# AOloopControl

Olivier Guyon

Jul 27, 2017

## Contents

# 1 Initial Setup

## 1.1 Scope

AO loop control package

## 1.2 Pre-requisites

Libraries required :

- gcc
- openMP

- fitsio
- fftw (single and double precision)
- gsl
- readline
- tmux

Recommended:

- CUDA
- Magma
- shared memory image viewer (`shmimview` or similar)

## 1.3 Installing the AdaptiveOpticsControl package

Source code is available on the AdaptiveOpticsControl git hub repository.

Download the latest tar ball (.tar.gz file), uncompress, untar and execute in the source directory (`./AdaptiveOpticsControl-<version>/`)

```
./configure
```

Include recommended high performance compile flags for faster execution speed:

```
./configure CFLAGS='-Ofast -march=native'
```

If you have installed CUDA and MAGMA libraries:

```
./configure CFLAGS='-Ofast -march=native' --enable-cuda --enable-magma
```

The executable is built with:

```
make
make install
```

The executable is `./AdaptiveOpticsControl-<version>/bin/AdaptiveOpticsControl`

## 1.4 Setting up the work directory

Conventions:

- `<srcdir>` is the source code directory, usually `.../AdaptiveOpticsControl-<version>`

- `<workdir>` is the work directory where the program and scripts will be executed. Note that the full path should end with `.../AOloop<#>` where `<#>` ranges from 0 to 9. For example, `AOloop2`.

The work directory is where all scripts and high level commands should be run from. You will first need to create the work directory and then load scripts from the source directory to the work directory by executing from the source directory the 'syncscript -e' command:

```
mkdir /<workdir>
cd <srcdir>/src/AOloopControl/scripts
./syncscripts -e /<workdir>
cd /<workdir>
./syncscripts
```

Symbolic links to the source scripts and executable are now installed in the work directory :

```
olivier@ubuntu:/data/AOloopControl/AOloop1$ ls -l
total 28
drwxrwxr-x 2 olivier olivier 4096 Feb 21 18:14 aocustomscripts
drwxrwxr-x 2 olivier olivier 4096 Feb 21 18:14 aohardsim
lrwxrwxrwx 1 olivier olivier   57 Feb 21 18:14 aolconf -> /home/olivier/src/Cfits/src/AOloopCont
drwxrwxr-x 2 olivier olivier 4096 Feb 21 18:14 aolconfscripts
lrwxrwxrwx 1 olivier olivier   70 Feb 21 19:08 AOloopControl -> /home/olivier/src/Cfits/src/AOlo
drwxrwxr-x 2 olivier olivier 4096 Feb 21 18:14 aosetup
drwxrwxr-x 2 olivier olivier 4096 Feb 21 18:14 auxscripts
lrwxrwxrwx 1 olivier olivier   61 Feb 21 18:13 syncscripts -> /home/olivier/src/Cfits/src/AOloop
```

If new scripts are added in the source directory, running `./syncscripts` again from the work directory will add them to the work directory.

The main executable is `./AOloopControl`, which provides a command line interface (CLI) to all compiled code. Type `AOloopControl -h` for help. You can enter the CLI and list the available libraries (also called modules) that are linked to the CLI. You can also list the functions available within each module (`m? <module.c>`) and help for each function (`cmd? <functionname>`). Type `help` within the CLI for additional directions.

```
olivier@ubuntu:/data/AOloopControl/AOloop1$ ./AOloopControl
type "help" for instructions
Running with openMP, max threads = 8  (defined by environment variable OMP_NUM_THREADS)
LOADED: 21 modules, 269 commands
./AOloopControl > m?
    0           cudacomp.c    CUDA wrapper for AO loop
```

```
    1  AtmosphericTurbulence.c    Atmospheric Turbulence
    2     AtmosphereModel.c    Atmosphere Model
   3            psf.c   memory management for images and variables
    4       AOloopControl.c    AO loop control
    5           AOsystSim.c    conversion between image format, I/O
    6    AOloopControl_DM.c    AO loop Control DM operation
    7         OptSystProp.c    Optical propagation through system
    8        ZernikePolyn.c    create and fit Zernike polynomials
    9         WFpropagate.c    light propagation
   10         image_basic.c    basic image routines
   11        image_filter.c    image filtering
  12       image_gen.c   creating images (shapes, useful functions and patterns)
  13    linopt_imtools.c   image linear decomposition and optimization tools
   14           statistic.c    statistics functions and tools
   15                 fft.c    FFTW wrapper
   16                info.c    image information and statistics
   17       COREMOD_arith.c    image arithmetic operations
   18      COREMOD_iofits.c    FITS format input/output
  19    COREMOD_memory.c   memory management for images and variables
   20       COREMOD_tools.c    image information and statistics
./AOloopControl > exit
Closing PID 5291 (prompt process)
```

The top level script is `aolconf`. Run it with `-h` option for a quick help

```
./aolconf -h
```

## 1.5 Supporting scripts, aolconfscripts directory

Scripts in the `aolconfscripts` directory are part of the high-level ASCII control GUI

| Script | Description |
|---|---|
| **aolconf_DMfuncs** | DM functions |
| **aolconf_DMturb** | DM turbulence functions |
| **aolconf_funcs** | Misc functions |
| **aolconf_logfuncs** | data and command logging |
| **aolconf_menuconfigureloop** | configure loop menu |
| **aolconf_menucontrolloop** | control loop menu |
| **aolconf_menucontrolmatrix** | control matrix menu |
| **aolconf_menu_mkFModes** | Make modes |
| **aolconf_menurecord** | |
| **aolconf_menutestmode** | Test mode menu |
| **aolconf_menutop** | Top level menu |

| Script | Description |
|---|---|
| **aolconf_menuview** | Data view menu |
| **aolconf_readconf** | Configuration read functions |
| **aolconf_template** | Template (not used) |

## 1.6 Supporting scripts (./auxscripts directory)

Scripts in the `auxscripts` directory are called by aolconf to perform various tasks. To list all commands, type in the `auxscripts` directory :

```
./listcommands
```

The available commands are listed in the table below. Running the command with the `-h` option prints a short help.

| Script | Description |
|---|---|
| ./mkDMslaveActprox | Create DM slaved actuators map |
| ./aolctr | AO control process |
| ./aolPFcoeffs2dmmap | GPU-based predictive filter coeffs -> DM MAP |
| ./aolInspectDMmap | Inspect DM map |
| ./acquRespM | Acquire response matrix |
| ./waitonfile | Wait for file to disappear |
| ./aolRM2CM | Align Pyramid camera |
| ./aolMeasureLOrespmat | Acquire modal response matrix |
| ./aollinsimDelay | Introduce DM delay |
| ./aolrun | Run AO control loop |
| ./aolMeasureZrespmat | Acquire zonal response matrix |
| ./shmimzero | Set shared memory image stream to zero |
| ./aolLinSim | AO Linear Simulator |
| ./aolmcoeffs2dmmap | GPU-based MODE COEFFS -> DM MAP |
| ./MeasDMmodesRec | Measure AO loop DM modes recovery |
| ./xp2test | Compute cross-product of two data cubes |
| ./aolmkLO_DMmodes | Create LO DM modes for AO loop |
| ./xptest | Compute cross-product of a data cube |
| ./aolblockstats | Extract mode values from WFS images, sort per block |
| ./aolMergeRMmat | Merge HO and LO resp matrices |
| ./aolscangain | AO scan gain for optimal value |
| ./aolARPFautoUpdate | Automatic update of AR linear predictive filter |
| ./aolMeasureLOrespmat2 | Acquire modal response matrix |
| ./aolgetshmimsize | Get shared memory image size |
| ./aolWFSresoffloadloop | Compute real-time WFS residual image |
| ./alignPyrTT | Align Pyramid TT |

| Script | Description |
|---|---|
| ./aolmkmodes2 | Create modes for AO loop |
| ./aolmkMasks | Create AO wfs and DM masks |
| ./modesextractwfs | Extract mode values from WFS images |
| ./aolARPFautoApply | Apply real-time AR linear predictive filter |
| ./aolmon | Display AO loop stats |
| ./aolRMmeas_sensitivity | Measure photon sensitivity of zonal response matrix |
| ./mkHpoke | Compute real-time WFS residual image |
| ./aoloffloadloop | DM offload loop |
| ./Fits2shm | Copy FITS files to shared memor |
| ./aolMeasureZrespmat2 | Acquire zonal response matrix |
| ./aolApplyARPF | Apply AR linear predictive filter |
| ./selectLatestTelemetry | Compute real-time WFS residual image |
| ./aolReadConfFile | AOloop load file to stream |
| ./predFiltApplyRT | Apply predictive filter to stream |
| ./aolCleanZrespmat2 | Cleans zonal resp matrix |
| ./processTelemetryPSDs | Process telemetry: create open and closed loop PSDs |
| ./listrunproc | List running AOloop processes |
| ./aolmkmodesM | CREATE CM MODES FOR AO LOOP, MODAL DM |
| ./aolCleanZrespmat | Cleans zonal resp matrix |
| ./waitforfilek | Wait for file to appear and then remove it |
| ./aolMeasureTiming | Measure loop timing |
| ./aolSetmcLimit | Compute real-time WFS residual image |
| ./alignPcam | Align Pyramid camera |
| ./aolmkWFSres | Compute real-time WFS residual image |
| ./MeasureLatency | Measure AO system response latency |
| ./aollindm2wfsim | Convert DM stream to WFS image stream |
| ./aolCleanLOrespmat | Measure zonal resp matrix |
| ./mkDMslaveAct | Create DM slaved actuators map |
| ./aolautotunegains | Automatic gain tuning |
| ./aolARPF | AO find optimal AR linear predictive filter |
| ./aolzploopon | WFS zero point offset loop |
| ./aolApplyARPFblock | Apply AR linear predictive filter (single block) |
| ./aolmkmodes | Create modes for AO loop |
| ./aolARPFblock | AO find optimal AR linear predictive filter (single block) |
| ./MeasLoopModeResp | Measure AO loop temporal response |
| ./aol_dmCave | dmC temporal averaging |

### 1.7 Hardware simulation scripts

Scripts in the `aohardsim` directory are called to simulate hardware for testing / simulations.

| Script | Description |
|---|---|
| **aosimDMstart** | Start simulation DM shared mem |
| **aosimDMrun** | Simulates physical deformable mirror (DM) |
| **aosimmkWF** | creates properly sized wavefronts from pre-computed wavefronts |
| **aosimWPyrFS** | Simulates WFS |

## 2 Hardware Simulation

### 2.1 Overview

There are 3 methods for users to simulate hardware

- METHOD 1: Provide an external simulation that adheres to AOloopControl input/output conventions
- METHOD 2: Use the physical hardware simulation provided by the package
- METHOD 3: Use the linear hardware simulation: this option is fastest, but only captures linear relationships between DM actuators and WFS signals

### 2.2 METHOD 1: Provide an external simulation that adheres to AOloopControl input/output conventions

The user runs a loop that updates the wavefront sensor image when the DM input changes. Both the DM and WFS are represented as shared memory image streams. When a new DM shape is written, the DM stream semaphores are posted by the user, triggering the WFS image computation. When the WFS image is computed, its semaphores are posted.

### 2.3 METHOD 2: Physical hardware simulation

The AOsim simulation architecture relies on individual processes that simulate subsystems. Each process is launched by a bash script. ASCII configuration files are read by each process. Data I/O can be done with low latency using shared memory and semaphores: a process operation (for example, the wavefront sensor process computing WFS signals) is typically triggered by a semaphore contained in the shared memory wavefront stream. A low-speed file system based alternative to shared memory and semaphores is also provided.

Method 2 simulates incoming atmospheric WFs, a pyramid WFS based loop feeding a DM, a coronagraphic LOWFS and coronagraphic PSFs.

#### 2.3.1 Running Method 2

Launch the simulator with the following steps:

- Create a series of atmospheric wavefronts (do this only once, this step can take several hrs):

  ```
  ./aohardsim/aosimmkwf
  ```

  Stop the process when a few wavefront files have been created (approximately 10 minimum). The AO code will loop through the list of files created, so a long list is preferable to reduce the frequency at which the end-of-sequence discontinuity occurs. The current wavefront file index is displayed as the process runs; in this example, the process is working on file #2:

  ```
  Layer  0/ 7, Frame  99/ 100, File    0/100000000  [TIME =    0.0990 s]  WRITING SCIENCE WAVE
  Layer  0/ 7, Frame  99/ 100, File    1/100000000  [TIME =    0.1990 s]  WRITING SCIENCE WAVE
  Layer  1/ 7, Frame  42/ 100, File    2/100000000  [TIME =    0.2420 s]
  ```

  Type `CTRL-C` to stop the process. Note that you can relaunch the script later to build additional wavefront files.

  By default, the wavefront files are stored in the work directory. You may choose to move them to another location (useful if you have multiple work directories sharing the same wavefront files). You can then create a symbolic link `atmwf` to an existing atmospheric wavefront simulation directory. For example:

  ```
  ln -s /data/AtmWF/wdir00/ atmwf
  ```

- Execute master script `./aohardsim/runAOhsim`
- To stop the physical simulator: `./aohardsim/runAOhsim -k`

Important notes:

- Parameters for the simulation can be changed by editing the `.conf` files in the `aohardsim` directory
- You may need to kill and relaunch the main script twice after changing parameters

### 2.3.2 Method 2 output streams

| Stream | Description |
|---|---|
| **wf0opd** | Atmospheric WF OPD |
| **wf0amp** | Atmospheric WF amplitude |
| **wf1opd** | Wavefront OPD after correction [um] ( = wf0opd - 2 x dm05dispmap ) |
| **dm05disp** | DM actuators positions |
| **dm05dispmap** | DM OPD map |
| **WFSinst** | Instantaneous WFS intensity |
| **pWFSint** | WFS intensity frame, time averaged to WFS frame rate and sampled to WFS camera pixels |
| **aosim__foc0__amp** | First focal plane (before coronagraph), amplitude |
| **aosim__foc0__pha** | First focal plane (before coronagraph), phase |
| **aosim__foc1__amp** | First focal plane (after coronagraph), amplitude |
| **aosim__foc1__pha** | First focal plane (after coronagraph), phase |
| **aosim__foc2__amp** | Post-coronagraphic focal plane, amplitude |
| **aosim__foc2__pha** | Post-coronagraphic focal plane, phase |

### 2.3.3 Processes and scripts details

#### 2.3.3.1 Process `aosimmkWF`

`aosimmkWF` reads precomputed wavefronts and formats them for the simulation parameters (pixel scale, temporal sampling).

Parameters for `aosimmkWF` are stored in configuration file:

File `aosimmkWF.conf.default` :

```
!INCLUDE "../scripts/aohardsim/aosimmkWF.conf.default"
```

#### 2.3.3.2 Process `aosimDMrun`

File `aosimDMrun.conf.default` :

```
!INCLUDE "../scripts/aohardsim/aosimDMrun.conf.default"
```

#### 2.3.3.3 Process `aosimPyrWFS`

File `aosimPyrWFS.conf.default` :

```
!INCLUDE "../scripts/aohardsim/aosimPyrWFS.conf.default"
```

#### 2.3.4 AO loop control

The `aolconf` script is used to configure and launch the AO control loop. It can be configured with input/output from real hardware or a simulation of real hardware.

##### 2.3.4.1 Shared memory streams

| Script | Description |
|---|---|
| **wf0opd** | Wavefront OPD prior to wavefront correction [um] |
| **wf1opd** | Wavefront OPD after correction [um] ( = wf0opd - 2 x dm05dispmap ) |
| **dm05disp** | DM actuators positions |
| **dm05dispmap** | DM OPD map |
| **WFSinst** | Instantaneous WFS intensity |
| **pWFSint** | WFS intensity frame, time averaged to WFS frame rate and sampled to WFS camera pixels |

##### 2.3.4.2 Hardware simulation architecture

Close-loop simulation requires the following scripts to be launched to simulate the hardware, in the following order :

- `aosimDMstart`: This script creates DM channels (uses dm index 5 for simulation). Shared memory arrays `dm05disp00` to `dm05disp11` are created, along with the total displacement `dm05disp`. Also creates the `wf1opd` shared memory stream which is needed by `aosimDMrun` and will be updated by runWF. `wf1opd` is the master clock for the whole simulation, as it triggers DM shape computation and WFS image computation.
- `aosimDMrun`: Simulates physical deformable mirror (DM)
- `aosimmkWF`: Creates atmospheric wavefronts
- `aosimWFS`: Simulates WFS

Some key script variables need to coordinated between scripts. The following WF array size should match :

- `WFsize` in script `aosimDMstart`
- `ARRAYSIZE` in `aosimmkWF.conf`
- `ARRAYSIZE` in `aosimDMrun.conf`

The main hardware loop is between `aosimmkWF` and `aosimWFS`: computation of a wavefront by `aosimmkWF` is *triggered* by completion of a WFS instantaneous

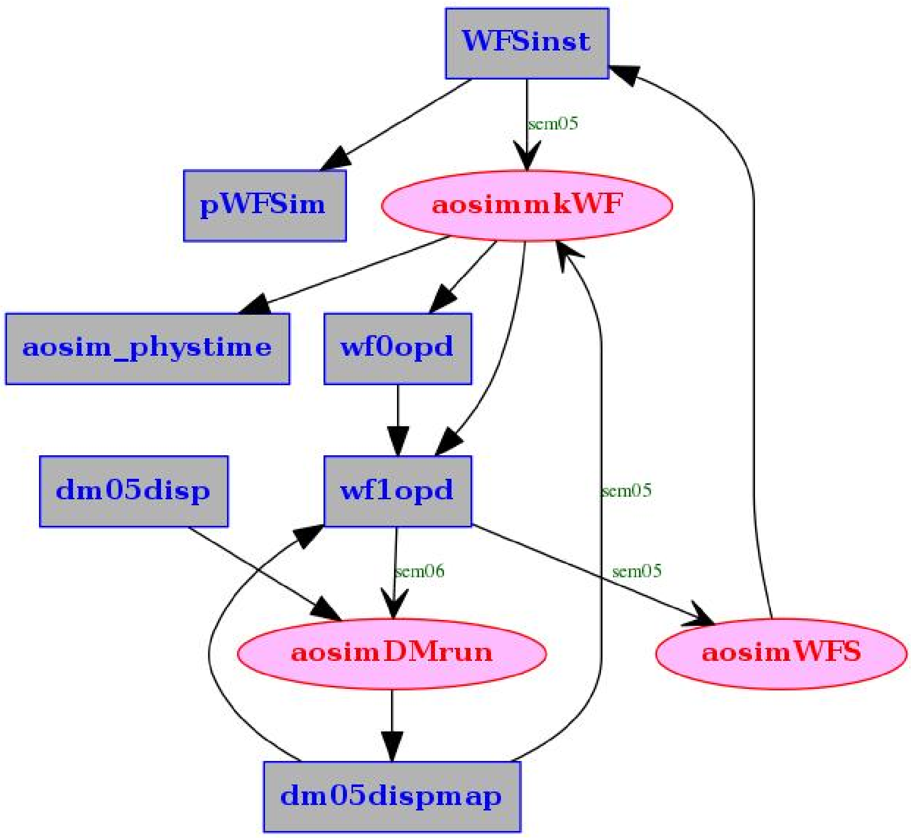


Figure 1: data flow

image computation by `aosimWFS`. The configuration files are configured for this link.

#### 2.3.4.3 DM temporal response

The DM temporal response is assumed to be such that the distance between the current position $p$ and desired displacement $c$ values is multiplided by coefficient $a < 1$ at each time step $dt$. The corresponding step response is :

$c - p((k+1)dt) = (c - p(kdt))a$

$c - p(kdt) = (c - p0)a^k$

$p(kdt) = 1 - a^k$

The corresponding time constant is

$a^{\frac{t0}{dt}} = 0.5$

$\frac{t0}{dt} ln(a) = ln(0.5)$

$ln(a) = ln(0.5)dt/t0$

$a = 0.5^{\frac{dt}{t0}}$

### 2.3.5 Processes and scripts: system ouput

The output (corrected) wavefront is processed to compute ouput focal plane images, and optionally LOWFS image.

#### 2.3.5.1 Process `aosimcoroLOWFS`

Computes coronagraphic image output and LOWFS image

File `aosimcoroLOWFS.conf.default`:

```
!INCLUDE "../scripts/aohardsim/aosimcoroLOWFS.conf.default"
```

#### 2.3.5.2 Ouput simulation architecture

## 2.4 METHOD 3: Linear Hardware Simulation

### 2.4.1 Overview

The Linear Hardware Simulation (LHS) uses a linear response matrix to compute the WFS image from the DM state. It is significantly faster than the Physical Hardware Simulation (PHS) but does not capture non-linear effects.

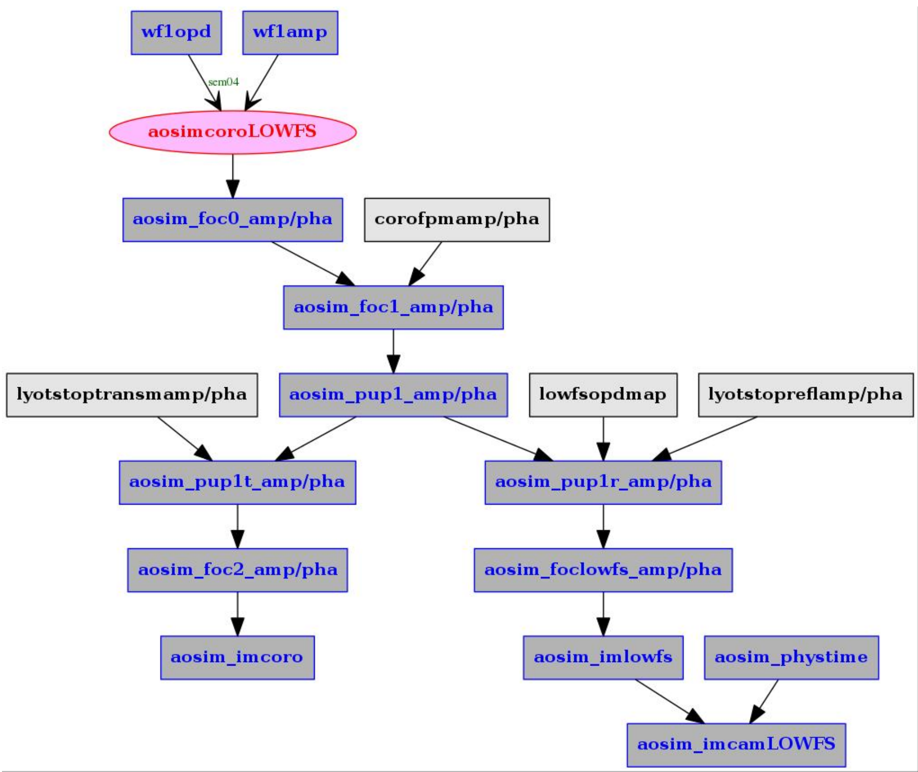


Figure 2: coroLOWFS data flow

#### 2.4.2 Setup

- Create directory LHScalib:

```
mkdir LHScalib
```

- Download response matrix and reference, place them in directory `LHScalib`.
- Start GUI, loop 5, name simLHS

```
./aolconf -L 5 -N simLHS
```

- Start DM: index 04, 50 x 50; Auto-configure: main DM (no link); STOP -> (re-)START DM comb process
- Go to `TEST MODE` GUI
- Enter linear simulation zonal response matrix and linear simulation WFS reference (`zrespMlinsim` and `wfsref0linsim` selections at top of screen).
- **Start linear simulator** (`lsimon` selection). The simulator reacts to changes in aol5_dmdisp (= dm04disp)

# 3 AOloopControl setup and overview

## 3.1 GUI description

The script `aolconf` starts the main GUI, from which all setup and control can be done. The GUI consists of several main screens, as shown below.

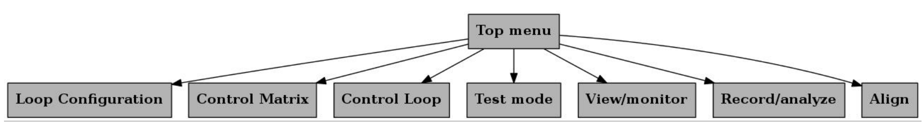


Figure 3: aolconf GUI screens

## 3.2 Commands log

#### 3.2.1 Automatically generated internal log (very detailed)

All commands are logged in an ASCII file. `aolconf` uses the script `aolconfscripts/aollog` to log into file `./logdir/<UTDATE>/logging/<LOOPNAME>.log`. A sym link to `aolconf.log` is created for convenience, so the log content can be viewed with:

```
tail -f aolconf.log
```

Inside the bash script, function `aoconflog` is used to call `aolconfscripts/aollog` with the proper loop name.

#### 3.2.2 External log (less verbose)

The user can provide a command to externally log commands. The executable should be in the path, and named `dologext`. The syntax is:

```
dologext <string>
```

The string usually consists of the loop name followed by comments.

Inside the bash script, function `aoconflogext` is used to call `aolconfscripts/aollog` with the proper loop name.

#### 3.2.3 Interactive user log

To start the interactive log script:

```
./aolconfscripts/aollog -i <LOOPNAME> NULL
```

Entries will be logged in the `./logdir/<UTDATE>/logging/<LOOPNAME>.log` file (with sym link to `aolconf.log`).

It is also common practice to start a `MISC` log for misc comments, also to be included in the external log:

```
./aolconfscripts/aollog -ie MISC NULL
```

# 4 Setting up the hardware interfaces

## 4.1 Top level script

Start aolconf with loop number and loop name (you can ommit these arguments when launching the script again):

```
./aolconf -L 3 -N testsim
```

The loop name (`testsim` in the above example) will both allocate a name for the loop and execute an optional custom setup script. The software package comes with a few such pre-made custom scripts for specific systems / examples. When the `-N` option is specified, the custom setup script `./setup/setup_<name>` is ran. The script may make some of the steps described below optional.

You can check the current loop number and name settings with:

```
./aolconf -h
```

The script can also launch a pre-written CPU/OS configuration script named `./aocscripts/cpuconfig_<LOOPNAME>` :

```
./aolconf -C
```

## 4.2 Setting the DM interface

There are four options for setting up the DM:

- [A] Connect to an existing DM
- [B] Create a new DM and connect to it
- [C] Create a new modal DM, mapped to an existing DM using another loop's control modes
- [D] Create a new modal DM, mapped to an existing DM channel using a custom set of modes

Before choosing an option, select if the DM to be controlled is `MODAL` or `ZONAL`. A zonal DM is one where the DM pixel locations map to physical actuator locations on the DM, allowing spatial filtering when creating control modes. With a zonal DM, each pixel of the DM map corresponds to a wavefront control mode, and spatial filtering functions are turned off.

Options [C] and [D] are `MODAL` options, as the DM does not represent physical spatial actuators. These options build a virtual DM which controls another DM.

### 4.2.1 Mode [A]: Connecting to an existing DM

1. **Set DM number** (`S` command in `Top Menu` screen). You should see its x and y size in the two lines below. If not, the DM does not exist yet (see next section).
2. **autoconfigure DM: main DM (nolink)** (`nolink` in `Top Menu` screen). This command automactically sets up the following symbolic links:

- dm##disp00 is linked to aol#_dmO (flat offset channel)
- dm##disp02 is linked to aol#_dmRM (response matrix actuation channel)
- dm##disp03 is linked to aol#_dmC (loop dm control channel)
- dm##disp04 is linked to aol#_dmZP0 (zero point offset 0 actuation channel)
- dm##disp05 is linked to aol#_dmZP1 (zero point offset 1 actuation channel)
- dm##disp06 is linked to aol#_dmZP2 (zero point offset 2 actuation channel)
- dm##disp07 is linked to aol#_dmZP3 (zero point offset 3 actuation channel)
- dm##disp08 is linked to aol#_dmZP4 (zero point offset 4 actuation channel)
- dm##disp is linked to aol#_dmdisp (total dm displacement channel)

3. **load Memory** (`M` in `Top Menu` screen). The dm performs the symbolic links to the DM channels.

#### 4.2.2 Mode [B]: Creating and Connecting to a DM

1. Set **DM number** (`S` command in `Top Menu` screen).
2. Enter the desired **DM size** with the `dmxs` and `dmys` commands.

- OPTIONAL: **set DM delay** (‘setDMdelayON’ and ‘setDMdelayval’ in `Top Menu` screen)

3. **Create the DM streams** with the `initDM` command in the `Top Menu`. You may need to run the `stopDM` command first.
4. **autoconfigure DM: main DM (nolink)** (`nolink` in `Top Menu` screen). This command automactically sets up the following symbolic links:
   - dm##disp00 is linked to aol#_dmO (flat offset channel)
   - dm##disp02 is linked to aol#_dmRM (response matrix actuation channel)
   - dm##disp03 is linked to aol#_dmC (loop dm control channel)
   - dm##disp04 is linked to aol#_dmZP0 (zero point offset 0 actuation channel)
   - dm##disp05 is linked to aol#_dmZP1 (zero point offset 1 actuation channel)
   - dm##disp06 is linked to aol#_dmZP2 (zero point offset 2 actuation channel)

- dm##disp07 is linked to aol#_dmZP3 (zero point offset 3 actuation channel)
- dm##disp08 is linked to aol#_dmZP4 (zero point offset 4 actuation channel)
- dm##disp is linked to aol#_dmdisp (total dm displacement channel)

5. **Load Memory** (`M` in `Top Menu` screen). The dm performs the symbolic links to the DM channels.

### 4.2.3 Mode [C]: Create a new modal DM, mapped to an existing DM using another loop's control modes

In this mode, the AO loop controls a virtual DM. The virtual actuators are correspond to modes controlling the zero point offset of another loop. In this section, I assume that **loopA** is the main loop (directly controls a physical DM) and that **loopB** is the virtual loop (this is the loop we are setting up).

1. Select **MODAL** DM (`DMmodeZ` in `Top Menu` screen)
2. Set **DM number** (`S` command in `Top Menu` screen). This is the DM index for loopB.
3. Set **DM x size** to the number of modes of loop A to be addressed by loop B's virtual DM
4. Set **DM y size** to 1
5. **Auto-configure: DM output linked to other loop** (`dmolink` in `Top Menu` screen).
   1. choose loop index from which modes will be extracted (loop A index)
   2. choose offset channel in output loop This will set up several key parameters and files:

   - **DM-to-DM** mode will be set to 1, and associated streams:
     - **dm2dmM** : **loopA** modes controlled by **loopB**
     - **dm2dmO** : symbolic link to **loopA** DM channel controlled by **loopB**
   - **CPU-based dmcomb output WFS ref** will be set to 1, and associated streams:
     - **dmwrefRM** : **loopA** WFS response to modes controlled by **loopB**
     - **dmwrefO** : **loopA** WFS zero point offset

- **OPTIONAL: set DM delay** ('setDMdelayON' and 'setDMdelayval' in `Top Menu` screen)

6. **Create the DM streams** with the `initDM` command in the `Top Menu`.
7. **Load Memory** (`M` in `Top Menu` screen). The dm performs the symbolic links to the DM channels.

#### 4.2.4 Mode [D]: Create a new modal DM, mapped to an existing DM channel using a custom set of modes

In this mode, the AO loop controls a virtual DM. The virtual actuators correspond to modes controlling another DM stream. In this section, I assume that **loop A** is the main loop (directly controls a physical DM) and that **loop B** is the virtual (higher level) loop.

1. Choose DM index number (`S`) for loop B
2. Select number of loop A modes controlled by loop B. The number is entered as DM x size (`dmxs` in `Top menu`)
3. Enter 1 for DM y size (`dmys` in `Top menu`)
4. Set **DM-to-DM** mode to 1, and associated streams:
   - **dm2dmM** : loop A modes controlled by loop B
   - **dm2dmO** : symbolic link to loop A DM channel controlled by loop B
5. Set **CPU-based dmcomb output WFS ref** to 0 (see section below more enabling this option)
6. **(Re)-create DM streams and run DMcomb process** (`initDM`)
7. **Load Memory** (`M` in `Top Menu` screen). The dm performs the symbolic links to the DM channels.

Commands to the loop B DM should now propagate to modal commands to loop A.

#### 4.2.5 Option: WFS Zero point offset

It is possible to add a zero point offset to mode D. Every write to the loop B's modal DM then generate both a write to loop A's DM (described above) and a write to the reference of a wavefront sensor (presumably loop A's wavefront sensor). This optional feature is refered to as a CPU-based WFS zero point offset.

To enable this feature, add between steps 4 and 5:

1. set **CPU-based dmcomb output WFS ref** to 1, and associated streams:
   - **dmwrefRM** : **loopA** WFS response to modes controlled by **loopB**
   - **dmwrefO** : **loopA** WFS zero point offset

#### 4.2.6 Notes

You can (Re-)Start DM comb to re-initialize arrays and links ('stopDM' and 'initDM' commands in `Top Menu` screen). The `initDM` command will

- (re-)create shared memory streams dm##disp00 to dm##disp11
- start the dmcomb process, which adds the dm##disp## channels to create the overall dm##disp displacement
- create poke mask and maps

### 4.3 Setting the camera interface

- **link to WFS camera** (`wfs` to `Loop Configuration` screen). Select the WFS shared memory stream.

### 4.4 Setup script

An `aosetup` script may be used to perform all these operations. Inspect the content of directory `aosetup` to see such scripts. You may use or modify as needed. If you use a `aosetup` script, execute it from the working directory, and then start aolconf:

```
./aosetup/aosetup_<myLoop>
./aolconf
```

## 5 Calibration

### 5.1 Acquiring a zonal response matrix

- **set response matrix parameters** in `Loop Configure` screen: amplitude, time delay, frame averaging, excluded frames
- **set normalization and Hadmard modes** in `Loop Configure` screen. Normalization should probably be set to 1.
- **start zonal response matrix acquisition** (`zrespon` in `Loop Configure` screen). The process runs in tmux session aol#zrepM.

- **stop zonal response matrix acquistion** (`zrespoff` in `Loop Configure` screen).

The following files are then created:

| File | Archived location | Description |
|---|---|---|
| **zrespmat.fits** | zrespM/zrespM_${datestr}.fits | zonal response matrix |
| **wfsref0.fits** | wfsref0/wfsref0_${datestr}.fits | WFS reference (time-averaged image) |
| **wfsmap.fits** | wfsmap/wfsmap_${datestr}.fits | Map of WFS elements sensitivity |
| **dmmap.fits** | dmmap/dmmap_${datestr}.fits | Map of DM elements sensitivity |
| **wfsmask.fits** | wfsmask/wfsmask_${datestr}.fits | WFS pixel mask, derived from wfsmap |
| **dmmaskRM.fits** | dmmaskRM/dmmaskRM_${datestr}.fits | DM actuator mask, derived from dmmap by selecting actuators with strong response |
| **dmslaved.fits** | dmslaved/dmslaved_${datestr}.fits | slaved DM actuators: actuators near active actuators in dmmaskRM |
| **dmmask.fits** | dmmask/dmmask_${datestr}.fits | DM mask: all actuators controlled (union of dmmaskRM and dmslaved) |

Note that at this point, the files are NOT loaded in shared memory, but the archieved file names are stored in the staging area "conf_zrm_staged/conf_streamname.txt" for future loading.

- **Adopt staged configuration** (`upzrm` in `Loop Configure` screen)
- **Load zrespm files into shared memory** (`SMloadzrm` in `Loop Configure` screen)

## 5.2 Acquiring a modal response matrix (optional, for ZONAL DM only)

In addition to the zonal response matrix, a modal response matrix can be acquired to improve sensitivity to low-oder modes.

To do so:

- activate `RMMon` to **toggle the modal RM on**.
- **select RM amplitude and maximum cycles per aperture (CPA)**

- **start the acquisiton** (`LOresp_on`)
- **stop the acquisiton** (`LOresp_off`)

The following files are then created:

| File | Archived location | Description |
|---|---|---|
| **LOrespmat.fits** | LOrespM/LOrespM_${datestr}.fits | Modal response matrix |
| **respM_LOmodes.fits** | LODMmodes/LODMmodes_${datestr}.fits | Low-order modes |
| **LOwfsref0.fits** | LOwfsref0/LOwfsref0_${datestr}.fits | WFS reference measured during LO RM acquisition |
| **LOwfsmap.fits** | LOwfsmap/LOwfsmap_${datestr}.fits | Map of WFS elements sensitivity |
| **LOdmmap.fits** | LOdmmap/LOdmmap_${datestr}.fits | Map of DM elements sensitivity |
| **LOwfsmask.fits** | LOwfsmask/LOwfsmask_${datestr}.fits | WFS pixel mask, derived from wfsmap |
| **LOdmmask.fits** | LOdmmask/LOdmmask_${datestr}.fits | DM actuator mask, derived from dmmap by selecting actuators with strong response |

Note that at this point, the files are NOT loaded in shared memory, but the archieved file names are stored in the staging area "conf_mrm_staged//conf_streamname.txt" for future loading.

- **Adopt staged configuration** (`upmrm` in `Loop Configure` screen)
- **Load LOrespm files into shared memory** (`SMloadmrm` in `Loop Configure` screen)

## 5.3 Automatic system calibration (recommended)

The automatic system calibration performs all steps listed above under zonal and modal response matrix acquisition.

The old calibrations are archived as follows:

- "conf_zrm_staged" and "conf_mrm_staged" hold the new configuration (zonal and modal respectively)
- "conf_zrm_staged.000" and "conf_mrm_staged.000" hold the previous configuration (previously "conf_zrm_staged" and "conf_mrm_staged")
- "conf_zrm_staged.001" and "conf_mrm_staged.001" hold the configuration previously named "conf_zrm_staged.000" and "conf_mrm_staged.000"
- etc for a total of 20 configuration

### 5.4 Managing configurations

At any given time, the current configuration (including control matrices if they have been computed) can be saved using the `SAVE CURRENT SYSTEM CALIBRATION` command. Saving a configuration will save all files in the conf directory into a user-specified directory.

Previously saved configurations can be loaded with the `LOAD SAVED SYSTEM CALIBRATION` command. This will load saved files into the conf directory and load all files into shared memory.

## 6 Building control matrix

- **set SVDlimit** (`SVDla` in `Control Matrix` screen). Set value is 0.1 as a starting point for a stable loop.
- **perform full CM computation** (`mkModes0` in `Control Matrix` screen). Enter first the number of CPA blocks you wish to use. Computation takes a few minutes, and takes place in tmux session `aol#mkmodes`.

The following files are created:

| File | Archived location | Description |
|---|---|---|
| **aolN_DMmodes** | DMmodes/DMmodes_${datestr}.fits | DM modes |
| **aolN_respM** | respM/respM_${datestr}.fits | WFS response to DM modes |

Block-specific files:

| File | Archived location | Description |
|---|---|---|
| **aolN_DMmodesbb** | DMmodes/DMmodesbb_${datestr}.fits | DM modes for block bb |
| **aolN_respMbb** | respM/respMbb_${datestr}.fits | WFS response to DM modes for block bb |
| **aolN_contrMbb.fits** | contrM/contrMbb_${datestr}.fits | Control matrix for block bb |
| **aolN_contrMcbb.fits** | contrMc/contrMcbb_${datestr}.fits | Collapsed control matrix for block bb |
| **aolN_contrMcactbb.fits** | contrMcact/contrMcactbb_${datestr}.fits | Collapsed control matrix for block bb, only active actuators |

Note that at this point, the files are NOT loaded in shared memory, but the archieved file names are stored in “conf/conf_.txt” for future loading.

- **Load CM files into shared memory** (`SMloadCM` in `Control Matrix` screen)

# 7 Running the loop: Choosing hardware mode (CPU/GPU)

There are multiple ways to perform the computations on CPU and/or GPUs. The main 3 parameters are:

- **GPU** : 0 if matrix multiplication(s) done on CPU, >0 for GPU use. This is the number GPUs to use for matrix mult.
- **CMmode** : 1 if using a combined matrix between WFS pixels and DM actuators, skipping intermediate computation of modes
- **GPUall** : if using GPUall, then the WFS reference subtraction is wrapped inside the GPU matrix multiplication

| GPU | CMmode | GPUall | Matrix | Features | Description |
|---|---|---|---|---|---|
| >0 | ON | ON | contrMcact | fastest no mcoeff | dark-subtracted WFS frame imWFS0 is multiplited by collapsed control matrix (only active pixels). normalization and WFS reference subtraction are wrapped in this GPU operation as subtraction of pre-computed vector output. This is the fastest mode. |
| >0 | ON | OFF | contrMcact | | WFS reference is subtracted from imWFS0 in CPU, yielding imWFS2. imWFS2 is multiplied by control matrix (only active pixels) in GPU. |
| >0 | OFF | OFF | contrM | | MWFS reference is subtracted from imWFS0 in CPU, yiedling imWFS2. imWFS2 is multiplied (GPU) by control matrix to yield mode values. Mode coefficients then multiplied (GPU) by modes. |
| 0 | ON | - | contrMcact | | imWFS2 is multiplied by control matrix (only active pixels) in CPU |
| 0 | OFF | - | contrM | | imWFS2 multiplied by modal control matrix |

# 8 Auxilliary processes

A number of auxilliary processes can be running in addition to the main loop operation.

## 8.1 Extract WFS modes

Launches script `./auxscripts/modesextractwfs` :

```
!INCLUDE "../scripts/auxscripts/modesextractwfs"
```

Converts WFS residuals into modes.

## 8.2 Extract open loop modes

Launches script C function (CPU-based):

```
key       :    aolcompolm
module    :    AOloopControl.c
info      :    compute open loop mode values
syntax    :    <loop #>
example   :    aolcompolm 2
C call    :    long AOloopControl_ComputeOpenLoopModes(long loop)
```

This function is entirely modal, and assumes that the WFS modes (see section above) are computed. The key input to the function is `aolN_modeval`, the WFS residual mode values. The function uses this telemetry and knowledge of loop gain and mult factor to track open loop mode values.

Optionally, it also includes `aolN_modeval_pC`, the predictive control mode values that are added to the correction in predictive mode.

## 8.3 Running average of dmC

Launches script `./auxscripts/aol_dmCave 0.0005` :

```
!INCLUDE "../scripts/auxscripts/aol_dmCave"
```

## 8.4 Compute and average wfsres

Launches script `./auxscripts/aolmkWFSres 0.0005` :

```
!INCLUDE "../scripts/auxscripts/aolmkWFSres"
```

# 9 Offsetting

## 9.1 Overview

Input channels are provided to offset the AO loop convergence point. By default, **DM channels 04, 05, 06, 07, and 08 are dedicated to zero-point offsetting**. The DM channels are sym-linked to `aolN_dmZP0` - `aolN_dmZP7`.

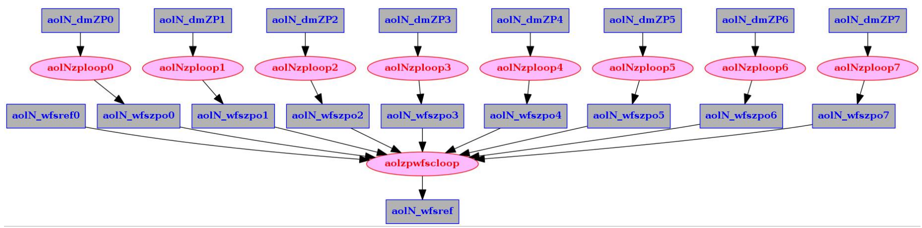


Figure 4: WFS zero point offsetting

## 9.2 DM offsets

### 9.2.1 Zonal CPU-based zero point offset

CPU-based zero point offsets will compute WFS offsets from the zero point offset DM channels (04-11) and apply them to the `aolN_wfsref` stream. To activate this features, the user needs to :

- **Toggle the zero point offset loop process ON** (`LPzpo`) prior to starting the loop.

Cfits command `aolzpwfscloop` (C function `AOloopControl_WFSzeropoint_sum_update_loop`) launches a loop that monitors shared memory streams `aolN_wfszpo0` to `aolN_wfszpo3`, and updates the WFS reference when one of these has changed. The loop is running insite tmux session `aolNwfszpo`, and is launched when the loop is closed (`Floopon`) if the loop zero point offset flag is toggled on (`LPzpo`)

- **Activate individual zero point offset channels** (`zplon0` to `zplon4`).

Every time one of the activated DM channel changes, the corresponding wfs `aolN_wfszpo#` zero point offset is CPU-computed.

### 9.2.2 GPU-based zero point offset

A faster GPU-based zero point offset from DM to WFS is provided for each of the 8 offset channels. GPU-based and CPU-based offsetting for a single channel are mutually exclusive.

### 9.3 WFS offsets

## 10 Controlling offsets from another loop

### 10.1 Running the loop

The next steps are similar to the ones previously described, with the following important differences:

- The control matrix should be computed in zonal mode (no modal CPA block decomposition)

## 11 Predictive control (experimental)

### 11.1 Overview

Predictive control is implemented in two processes:

- The optimal auto-regressive (AR) filter predicting the current state from previous states is computed. The AR filter is computed from open-loop estimates, so the processes computing open-loop telemetry need to be running.
- the AR filter is applied to write a prediction buffer, which can be written asynchronously from the main loop steps.

The predictive filter is modal, and adopts the same modes as the main control loop.

### 11.2 Scripts

| File | Description |
|---|---|
| **aolARPF** | find auto-regressive predictive filter |
| **aolARPFblock** | AO find optimal AR linear predictive filter |

### 11.3 Data flow

Predictive control is set up by blocks of modes. A block is configured through the aolconf predictive control sub-panel, which writes to configuration files `conf/conf_PFblock_XXX.txt`, where XXX is the block number (000, 001, 002 etc…). Configuration files specify the modes within each block (index min to index max), the predictive filter order, time lag and and averaging gain.

For each block, there are 3 main processes involved in running the predictive control:

- **Watching input telemetry** this process listens to the input telemetry stream and periodically writes data to be used to compute a filter. This runs function `long AOloopControl_builPFloop_WatchInput(long loop, long PFblock)` in `AOloopControl.c`.
- **Computing filter**. Runs CLI command `mkARpfilt`, which runs function `LINARFILTERPRED_Build_LinPredictor` in `linARfilterPred.c`.
- **Prediction engine** (= apply filter). Runs script `./auxscripts/predFiltApplyRT`.

All 3 processes work in a chain, and can be turned on/off from the GUI.

# 12 REFERENCE, ADDITIONAL PAGES

- Page @subpage streams_semaphores
- Page @subpage streams_semaphores
- Page @subpage streams_semaphores