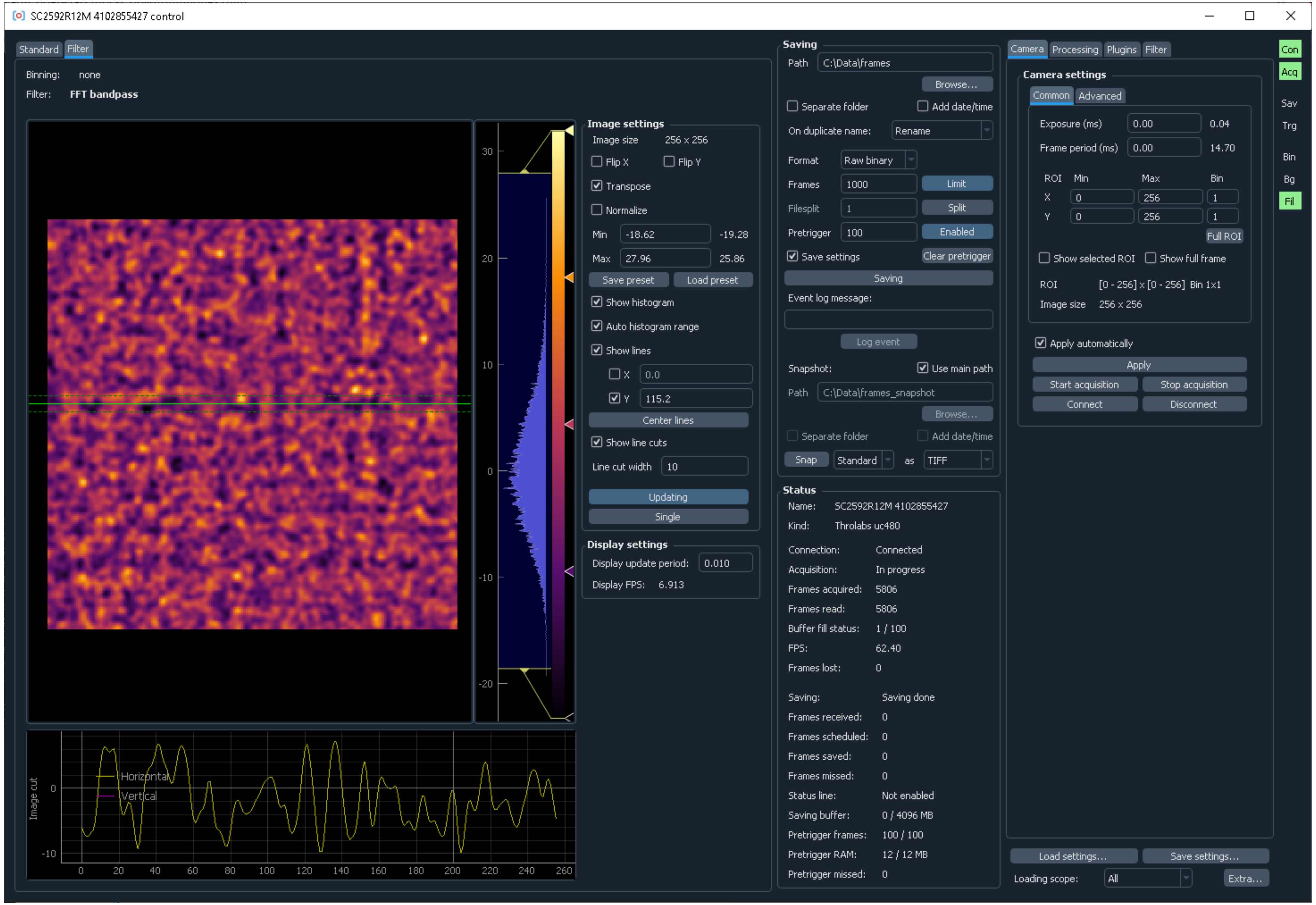
Tektronix oscilloscopes (TDS2002B, TDS2004B, and DPO2004B), **Agilent-style AWGs** (Agilent 33500 and 33220A, Rigol DG1022, Tektronix AFG1022, GW Instek AFG2225 and AFG2115), NI DAQmx (NI USB-6008, NI USB-6343, and NI PCIe-6323)

Misc

Thorlabs (MFF flip mirror, FW filter wheel, MDT693 voltage source), **OZOptics** (EPC04, DD100, TF100), **Arduino** (basic communication wrapper)

The list is available at https://pylablib.readthedocs.io/

Standalone camera control software



- Includes all pylablib-supported cameras
- High performance
- Customizable on-line processing in Python
- Flexible data acquisition

Conclusions and further links

Packages

- *pySerial* (https://pyserial.readthedocs.io/)
- *PyVISA* (https://pyvisa.readthedocs.io/)
- *pyft232* (https://pypi.org/project/pyft232/)
- *pyusb* (https://pyusb.github.io/pyusb/)
- *socket, re, struct, ctypes* (built-in)

Specific devices

- Usually manuals (or “Programming manual”) are a good place to start for message-based devices
- For library-based devices APIs are usually freely available, either separately, or as a part of standard communication software

Pylablib

- Documentation: https://pylablib.readthedocs.io/
- Repository: https://github.com/AlexShkarin/pyLabLib
- Cam-control: https://pylablib-cam-control.readthedocs.io/

Images and text source: Python package logos, device manuals, https://www.flaticon.com/