PLICO: a framework for Adaptive Optics laboratory experiments

Chiara Selmi, Lorenzo Busoni, Alfio Puglisi, Edoardo Bellone de Grecis

PLICO: a framework for Adaptive Optics laboratory experiments

Chiara Selmi[a], Lorenzo Busoni[*a], Alfio Puglisi[a], Edoardo Bellone de Grecis[a]
[a]INAF - Osservatorio Astrofisico di Arcetri, 50125 Firenze, Italy

ABSTRACT

PLICO (Python Laboratory Instrumentation COntrol) is a framework for developing instrument control applications, such as the devices usually available in a scientific laboratory. It is entirely written in Python and based on a client-server model, typically using zeromq as message dispatcher and pickle for serialization and deserialization of data.

Keywords: Adaptive Optics, laboratory, instrumentation control, devices, Python

1. INTRODUCTION

The PLICO project originated in the context of small laboratory experiments with typically 2-5 devices of different types. These devices have to be controlled in a synchronized mode.

We decided to build the framework by creating libraries that expose basic operations for each device type, abstracting from the device details and giving users a simplified interface. Researchers will use standard control commands from these libraries to make the lab experiment application. When possible, drivers provided by the device vendors are used to speed up software development. The entire framework is written in Python, which allows both application development and interactive use.

The architecture framework was designed using a client-server model: device servers handle the device details, while clients connect to servers typically using zeromq as a message dispatcher and pickle for serialization and deserialization of data. The client-server communication details are implemented by the framework, and the application only has to create a client, tell it to which server to connect to, and use the methods defined in the abstract client interface to control and query the device. On the server side, the choice of the device is made through a configuration file. The PLICO framework is available on Github[1] where for each package testing, pip upload, docs and codecov actions are automatically run.

The available packages of the PLICO framework are:

1. plico_motor to control linear and rotary motors
2. plico_interferometer to control interferometers
3. plico_dm to control Deformable Mirrors
4. plico_dm-characterization for deformable mirrors calibration and characterization
5. pysilico to control video cameras, including basic real-time displays

* lorenzo.busoni@inaf.it

2. PURPOSE

In this section, the basic requirements for the realization of the project are reported, explaining the reasons for these starting points and also pointing out which objectives were not realized since they were not of interest for the context of this project.

2.1 Separation of concerns

Our aim is to create libraries separate from the final application: the laboratory application should not concern itself with the device control details. This allows the experiment to be easily moved to another laboratory or reworked at another time, and also makes it possible to update the device hardware without impacting the laboratory application.

2.2 Interactivity

Since a laboratory is a dynamic environment where requirements can change often and ideas have to be tested quickly, it should be possible to use the framework in an interactive or scripting environment, in addition to standard applications. For the same reason, the user experience should be simplified as much as possible, using the same interface for all devices of a given type (e.g. linear motors provided by different vendors), even at the cost of sometimes losing fine control or peculiar capabilities available from a single vendor.

2.3 OS independence

Often, vendor drivers are bound to a specific hardware or operating system. For example, we have devices with drivers only available for Microsoft Windows, or that have hardware controllers installed in a specific PC supplied with the instrument, or we must use equipment already available in the laboratory that is outdated so the drivers are available for a specific version of Windows or Python. We want to abstract away from specific OS/hardware/versioning, and be able to drive all devices from any computer or laptop.

2.4 Shared standard data

Sometimes we work with instruments that require a long calibration procedure before they can be inserted into the experimental setup, e.g deformable mirrors (DM), and it may be that the person using the DM in the laboratory is not the same person who calibrated it. For this reason we want calibration data to be available outside the calibration setup (i.e. deformable mirror in front of interferometer) stored in standard format and available to everyone. The library controlling the device must be able to use the data transparently.

2.5 Not in our requirements

- Real-time interfacing: the execution time of the individual step is not guaranteed. Parallelisation or threading in the handling of requests is not required.
- High speed and low latency.
- High scalability: we are not interested in controlling a large number of devices but only the typical number of instruments required for laboratory setups, typically 2-5 devices of different types.
- Graphical User Interfaces (GUIs): however possible via the client library but the user's target is to use the framework from the terminal. We have implemented a GUI for pysilico (to help during alignment) and one for the motors.
- Server configuration via client interface. Clients have no control over the server configuration.

3. ARCHITECTURE

In order to free applications from the need of managing multiple OS and hardware configurations, we decided on a client-server configuration where the complexity is managed by device servers that expose a simplified interface to clients connecting over TCP/IP. Multiple clients can connect to the same server, and their commands are handled without priority. Servers also provide the device status in broadcast mode to all connected clients, and use a configuration file (not visible to clients) to specify the device type and details.

Given the interactivity requirements, we chose Python as the implementation language for the whole project. Servers are Python applications and clients are Python classes that can be imported by the user's application, or run interactively in a Python/IPython terminal. Depending on the hardware to be controlled, servers may import vendor-provided Python modules, or might even load custom C++ code if needed, while clients are pure Python code.

Communication between servers and clients uses an RPC mechanism based on ZeroMQ as a message dispatcher and pickle for serialization and deserialization of data. These communication details are hidden from the user application.

3.1 Clients

Each device type corresponds to a Python abstract class, which defines methods to control a device and query its status. For each device type, Plico provides a client implementation that is able to communicate with any server of that device type. Special implementations are sometimes available for testing or HW simulation, running a client without a corresponding server.

The application creates a new client instance giving it the server TCP/IP address, and when the client's methods are called, the call is forwarded to the server and the answer returned to the application. Any server must expose the methods for the corresponding device type (e.g. for a motor: move to a new position, start homing, query position…), and implement them for its particular OS/hardware combination. Therefore, a single client class is able to connect to any server that implements the device type.

3.2 Servers

On the server side, we decided on a generic server class that can instantiate one of several kinds of known devices. The choice of the device to instantiate is done via a global server configuration file. In this file all the possible devices are defined by model, serial number and other parameters depending on the group to which the device belongs (e.g. library position for Software Development Kit (SDKs) and reference shape for DMs or binning for cameras). The same file defines the list of servers to instantiate, each with the device to be controlled. Each server will run in its separate process.

Figure 1 shows the server control loop and device selection via configuration file in the specific case of the deformable mirror server.

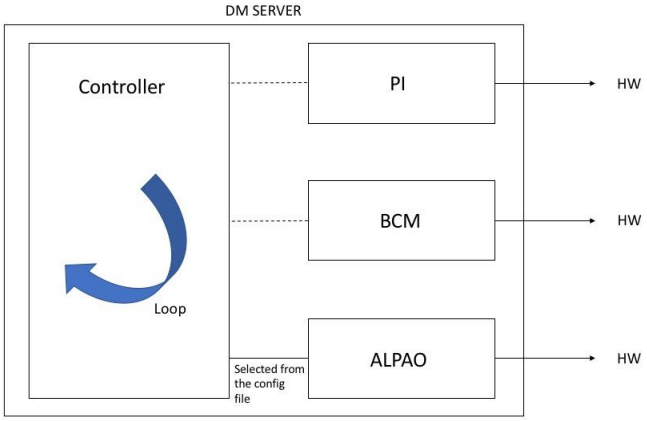

Figure 1. DM server control loop. The controller code can handle one of three different devices, and a configuration file selects the device to use. Any client will use an abstract interface identical for the three devices.

The server is designed to accept commands from multiple clients with no priority, i.e. following the simplest scheduling algorithm First Come, First Served (FCFS): we did not create parallelisation or threads in the handling of requests and for this reason real-time interfacing, high speed and low latency are not in our requirements. Acceptance of multiple clients allows interactive terminal overwriting of application behavior. Server also provides device status in broadcast mode which is a useful feature for device monitoring, data storage and Graphical User Interfaces (GUIs).

Figure 2 shows an example of system architecture used in a laboratory experiment: the deformable mirror (DM) calibration is made using an interferometer (INTERF). The mirror is connected to the vendor-supplied Windows PC via USB or Ethernet and the software runs on that PC using SDK from the vendor; the interferometer is connected to its windows PC via ethernet. The main client, OS independent and responsible for the application of the experiment, communicates with servers via Ethernet. Servers provide information in broadcast mode to other clients for applications such as GUI monitoring.

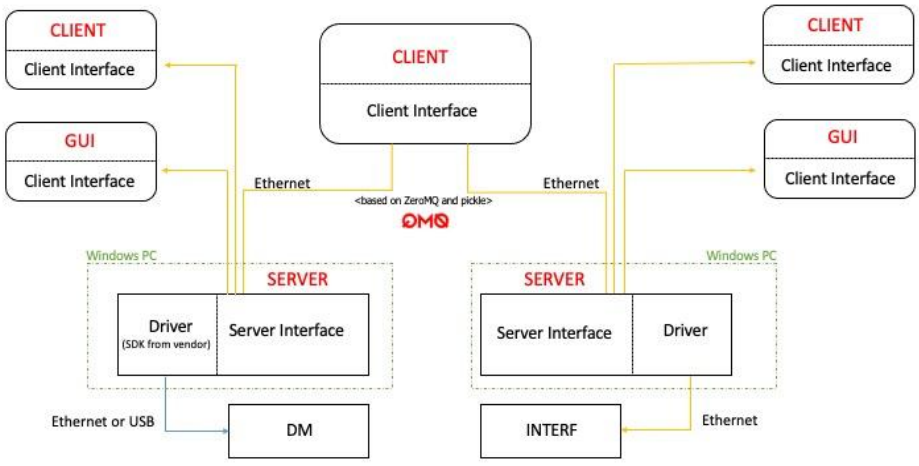

Figure 2. Diagram of the system architecture used in the laboratory for the deformable mirror calibration.

PLICO framework also has testing and documentation. The tests developed are unittest and integration test where also devices and SDK are simulated. The documentation is made using Sphinx and regularly updated on the hosting platform '*Read the Docs*[2]*'*. In addition, each application that constitutes the framework uses the Python logging functions.

4. STATUS OF IMPLEMENTATION

In this section are reported the available packages of the PLICO framework.

4.1 pysilico / pysilico_server

Pysilico is used to control basic camera parameters like exposure time and binning. It provides a slow interface to get full frames and a fast interface with reduced-resolution frames for fast real time displays. It's the first application introduced in the PLICO framework.

Servers are available for:

- Allied AVT/Prosilica using Vimba
- FirstLight Ocam2k (in progress)

Pysilico is used for the M4 Sensor of Phase Lag (SPL), LIFT bench of GMT, PRISMA (Pyr WFS for metrology applications), didactics, ERIS MAIT in Arcetri and requests from outside.

4.2 plico_motor / plico_motor_server

Plico_motor defines a motor as a device that can be controlled in step along one axis, including for example tunable filters (the axis is the wavelength setting) and filter wheels.
Servers are available for:

- Any motor using the PI GCS communication protocol with serial or USB connection
- Tunable Filter (KURIOS VB1 - Thorlabs) with serial or USB connection
- Filter Wheel (FW102B - Thorlabs) with serial or USB connection
- Newport Picomotor linear actuators
- Standa 8SMC5-USB motor using driver Standa 8smc4-5

Plico_motor is used for the M4 Sensor of Phase Lag (SPL) and the LIFT bench of GMT.

4.3 plico_interferometer / plico_interferometer_server

Servers are available for controlling Wyko and PhaseCam interferometers. We use Pyro to interface with the 4Sight SW GUI provided by the vendor and the interfacing depends on the version of the GUI and the version of the Python used. This interface mode allows you to control PhaseCam 4020 and Wyko 4110. PhaseCam 6110 and all the other models that use WCF interface for 4SightFocus are controlled using JSON (no server needed).

Plico_interferometer is used in the Optical Test Tower (OTT) of M4, in the Arcetri laboratories and for requests from outside.

4.4 plico_dm / plico_dm_server

Plico_dm defines a deformable mirror as a device with N actuators that can be positioned independently, using a single command vector.
Servers are available for:

- ALPAO DMs (using asdk.py provided by ALPAO)
- MEMS Boston Micromachines Corporation (BMC), using bmc.py files provided by the vendor
- PI tip tilt mirror implemented with PI CGS
- Spatial Light Modulator by Meadowlark.

Plico_dm uses zonal control for all DMs. A modal control feature is planned.

4.5 plico_dm_characterization

It's an application for calibration and characterization of deformable mirrors that uses plico_dm and plico_interferometer. Even if outside of the low-level scope of device control, it has been included in plico since it is a common and useful use case. Its goal is to create the Interaction Matrix and the flat command for the mirror and to return them as output data in a standard format to be shared. The library also has some functions for measuring and analyzing data for mirror characterisation.

The algorithm was used to perform the calibration of ALPAO 277/ALPAO 88 for the ERIS DSM simulator[3] and ALPAO 97 in use at SHARK-NIR[4] . It will also be used for the calibration of ELT's M4.

4.6 Future developments

One limitation of the current scheme is that a client must know the hostname/port to connect to control a specific device. We are planning to implement an auto-discovery feature where a client can automatically find the correct server for a given device name/model, or alternatively display a list of available devices and let the user choose among them.

Sometimes a device makes available many more features than the ones defined by the plico client. In this case, we plan to implement an interface that allows sending to a device server a vendor-specific command in a generic way, e.g. with a text buffer. This allows the use of vendor-specific features, even without fine control on parameters or errors.

5. APPLICATION EXAMPLE

In this section we report on an application of the plico framework involving LIFT, the GMT Linearized Focal-plane Technique, and SPL, the M4 Sensor of Phase Lag.

5.1 LIFT and SPL scientific goal

Adaptive optics systems for the future Extremely Large Telescopes will have to deal with large gaps in the pupil due to the spiders and/or the segmentation of one or several mirrors. These gaps are typically larger than the expected r0 at the sensing wavelength. They can thus create significant discontinuities in the wavefront, which might eventually lead to the so-called "island effect" or "petaling": the wavefront in each segment is well corrected, but differential pistons at a multiple of the sensing wavelength appear between the segments. LIFT and SPL are a potential solution to correct the differential pistons generated while the adaptive optics system is running.

5.2 Laboratory setup

An experimental bench with a two-segment mirror was set up in Arcetri to reproduce the differential piston gap: using a beam splitter the same piston difference is measured by the two sensors. LIFT[5] images were taken on a defocused camera (AVT G-033) using one narrow-band filter (Thorlabs - FW102B) at a time, while SPL[6] images were taken using a tunable narrow-band source (Thorlabs - KURIOS VB1) for wavelength sweep and a Allied Vision - Manta G-125B as camera. Figure 3 shows the setup realized in the Arcetri laboratories.

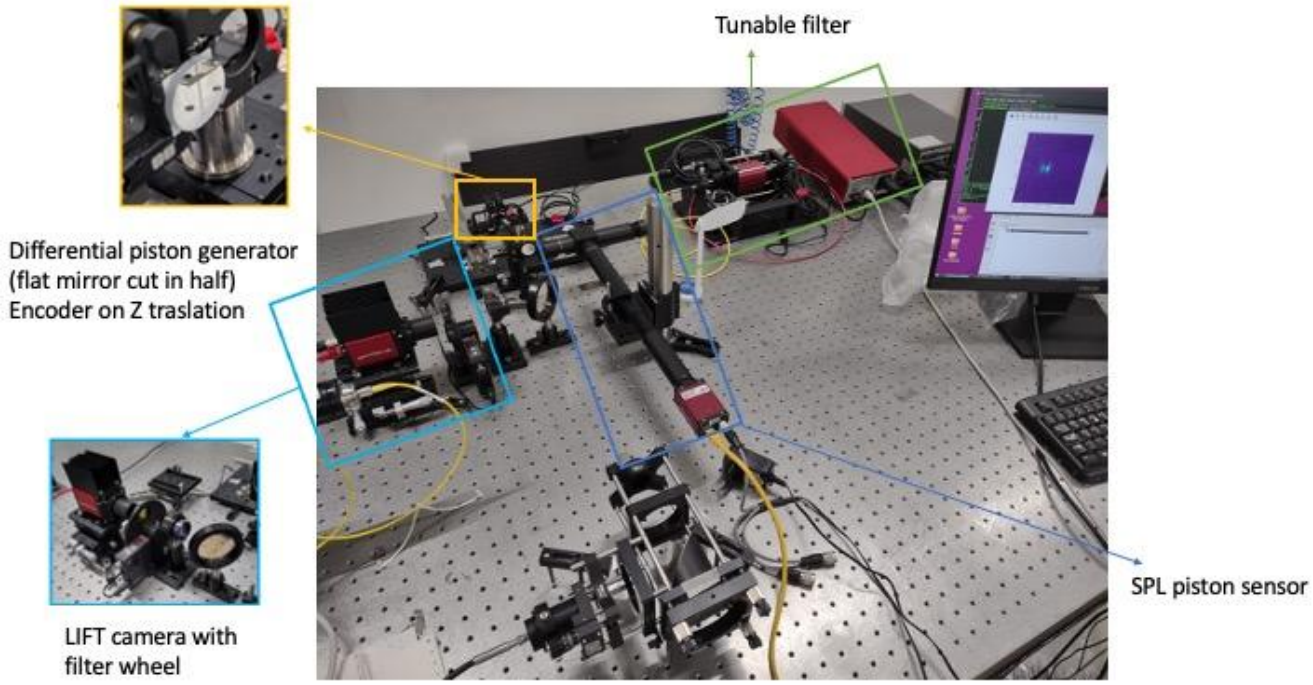

Figure 3. A two-segment mirror reproduces a differential piston gap for LIFT bench setup and SPL sensor setup. All the devices used in the experiment are controlled using the PLICO framework.

A Linux computer is used as a server machine: the motors are physically connected by USB while the cameras use an ethernet connection. All servers are running on the same machine, while another computer or laptop is used as a client to carry out the measurements of the two experimental setups.

Since this experimental bench only uses the pysilico and plico_motor library it is also possible to take advantage of GUIs. Figure 4 shows the control GUI for motors on the left and the control GUI for cameras on the right: once the host server and port number have been indicated, it is possible to create the connection to the server and use the commands provided to control the device. It is necessary to run as many GUIs as there are devices to be controlled.

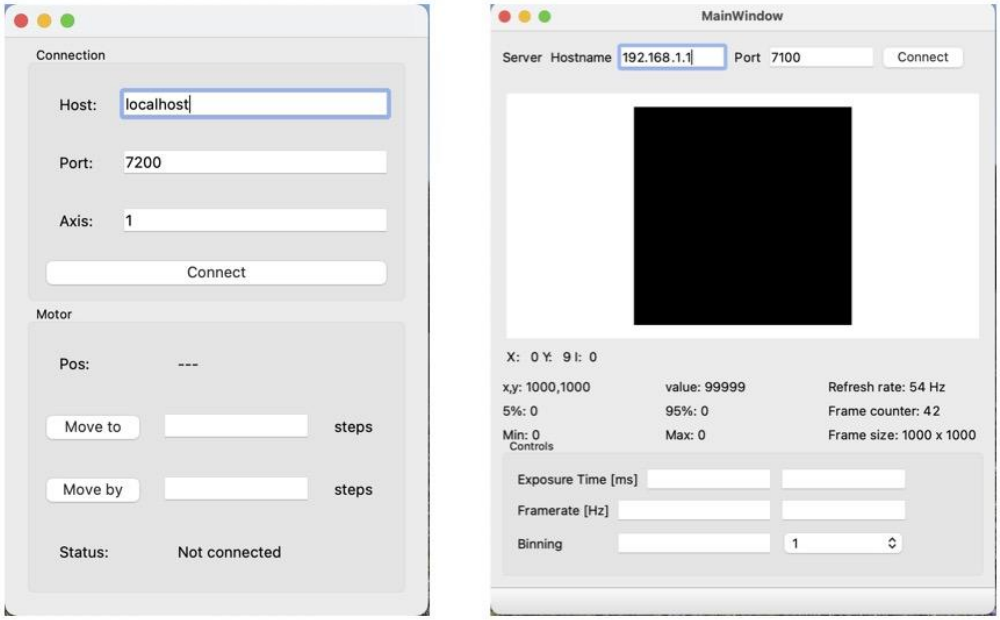

Figure 4. Control GUI for motors on the left and control GUI for cameras on the right.

In parallel with the GUIs, the system was also controlled via Python scripting, importing the plico and pysilico modules on the fly, as shown in Figure 5.

SPL_controller.py (~/git/SPL/spl) - VIM

```
'''
Authors
    - C. Selmi: written in 2021

HOW TO USE IT::

    from spl import SPL_controller as s
    camera = s.define_camera()
    filter = s.define_filter()
    tt, piston = s.SPL_measurement_and_analysis(camera, filter)
'''
import os
import numpy as np
from spl.SPL_data_acquirer import SplAcquirer
from spl.SPL_data_analyzer import SplAnalyzer
from spl.conf import configuration as config

def define_camera():
    ''' Function to use to define the camera with pysilico
    '''
    #far partire pysilico_server_2
    import pysilico
    cam = pysilico.camera(config.IPCAMERA, config.PORTCAMERA)
    return cam

def define_filter():
    ''' Function to use to define the tunable filter
    '''
    #far partire plico_motor_server_3
    from plico_motor import motor
    filter = motor(config.IPFILTRO, config.PORTFILTRO, axis=1)
    return filter
```

Figure 5. Python script screenshot

REFERENCES

[1] https://github.com/ArcetriAdaptiveOptics/plico

[2] https://plico.readthedocs.io

[3] Runa Briguglio, Armando Riccardi, Luca Carbonaro, Chiara Selmi, Paolo Grani, Enrico Pinna, *"The simulator of the VLT Deformable Secondary Mirror: a test tool for adaptive optics instruments for the Yepun-UT4 telescope",* SPIE 2022.

[4] Biondi Federico, Arcidiacono Carmelo, Selmi Chiara, and al., *"The laboratory characterization of the SHARK-NIR Adaptive Optics channel" ,* Wavefront sensing in the VLT/ELT era, 4th edition 2019.

[5] S. Lombardi, C. Plantet, A.- L. Cheffot, M. Bonaglia, A. Puglisi, C. Selmi, L. Busoni, S. Esposito, *"Experimental validation of large differential piston sensing with the double-wavelength LIFT"*, AO4ELT7, Jun 2023.

[6] Runa Briguglio , Giorgio Pariani , Marco Xompero , Armando Riccardi , Matteo Tintori , Elise Vernet , and Marc Cayrel, *"Performances of the phasing sensors for the M4 adaptive unit"*, AO4ELT6, Jun 2019.