Foswiki > FLAO Web > SoftwareDevelopment > InterferometerLib (22 Oct 2009, LucaFini)

Interferometer control

Requirements

An interferometer (at the moment the *4D PhaseCam 4020*) must be controlled from IDL to perform the adaptive secondary flattening. In genaral, and for the PhaseCam 4020 model too, the interferometer is attached to a stand-alone workstation, and must be **controlled remotely**.

Is fundamental to consider the following requirement:

{req_intf_1} The interferometer functionalities must be requested using an API independent from the specific interferometer model.

The API should come as part of the AO-Supervisor, but only the implementation for the 4D PhaseCam 4020 Interferometer will be delivered. Because the AO-Supervisor is written in C/C++ and Python, only these languages must be used.

{req_intf_2} The interferometer API must provide this set of features

- **Initialization**: setup the interferometer, loading a specified configuration
- **Trigger enabling/disabling**: setup the hw trigger
- **Get measurement**: acquire a new interferometer measurement and return "reference" to it.
  - This routine should be only used to acquire a "test" frame, not during the normal *flattening operations* (because it can be slow).
  - This routines must not be used to acquire a burst with aloop af calls.
- **Get measurement burst**: acquire a burst (set of measurements) and return a "reference" to them.
  - This routine must guarantee a lower bound for the performance on the burst acquisition: at the moment **4 Hz is required.**
  - The performance is relative to the acquisition in the interferometer workstation, without considering the measurements' transfer to the local IDL host.

Interferometer Library

Defines a C++ communication interface to send commands and receive replies to/from Interferometer Controller. See lib/IntfLib/InterferometerInterface.h for more info.

This library is wrapped to be used from **IDL**: see idl_wrapper/idl_wraplib.dlm (*4D INTERFEROMETER* section) for a list of available functions.

Interferometer Controller

To satisfy the *req_intf_1* the **InterferometerCtrl** is implemented as an AOApp (C++ *server* application).

The *InterferometerCtrl* it is composed by a few modules (classes):

- InterferometerCtrl: AOApp implementation that receive MsgD commands, execute them and reply.
- AbstractFactory: *generic interterometer controller, providing initialization* and creation of *AbstractMeasurement* and *AbstractMeasurementBurst* objects
- AbstractMeasurement: proxy for a generic measurement
- AbstractMeasurementBurst: proxy for a generic measurement burst

All the Abstract classes must be derived, implementing their pure virtual methods, to provide a concrete interferometer controller.

4D PhaseCam 4020

The interferometer 4D PhaseCam 4020 is managed by a dedicated PC running Windows 2000 and a specific software (4D GUI). The interferometer can be controlled only using a python library, which runs inside the 4D GUI using an embedded python interpreter.

NOTE: 4D didn't provide any way to use a standolone Python interpreter, neither to directly use the Windows DLL to control the interferometer. The only alternative (and official) way to remotely control the interferometer is using Java WebService module (provided by 4D)

Implementation details

The **4DFactory** is the implementation of an AbstractFactory for 4D PhaseCam 4020 interferometer. This class wraps a **Python Client** wich is able to communicate with a remote **Python Controller/Server** using the Pyro library (all the python code is in CVS: Supervisor/PyModules/I4D).

Basically the C++ library is only a local interface providing some logging: all the job is done remotely by the Python Controller/Server.

IMPORTANT NOTE: please set the PYRO_CONFIG_FILE environment variable to a valid config file, i.e. $(ADOPT_ROOT)/left/Pyro_Client.conf

Performance

The Python implementation of the Controller/Server determines the performances: it reaches the speed of **8 Hz** using a detector mask suitable for the flattening (a ring inscribed in the detector's rectangular mask).

Parameters

Some Controller/Server parameters are store in the class *PyModules/I4D/Commons.Constants*:

- I4D_CONFIG_FILE_DIR = 'C:/4D/config
- I4D_DEFAULT_CONFIG_FILE = 'default.ini'
- **I4D _DATA_PATH = 'D:/4D/Data'**

- I4D_RAW_DATA_PATH = 'raw'
- I4D_CALIB_DATA_PATH = 'calibrated'
- I4D_HDF5_DATA_PATH = 'hdf5'
- I4D_FITS_DATA_PATH = 'fits'
- I4D_SINGLE_MEAS_FOLDER = "SINGLE_MEAS"
- I4D_FOLD_PREFIX = "BURST"
- I4D_MEAS_PREFIX = "MEAS"
- I4D_CALIBRATE_RAW_FRAMES = 1
- I4D_REMOVE_RAW_FRAMES = 0
- I4D_POSTPROCESS = None # *Available values: NONE, HDF5* At the moment Only I4D _POSTPROCESS is modifiable using the API.

Path and file naming

The Python controller/server defines the rules for saving the measurements (used symbols are defined in *Commons.Constants* class).

- The remote root for data is fixed: **I4D _DATA_PATH**

Measurement's folders **(MEAS_FOLDER**):

- The folder containing single measurements is fixed: **I4D_DATA_PATH/I4D_SINGLE_MEAS_FOLDER**
- The folder containing a burst is got from the parameter 'burstName' of *getMeasurementBurst(...)* is specified: **I4D _DATA_PATH/'burstName'**
- The folder containing a burst is automatically created: **I4D _DATA_PATH/I4D_FOLD_PREFIX_yyyymmdd_hhmmss**

Measurement's subfolders:

- The **raw** files are stored in a fixed subfolder: **MEAS_FOLDER/I4D_RAW_DATA_PATH**
- The **calibrated** files are stored in a fixed subfolder: **MEAS_FOLDER/I4D_CALIB_DATA_PATH**
- The **hdf5** files are stored in a fixed subfolder: **MEAS_FOLDER/I4D_HDF5_DATA_PATH**

Measurement's name:

- The name of a single measurement is got from the parameter 'measName' of *getMeasurement(...):* **measName_0000**
- The name of each measurement in a burst is fixed *:* **I4D _MEAS_PREFIX_nnnn**
- The extension is respectively **.dat** and **.h5** for raw/calibrated and hdf5 files

Logging

There are 4 differents logging:

- InterferometerLib: log to $(ADOPT_LOG) in the log file defined by the process that uses it.

- InterferometerCtrl: log to $(ADOPT_LOG)/INTERFCTRL00.log
- Python Pyro Client: log to $(ADOPT_LOG)/Client_Pyro_userlog.log
- Python Pyro Controller/Server (remote): D:/4D/dat/Pyro/Server_Pyro_userlog.log

-- FabioTosetti - 22 Apr 2008

Edit | Attach | Print version | History: r12 < r11 < r10 < r9 | Backlinks | View wiki text | Edit wiki text | More topic actions

Topic revision: r12 - 22 Oct 2009, LucaFini