

AO Loop Control Software

Overview

Linux-based

Open-source, no closed library

C code (~100k lines) + high-level scripts (baseline control interface using bash scripts provided)

Uses libraries: CUDA & MAGMA (GPU computing, optional), FFTW, FITSIO, GNU scientific library, readline

Source code + example simulated AO system:

<https://github.com/oguyon/AdaptiveOpticsControl>

Hardware

Hardware Requirements / compatibility:

RTS runs on a single multi-core computer. Minimum ~15 cores system, 128GB ram (heavy use of shared memory and shielded processes running on single core)

CPU only or CPU+GPU computing engine. Requires GPU(s) for high speed / high actuator count. Supports NVIDIA hardware (CUDA lib).

Can span multiple computers (for example, camera or DM driven by computer other than main RTS). Software uses and configures fast private low-latency TCP link (eg. 10GbE or 40GbE fibers) for transfers.

Interfaces to hardware through shared memory structure. Hardware already coupled with RTS: BMC deformable mirror, Ocam2k camera, SAPHIRA camera (with UH readout electronics), OwlCam InGaAs Raptor Photonics camera, Andor sCMOS.

Capabilities

Speed

Largely limited by hardware. Fully system timing stable at 10us level, and RTS latency due to IPC, TCP transfers between computers, and GPU transfers is <100us total → can drive ~10 kHz loop on multi-computer system, and ~20 kHz loop on single computer. SCEXAO implementation drives 2000-actuator, 14,400-sensor loop at 3.5kHz, limited by camera readout speed.

Flexible architecture

All input, output and intermediate data is stored as shared memory. A common format for all shared memory data streams facilitates software development. Multiple processes run simultaneously to perform operations on shared memory streams. Additional processes can be deployed (for example, real-time analysis of an intermediate data stream) without impacting existing processed.

IPC is built in the shared memory structure which contains POSIX semaphores (default of 10 semaphores, more if needed): 10 different processes can run on the same input. Each process waits on input stream(s), and posts output stream(s) semaphore(s) → Real-time operations can be chained, with multiple branches

Example control GUI (bash scripts)

```
TOP MENU
[Active conf = ] [ Mon Apr 10 12:48:10 UTC 2017 ]

DM CHANNELS AND OUTPUT (dcomb process)

S [00] Set DM index
dmax [50] Set DM x size (if modal control, = number of modes)
dms [50] Set DM y size (if modal control)

noLink Auto-configure: main DM (no link) -> DM actuators are physical actuators
dnoLink Auto-configure: DM output linked to other loop -> DM actuators represent modes

DmodeM DM is ZONAL Modes constructed from spatial DM actuators (select to toggle to MODAL)

d2dmModel [ OFF ] DM-to-DM is OFF (select to activate virtual (modal) DM to physical DM mode)

dmuref1 [ OFF ] CPU-based dcomb output WFS ref is OFF (select for DM output applied as WFS offset)
dmurefM [ MISSING ] WFS Resp Matrix aol0_dmurefM -> empty
dmurefD [ MISSING ] WFS zp output stream aol0_dmurefD -> empty

dmvlt0 [ ON ] De-activate DM volt output [-> dmvolt]

dcombave [0] DM combination averaging mode

setDMdelayval [0] Set DM delay value [us]
setDMdelayDM [DM delay is OFF] press to toggle DM delay to ON state

initDM (re)-START DM comb process (-> dm0disp00..07 dm0disp)

2 -> AD CONFIGURE AND CONTROL

C0 CALIBRATE SYSTEM (CPAmax = 22.0) RM, CH -> staged (compute masks)
C01 CALIBRATE SYSTEM (CPAmax = 22.0) RM -> staged (Re-use masks)
C01 RM -> CH (staged)
AD ADOPT CALIBRATION: staged -> conf, SharedMem

M load all (Memory)
C (Configure/Link AO loop
CH Modes and Control Matrix

L Control AO (Loop)

3 -> PREDICTIVE CONTROL

P Predictive Control
F Filtering

4 -> TEST AND MONITOR

L List running AO processes, locks
T Test mode: simulated AO system
V View / monitor

5 -> DATA LOGGING / ANALYSIS

R Record / analyze

6 -> CUSTOM EXTERNAL SCRIPTS

A Align
MC Hardware Control

< Select > < Exit >
```

```
ALIGNMENT - LOOP SCEAOPyWFS (0)

TT loop is : OFF
Pcam loop is : OFF
PyF filter : 3

1 -> Pyramid modulation

pyfr05 freq = 0.5 kHz
pyfr10 freq = 1.0 kHz
pyfr15 freq = 1.5 kHz
pyfr20 freq = 2.0 kHz
pyfr25 freq = 2.5 kHz
pyfr30 freq = 3.0 kHz
pyfr35 freq = 3.5 kHz

pymoda05 modulation amplitude = 0.05 (modulation radius = 12.5 mas)
pymoda10 modulation amplitude = 0.10 (modulation radius = 25.0 mas)
pymoda15 modulation amplitude = 0.15 (modulation radius = 37.5 mas)
pymoda20 modulation amplitude = 0.20 (modulation radius = 50.0 mas)
pymoda25 modulation amplitude = 0.25 (modulation radius = 62.5 mas)
pymoda30 modulation amplitude = 0.30 (modulation radius = 75.0 mas)
pymoda35 modulation amplitude = 0.35 (modulation radius = 87.5 mas)
pymoda40 modulation amplitude = 0.40 (modulation radius = 100.0 mas)
pymoda45 modulation amplitude = 0.45 (modulation radius = 112.5 mas)
pymoda50 modulation amplitude = 0.50 (modulation radius = 125.0 mas)
pymoda55 modulation amplitude = 0.55 (modulation radius = 137.5 mas)
pymoda60 modulation amplitude = 0.60 (modulation radius = 150.0 mas)
pymoda65 modulation amplitude = 0.65 (modulation radius = 162.5 mas)
pymoda70 modulation amplitude = 0.70 (modulation radius = 175.0 mas)
pymoda75 modulation amplitude = 0.75 (modulation radius = 187.5 mas)
pymoda80 modulation amplitude = 0.80 (modulation radius = 200.0 mas)
pymoda85 modulation amplitude = 0.85 (modulation radius = 212.5 mas)
pymoda90 modulation amplitude = 0.90 (modulation radius = 225.0 mas)
pymoda95 modulation amplitude = 0.95 (modulation radius = 237.5 mas)
pymoda100 modulation amplitude = 1.00 (modulation radius = 250.0 mas)

pyfilt1 PyWFS filter 1 (Open)
pyfilt2 PyWFS filter 2 (700 nm BW)
pyfilt3 PyWFS filter 3 (850 nm BW)
pyfilt4 PyWFS filter 4 (750 nm, 50 nm BW)
pyfilt5 PyWFS filter 5 (850 nm, 25 nm BW)
pyfilt6 PyWFS filter 6 (850 nm, 40 nm BW)

pypick01 PyWFS pickoff 01 (Open)
pypick02 PyWFS pickoff 02 (Silver mirror)
pypick03 PyWFS pickoff 03 (50/50 splitter)
pypick04 PyWFS pickoff 04 (650 nm SP)
pypick05 PyWFS pickoff 05 (700 nm SP)
pypick06 PyWFS pickoff 06 (750 nm SP)
pypick07 PyWFS pickoff 07 (800 nm SP)
pypick08 PyWFS pickoff 08 (850 nm SP)
pypick09 PyWFS pickoff 09 (750 nm LP)
pypick10 PyWFS pickoff 10 (800 nm LP)
pypick11 PyWFS pickoff 11 (850 nm LP)
pypick12 PyWFS pickoff 12 (Open)

2 -> Pyramid TT align ( 90.3 mas/V )

Current position ( scale = 90.3 mas/V ) = -5.623 -4.403
tz Zero TT align (-5.0 -5.0)
ttr Move to TT reference position [ -5.268 -4.801 ]
tts Store current position as reference
ts10 alignment step = 0.05
ts11 alignment step = 0.1
ts12 alignment step = 0.2
ts13 alignment step = 0.5
ttx TT x -0.2 (PyWFS bottom left)
txp TT x +0.2 (PyWFS top right)
txm TT x -0.2 (PyWFS top left)
txp TT x +0.2 (PyWFS bottom right)
ts ===== Start TT align =====
tp py TT loop gain = 0.1
tm Monitor TT align txmx session

3 -> Pyramid Camera Align ( 5925 steps / pix )

pz Zero Pcam align ( 143736 60198 )
ps10 alignment step = 50000
ps11 alignment step = 10000
ps12 alignment step = 3000
ps13 alignment step = 1000
psm Pcam x -10000 (right)
psp Pcam x +10000 (left)
psm Pcam y -10000 (top)
psp Pcam y +10000 (bottom)
ps ===== Start Pcam align =====
pg Pcam loop gain = 0.4
pm Monitor Pcam align txmx session

4 -> DM Flatten

f1 Flatten DM for pyWFS
f1x End Flatten DM process
f1x Remove Flatten DM solution
f1x Apply Flatten DM solution
f1x Monitor DM flatten txmx session

< Top > < Exit >
```

For each file:
conf_<name>_name.txt points to archived file location

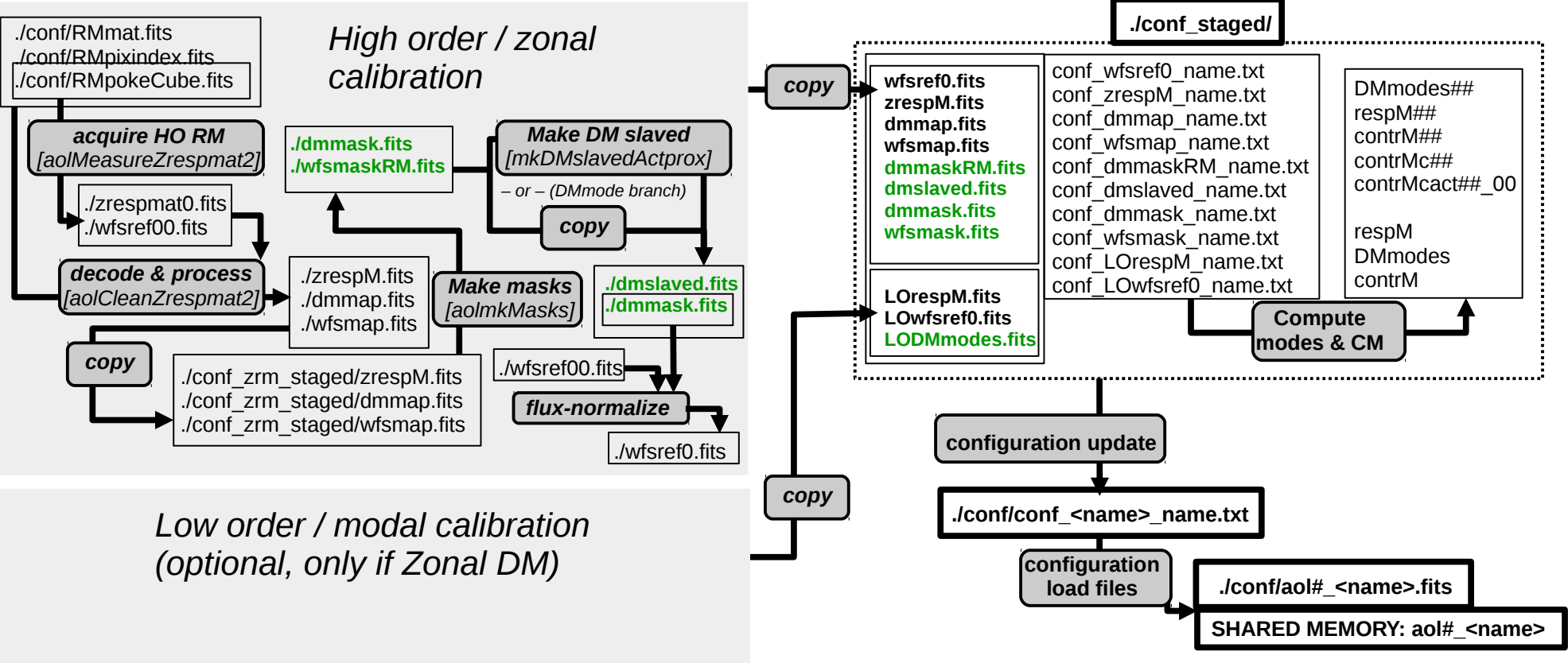
conf/conf_<name>_name.txt are read by function ReadConfFile for loading into shared memory and FITS copy to
./conf/aol#_<name>.fits

Calibration Work Flow

Conventions :

- Modal DM:** “actuators” indices have no spatial meaning
- No spatial filtering options
 - “Direct write” CM and “Modal” CM are the same (1 mode = 1 actuator)
- Zonal DM:** actuator indices correspond to spatial coordinates
- Need linear transformation between mode coefficients and actuators

If re-using masks, keep from previous calibration



Control Matrix Computation Modes

WFSnorm (*./conf/param_WFSnorm.txt*) **WFS normalization mode** C code: WFSnormalize

0: Do not normalize WFS images

1: Normalize WFS images (default)

WFSnorm should be left unchanged between RM acquisition and Loop control

DMprimaryWrite (*./conf/param_DMprimaryWriteON.txt*) **DM primary write** C code: DMprimaryWriteON

0: DM primary write is off (default)

1: DM primary write is on

DMfilteredWrite (*./conf/param_DMfilteredWriteON.txt*) **DM filtered write** C code: DMfilteredWriteON

0: DM filtered write is off (default)

1: DM filtered write is on

CMMODE (*./conf/param_CMMODE.txt*) **Combined Control matrix mode** C code: CMMODE

0: (not combined): control matrix is WFS pixels → modes

→ **Linking aol#_DMmode_meas** ↔ **aol#_modeval**

→ modesextractwfs reads from aol_DMmode_meas instead of computing

1: (combined): control matrix is WFS pixels → DM actuators

DMMODE (*./conf/param_DMMODE.txt*) **DM mode (zonal vs. modal)** C code: DMMODE

0 (**ZONAL**): pixel coordinates correspond to DM actuators physical location

→ spatial filtering enabled for DM modes creation

→ blocks built by spatial frequencies, user can set independent gain values for mode blocks

1 (**MODAL**): DM pixels correspond to abstract modes

→ no spatial filtering, setting 1 block only

Note: **DMMODE**=0 → **CMMODE**=0 (CPA-splitting of modes into blocks)

GPU (*./conf/param_GPU.txt*) **# of GPUs to use for CM multiplication** C code: AOconf[loop].GPU

0: use CPU

1+: number of GPUs

if CMmode=1 and GPUmode>0:

GPUall (*./conf/param_GPUall.txt*) **Use GPU for all computations** C code: AOconf[loop].GPUall = COMPUTE_GPU_SCALING

0: Use CPU for WFS reference subtraction and normalization

→ WFS reference subtraction and normalization done by CPU (imWFS0 → imWFS1 → imWFS2)

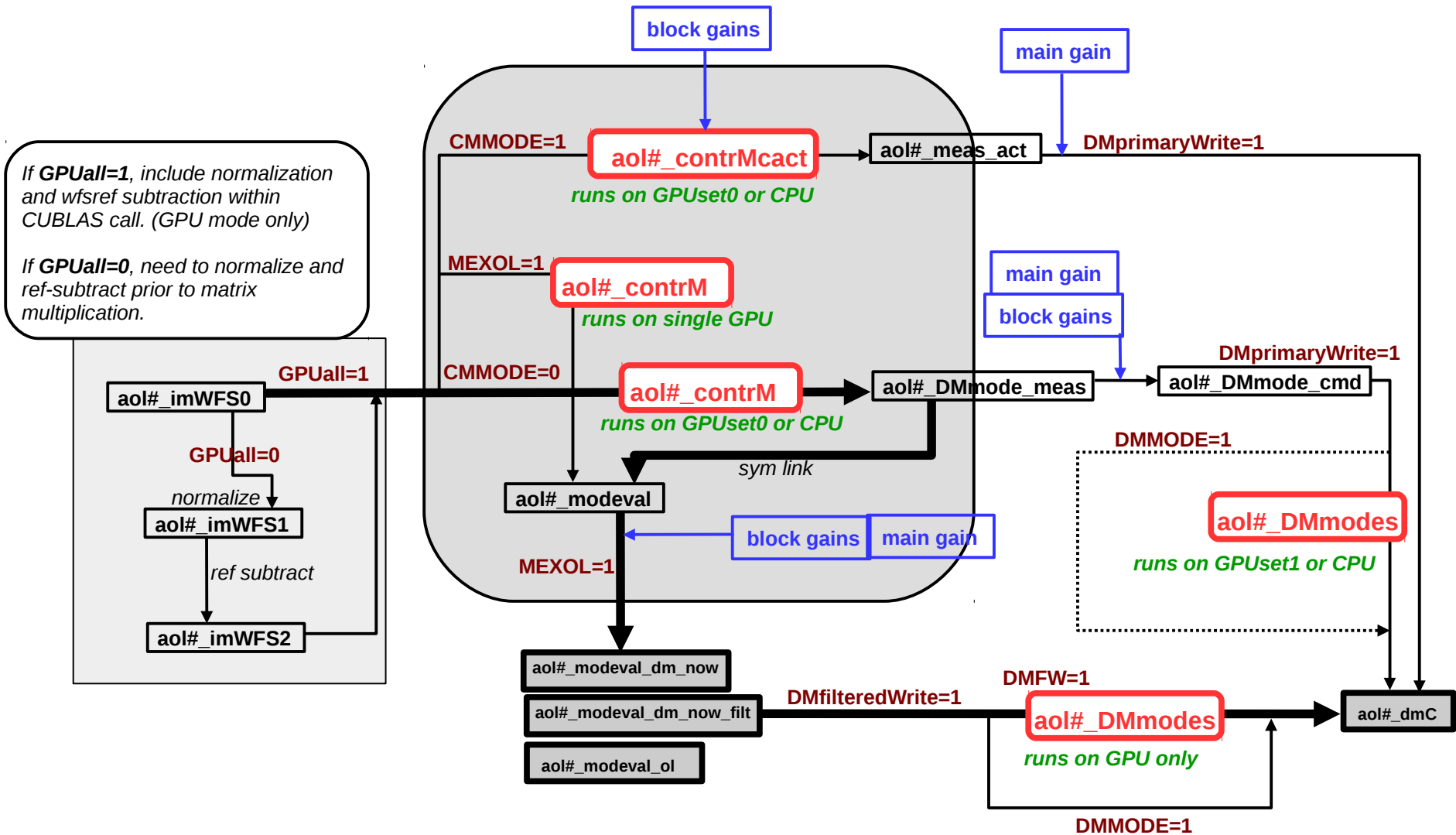
→ CM multiplication input is imWFS2 (GPU or CPU)

1: Use GPU for all computation

→ WFS reference subtraction and normalization done by GPU

→ GPU-based CM multiplication input is imWFS0

Computing Modes and Control Matrices



Supported Computing Modes

GPUset0=0 → GPUall=OFF



CMMODE	GPUset0	GPUall	DMprimaryWrite	GPUset1	DMFWB	Comments
0 (modal control)	0 (CPU mode)	OFF	0	N/A	0	Note: DOES NOTE WRITE TO DM
0 (modal control)	0 (CPU mode)	OFF	0	N/A	1	OK
0 (modal control)	0 (CPU mode)	OFF	1	0 (CPU mode)	0	OK
0 (modal control)	0 (CPU mode)	OFF	1	0 (CPU mode)	1	Note: 2 DM writes per iteration
0 (modal control)	0 (CPU mode)	OFF	1	>0 (GPU mode)	0	OK
0 (modal control)	0 (CPU mode)	OFF	1	>0 (GPU mode)	1	Note: 2 DM writes per iteration
0 (modal control)	>1 (GPU mode)	OFF	0	N/A	0	Note: DOES NOTE WRITE TO DM
0 (modal control)	>1 (GPU mode)	OFF	0	N/A	1	OK
0 (modal control)	>1 (GPU mode)	OFF	1	0 (CPU mode)	0	OK
0 (modal control)	>1 (GPU mode)	OFF	1	0 (CPU mode)	1	Note: 2 DM writes per iteration
0 (modal control)	>1 (GPU mode)	OFF	1	>0 (GPU mode)	0	OK
0 (modal control)	>1 (GPU mode)	OFF	1	>0 (GPU mode)	1	Note: 2 DM writes per iteration
0 (modal control)	>1 (GPU mode)	ON	0	N/A	0	Note: DOES NOTE WRITE TO DM
0 (modal control)	>1 (GPU mode)	ON	0	N/A	1	PREFERRED MODE FOR GPU-BASED MODAL ExAO PRIMARY LOOP
0 (modal control)	>1 (GPU mode)	ON	1	0 (CPU mode)	0	OK
0 (modal control)	>1 (GPU mode)	ON	1	0 (CPU mode)	1	Note: 2 DM writes per iteration
0 (modal control)	>1 (GPU mode)	ON	1	>0 (GPU mode)	0	OK
0 (modal control)	>1 (GPU mode)	ON	1	>0 (GPU mode)	1	Note: 2 DM writes per iteration
1 (zonal control)	0 (CPU mode)	OFF	0	N/A	0	Note: DOES NOTE WRITE TO DM (→ aol0_meas_act)
1 (zonal control)	0 (CPU mode)	OFF	1	0 (CPU mode)	0	OK
1 (zonal control)	0 (CPU mode)	OFF	1	>0 (GPU mode)	0	OK
1 (zonal control)	>1 (GPU mode)	OFF	0	N/A	0	Note: DOES NOTE WRITE TO DM
1 (zonal control)	>1 (GPU mode)	OFF	1	N/A	0	OK
1 (zonal control)	>1 (GPU mode)	ON	0	N/A	0	Note: DOES NOTE WRITE TO DM
1 (zonal control)	>1 (GPU mode)	ON	1	N/A	0	OK

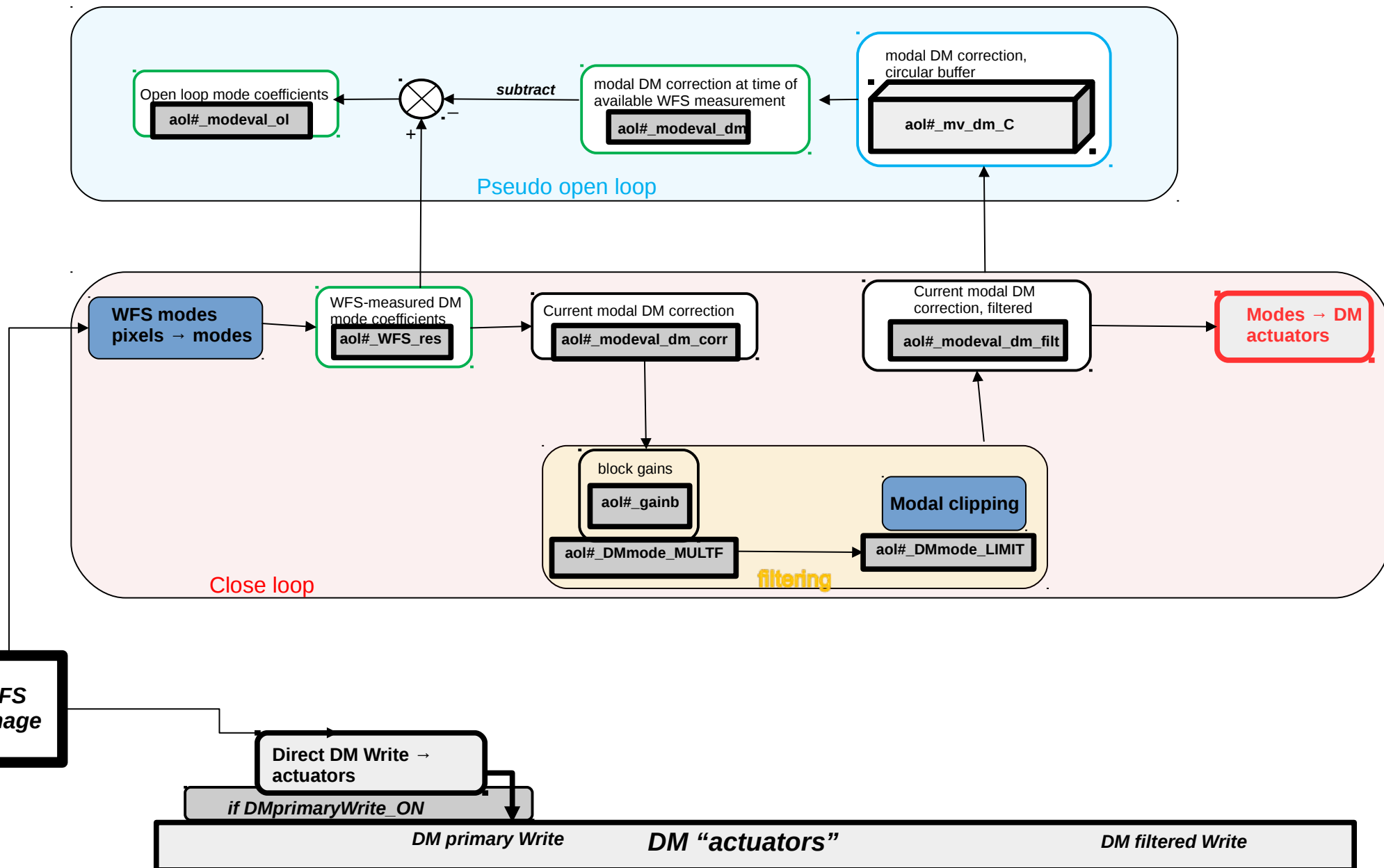
Green: GPU computation
Gray : CPU computation

Open & pseudo open loop

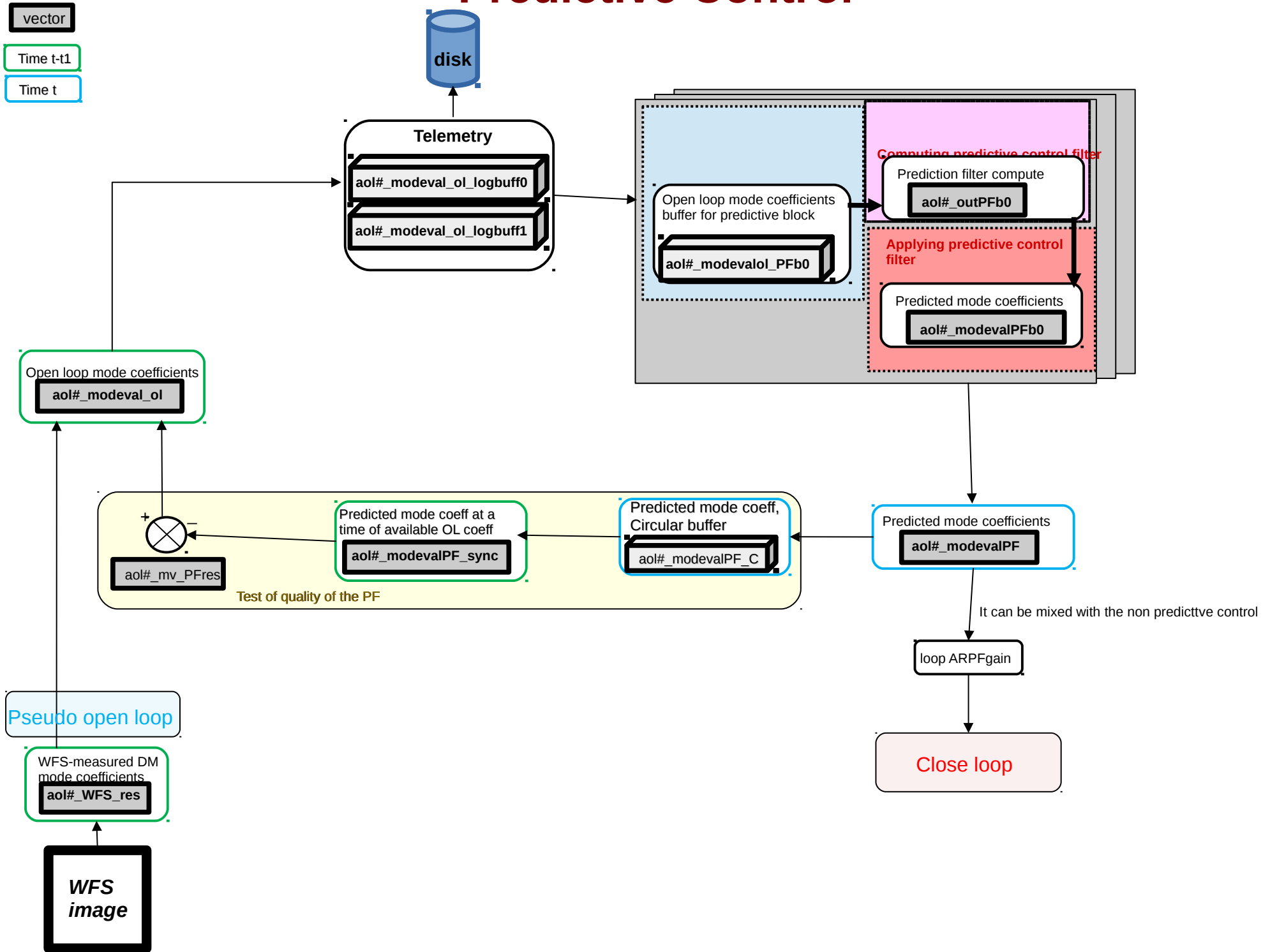
vector

Time t-t1

Time t



Predictive Control

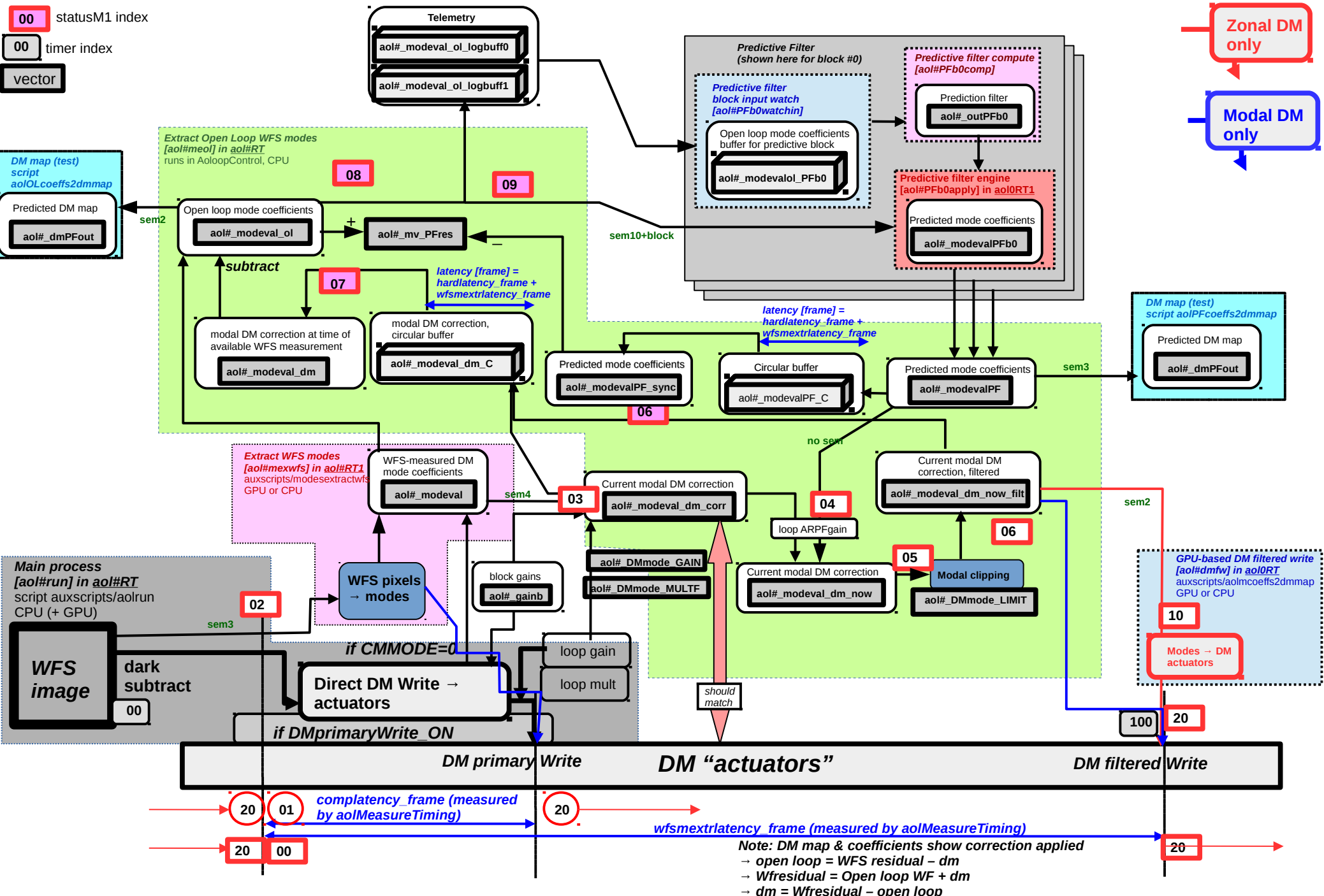


[process name] (same name as tmux session)
aol0RT : CPU set

Processes, output to DM (main loop)

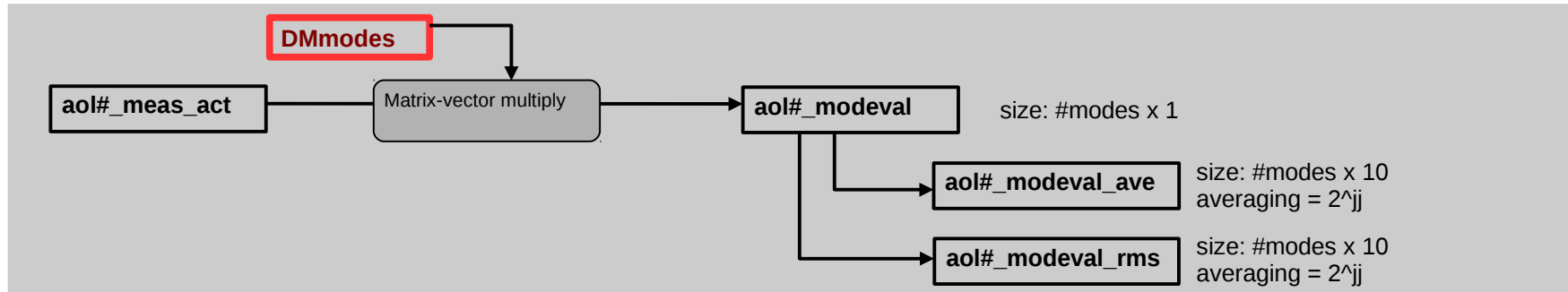
- 01 status index
- 00 statusM index
- 00 statusM1 index
- 00 timer index
- vector

$gain[m] = loopgain * gainb[block] * aol_DMmode_GAIN[m]$
 $mult[m] = loopmult * multb[block] * aol_DMmode_MULT[m]$
 $limit[m] = loopmaxlimit * limitb[block] * aol_DMmode_LIMIT[m]$

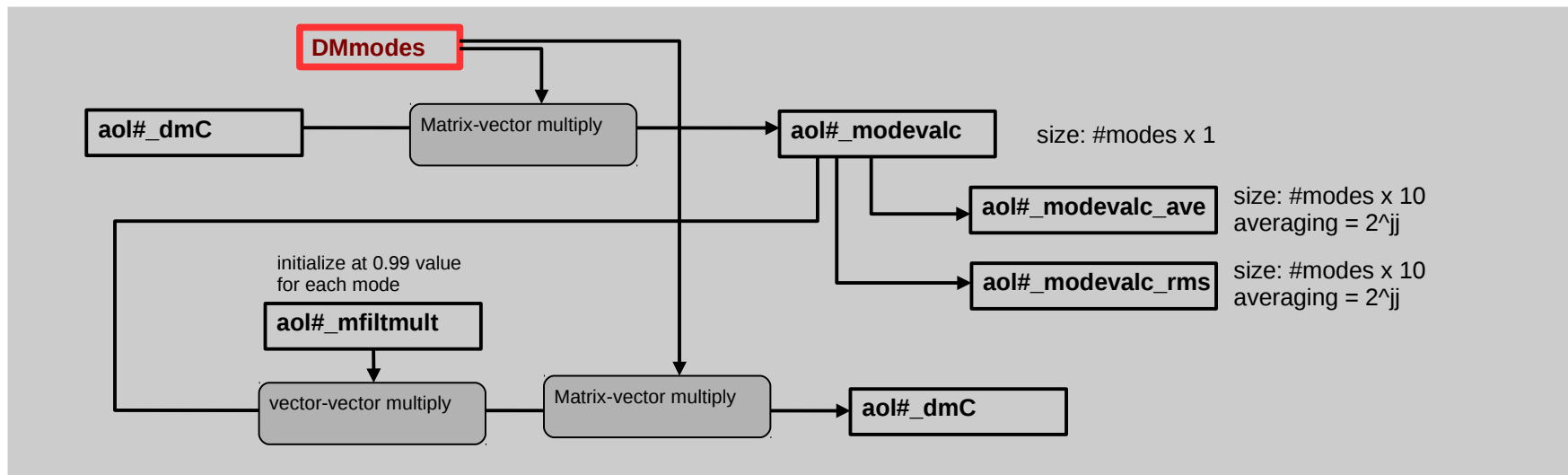


Auxillary processes

Decompose WFS measurements in modes

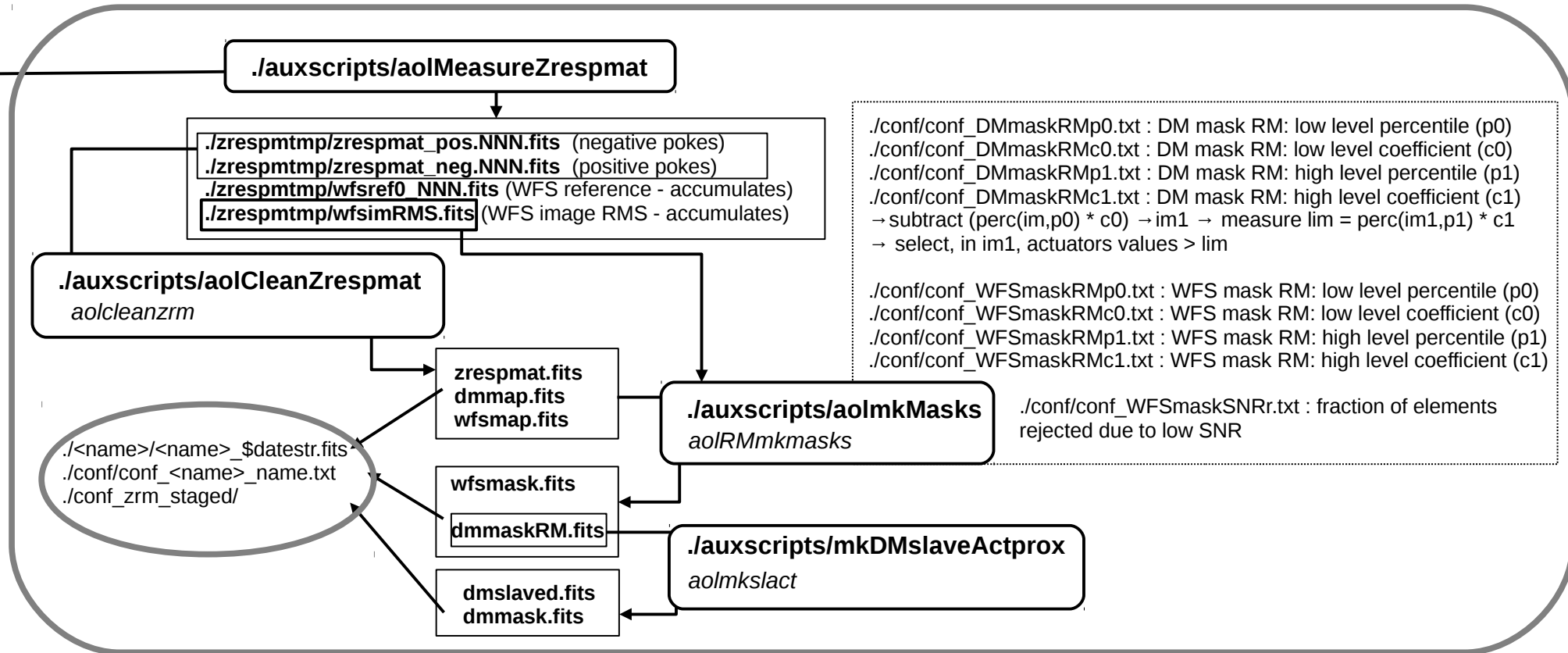


Decompose DM commands in modes + apply modal mult gains

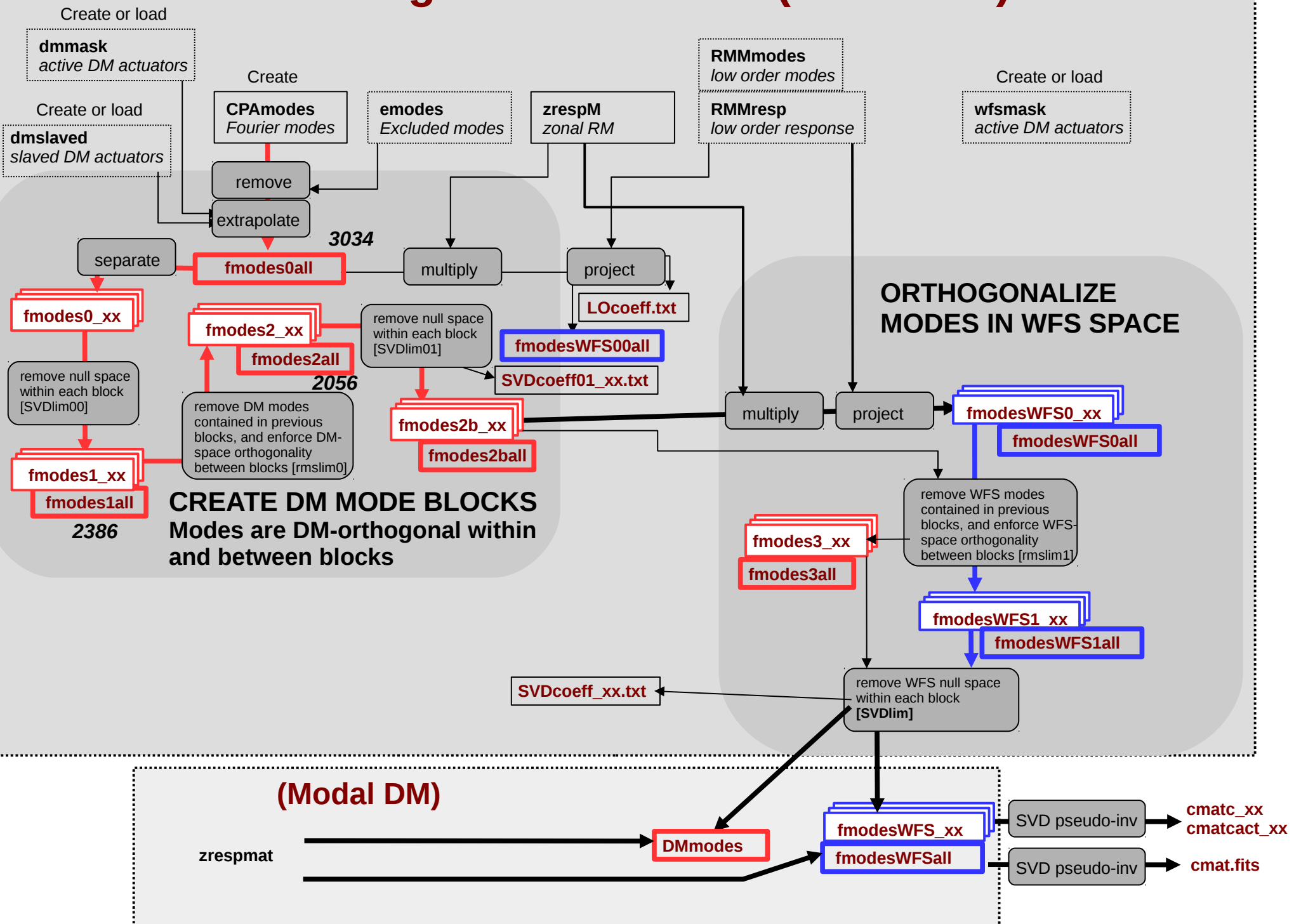


Zonal response matrix acquisition → masks

function_zresp_on → function_zresp_off



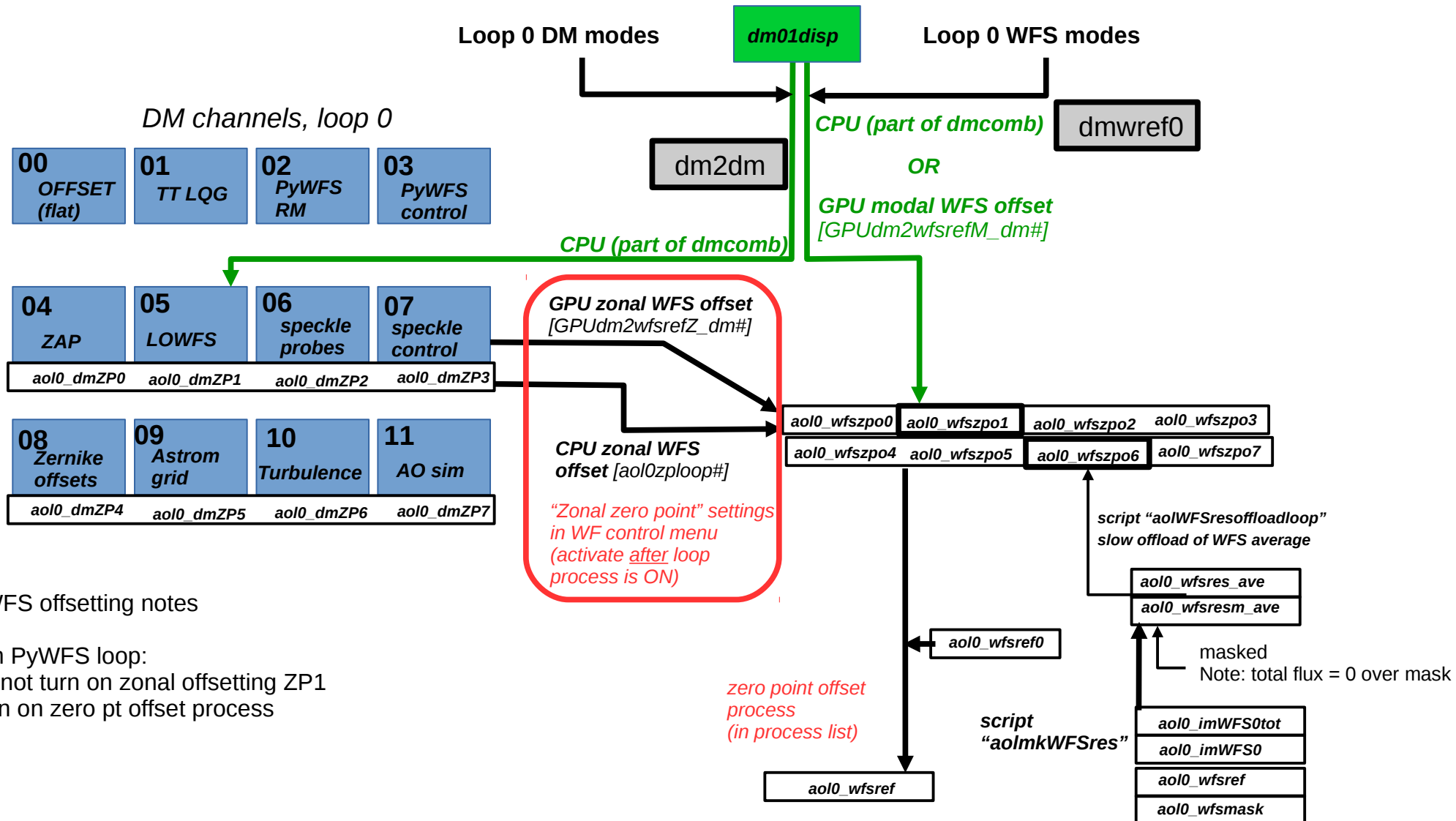
Making control modes (Zonal DM)



OFFSETTING

LOWFS (loop #1, dm01) → PyWFS (loop #0, dm00)

Green color: process is part of loop #1



Hardware Latency

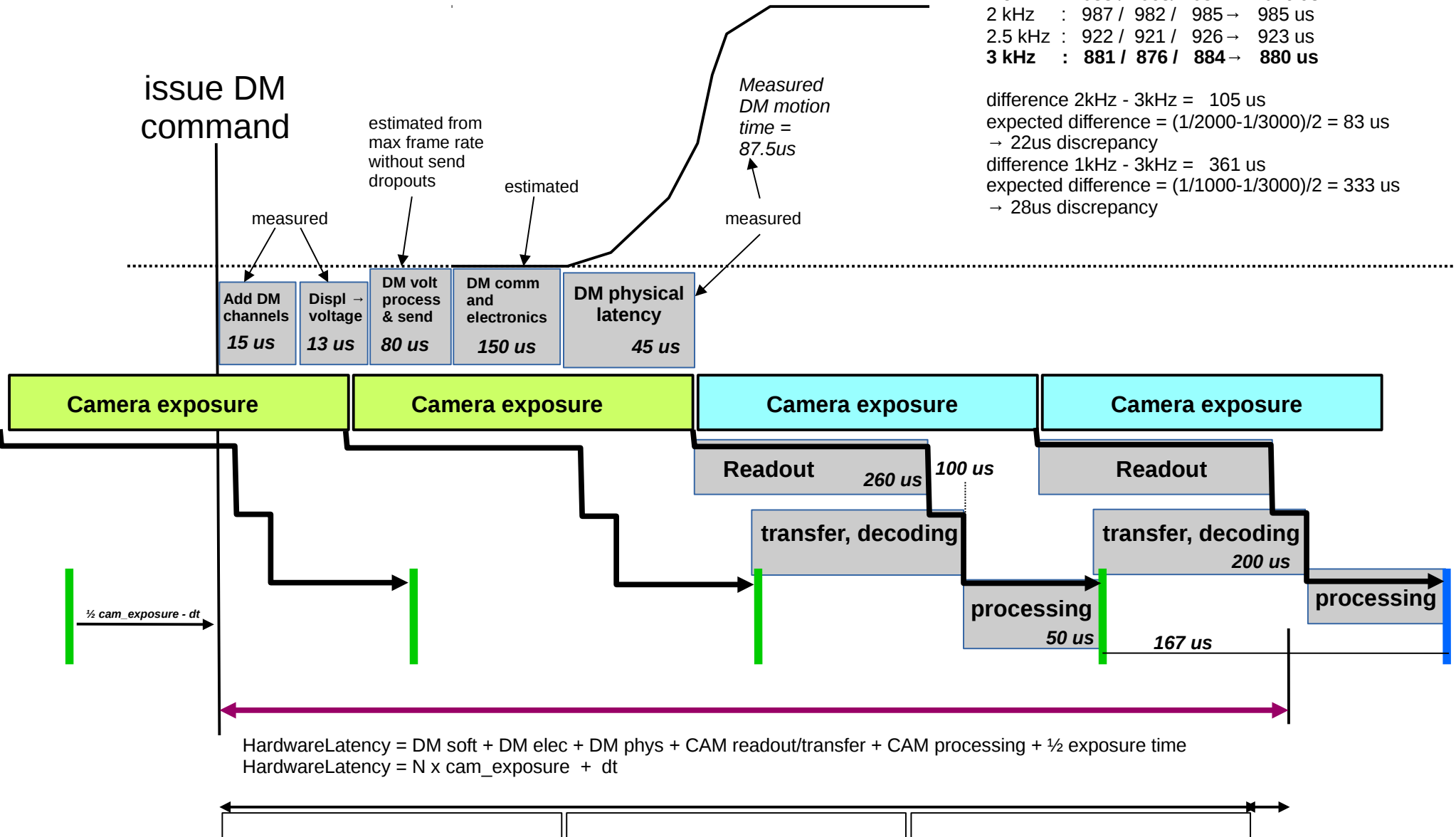
Definition:

*Time offset between **DM command issued**, and **mid-point between 2 consecutive WFS frames with largest difference***

SCEXAO measured hardware latencies:

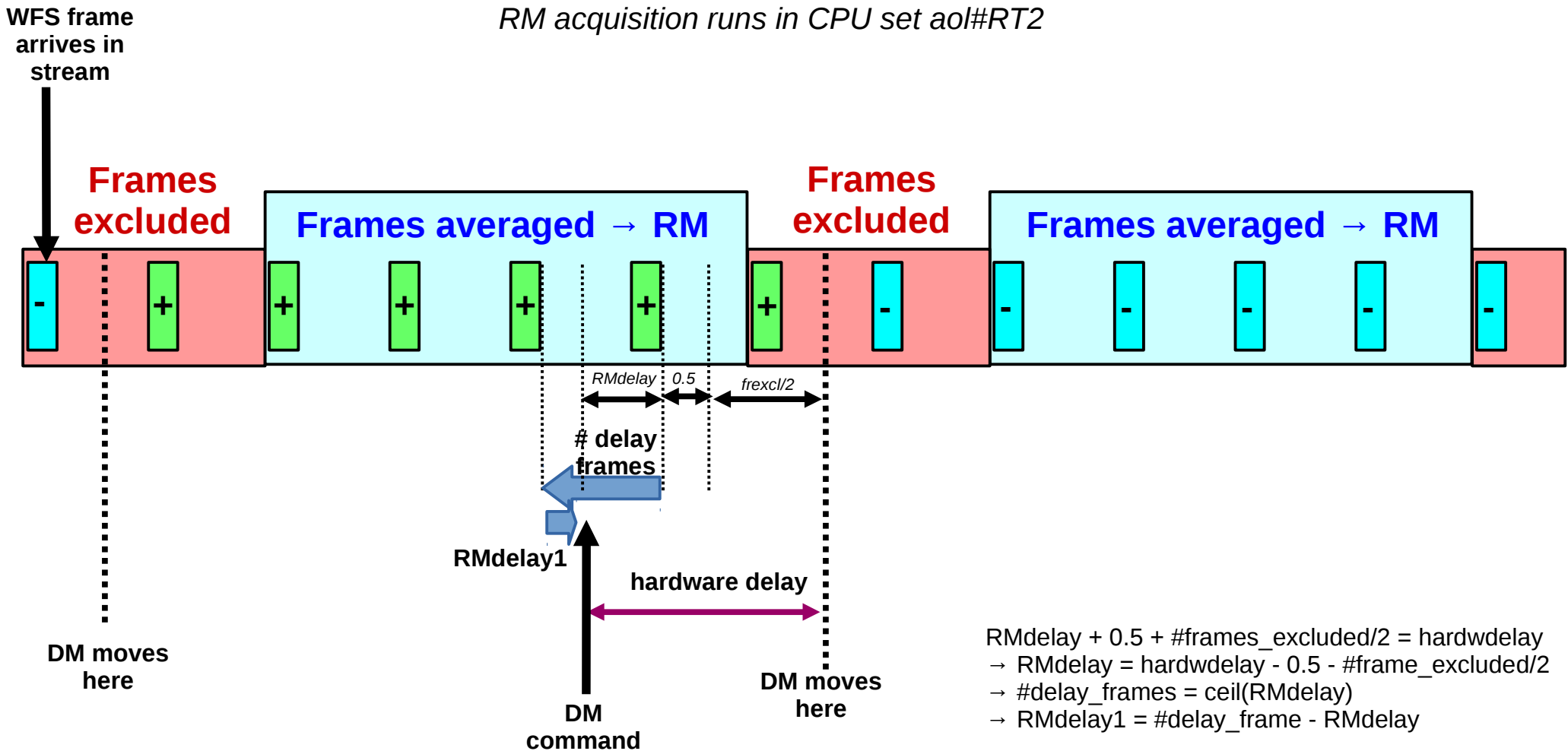
1 kHz : 1253 / 1260 / 1269 → 1261 us
 1.5 kHz : 1083 / 1065 / 1081 → 1076 us
 2 kHz : 987 / 982 / 985 → 985 us
 2.5 kHz : 922 / 921 / 926 → 923 us
3 kHz : 881 / 876 / 884 → 880 us

difference 2kHz - 3kHz = 105 us
 expected difference = $(1/2000 - 1/3000)/2 = 83$ us
 → 22us discrepancy
 difference 1kHz - 3kHz = 361 us
 expected difference = $(1/1000 - 1/3000)/2 = 333$ us
 → 28us discrepancy



RM acquisition - Timing

RM acquisition runs in CPU set aol#RT2



Loop:

Wait on and read WFS frame \rightarrow allocate WFS frame to appropriate frame block
 If poke required: wait RMdelay1, then poke

Predictive Control Implementation Notes

Predictive control implementation...

- ... **must be compatible with multi-WFS architecture**. See slide #6 for SCExAO multi-loop architecture, showing how master loop (Pyramid WFS) is offset by secondary loops (LOWFS and speckle control)

- ... **must be able to smoothly transition from/to conventional zonal DM write**. Slide #9 shows the details of the “direct fast DM write” mode, which is a GPU-based matrix multiplication going from WFS pixels to DM actuators (no modal step). Slide #9 shows the red arrow in the center-left part of slide #6

- ... **operates on modes that are constructed from the master WFS response**. Slide #6 shows how the modes are constructed. Slides #19-23 show actual reconstruction performance in real-time (in lab)

- ... **requires accurate and stable (at ~10us level) timings** to allow pseudo open-loop telemetry reconstruction. Slide #15 shows the timing diagram for SCExAO

- ... **requires reliable WFS & DM calibration**. This is achieved by frequently taking on-sky response matrix at full speed modulation (3 kHz) while the loop the closed. Slide #16 shows how timing info allows fast RM acquisition: SCExAO now takes a RM in 2 sec by modulating the DM at 3 kHz (frames excluded = 0, frames averaged = 1)

Slide #14 shows the **overall data flow** with predictive control, including real-time pseudo open-loop reconstruction.

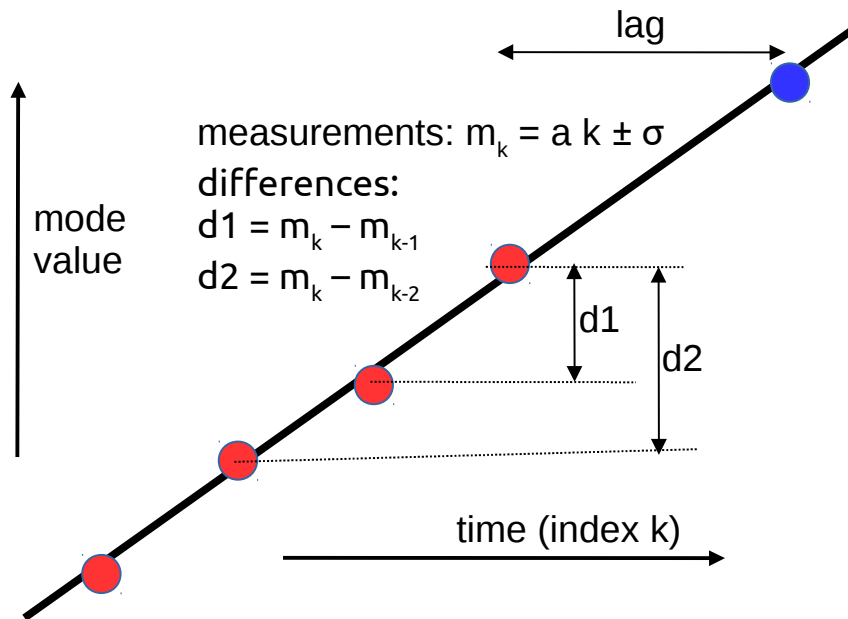
Automatic Gains Setting – Fast Mode

C code: [AOloopControl_AutoTuneGains](#) Script: [auxscripts/aolautotunegains](#)

Goal: **Find optimal gain for each mode in non-predictive mode in bright star regime.** This mode should be very reactive and robust, and able to recompute 1200 optimal gains in < 300 us to allow gain updates @ up to 3 kHz.

Bright star regime: input WF mode evolves linearly with time (control frequency > vibrations)

→ Error is quadratic sum of time lag and measurement noise, which can be expressed as simple functions of recent measurements.



With integrator (gain = g)

Time lag error:

$$\sigma_{TL} = a (\text{lag} + 1/g)$$

Measurement noise propagation:

$$\sigma_{MN} = \text{sqrt}(g/(1-g)) \sigma$$

Estimating slope (a) and measurement noise (σ)

$$\langle d1^2 \rangle = a^2 + 2 \sigma^2$$

$$\langle d2^2 \rangle = 4 a^2 + 2 \sigma^2$$

$$\langle d3^2 \rangle = 9 a^2 + 2 \sigma^2$$

$$\langle d4^2 \rangle = 16 a^2 + 2 \sigma^2$$

$$a^2 = (\langle d2^2 \rangle - \langle d1^2 \rangle) / 3 = (\langle d4^2 \rangle - \langle d2^2 \rangle) / 12$$

$$\sigma^2 = (4\langle d1^2 \rangle - \langle d2^2 \rangle) / 6 = (4\langle d2^2 \rangle - \langle d4^2 \rangle) / 6$$

$$\text{MstdDev}^2 = \langle m^2 \rangle - \sigma^2$$

Real time process steps:

- Compute open loop coefficient mode values while loop is closed
- Update slope and measurement noise from running averages of $d1^2$ and $d2^2$
- Optimize $\sigma_{TL}^2 + \sigma_{MN}^2$ as a function of gain → update optimal gain

Open loop reconstruction

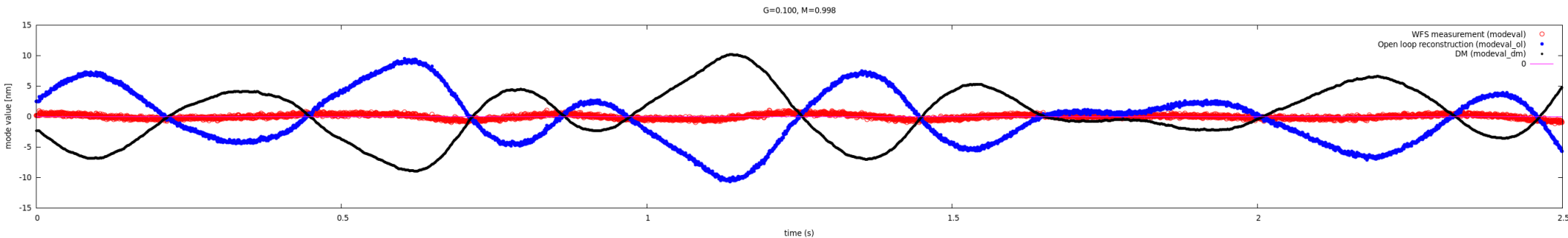
Turbulence injected:

8m/s wind speed, slightly filtered Kolmogorov spectrum (LOcoeff=0.1), 50nm RMS on DM (100nm WF), speed = 8 m/s
written to DM every 300us (3.333 kHz frequency)

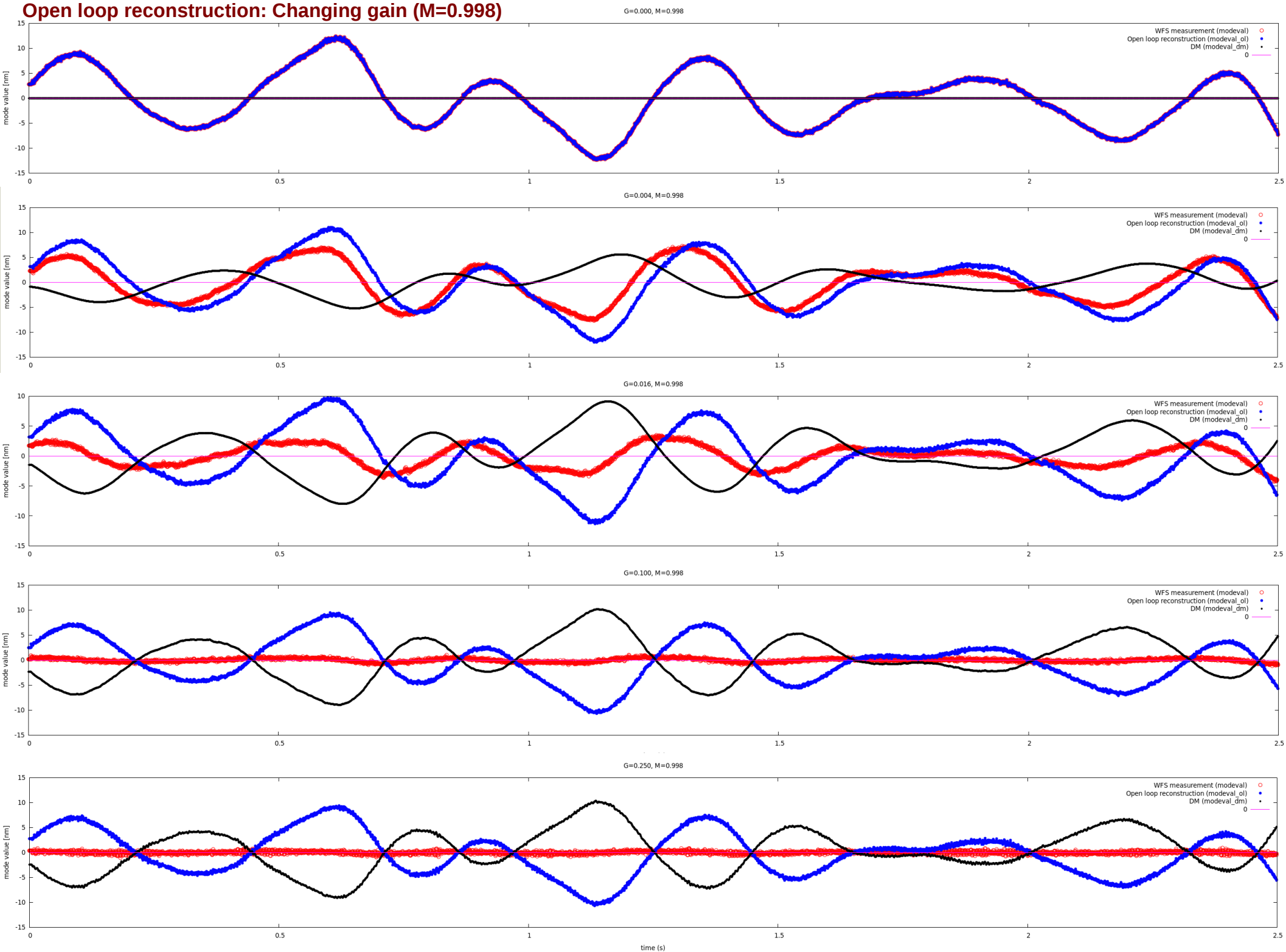
Showing **Open loop (blue)** and **WFS measured (red)** modal coefficient #30

Data acquisition start synchronized to turbulence sequence start

PyWFS running at 2kHz, 125mas modulation radius. Total delay = 2.6 frame



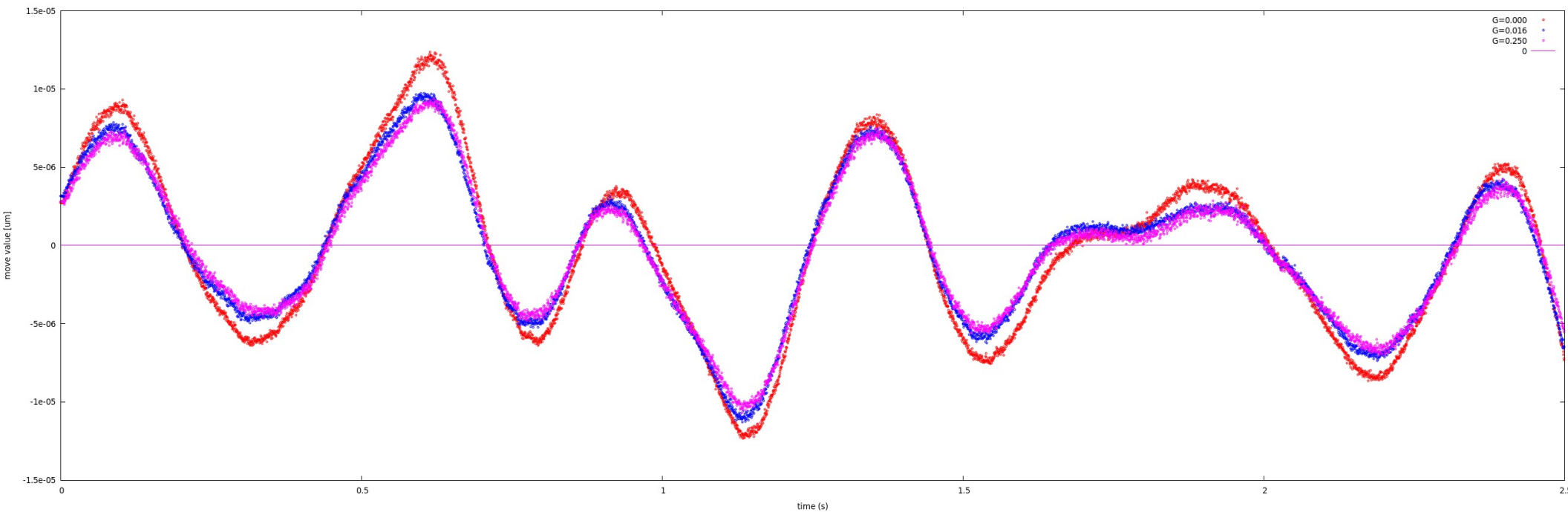
Open loop reconstruction: Changing gain (M=0.998)



Open loop reconstruction

Comparison between gain values

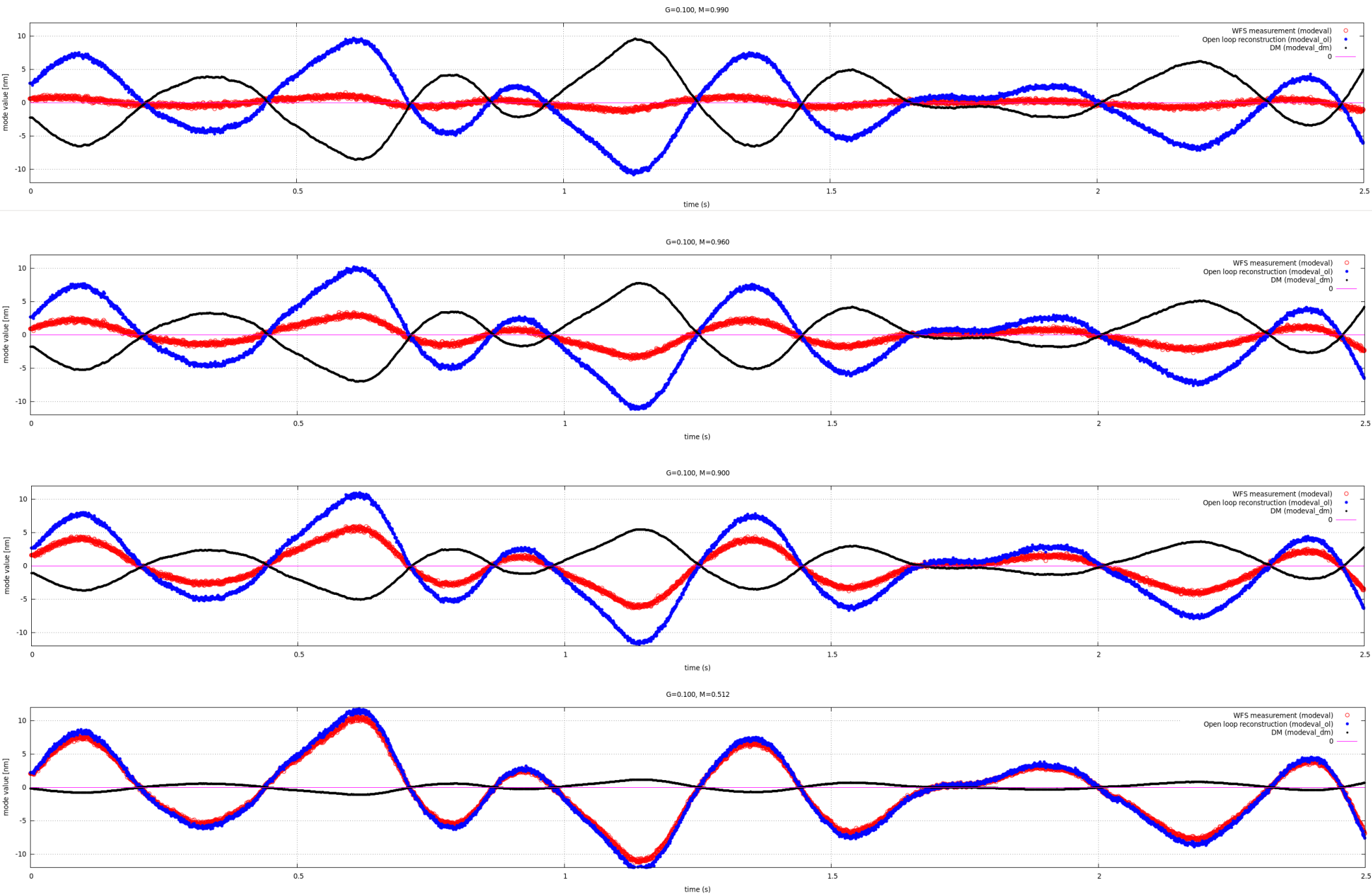
$G=0.000 \rightarrow$ over-estimates OL values
All $G>0.0$ reconstructions match at %-level



$G=0.000$ test relies entirely on WF residuals for OL estimation
 $G>0.000$ tests rely mostly on DM values for OL estimation

Test shown here uses full speed RM acquisition which underestimates RM by ~15% due to DM time-of-motion \rightarrow reconstructed WFs from WFS are over-estimated by ~15%

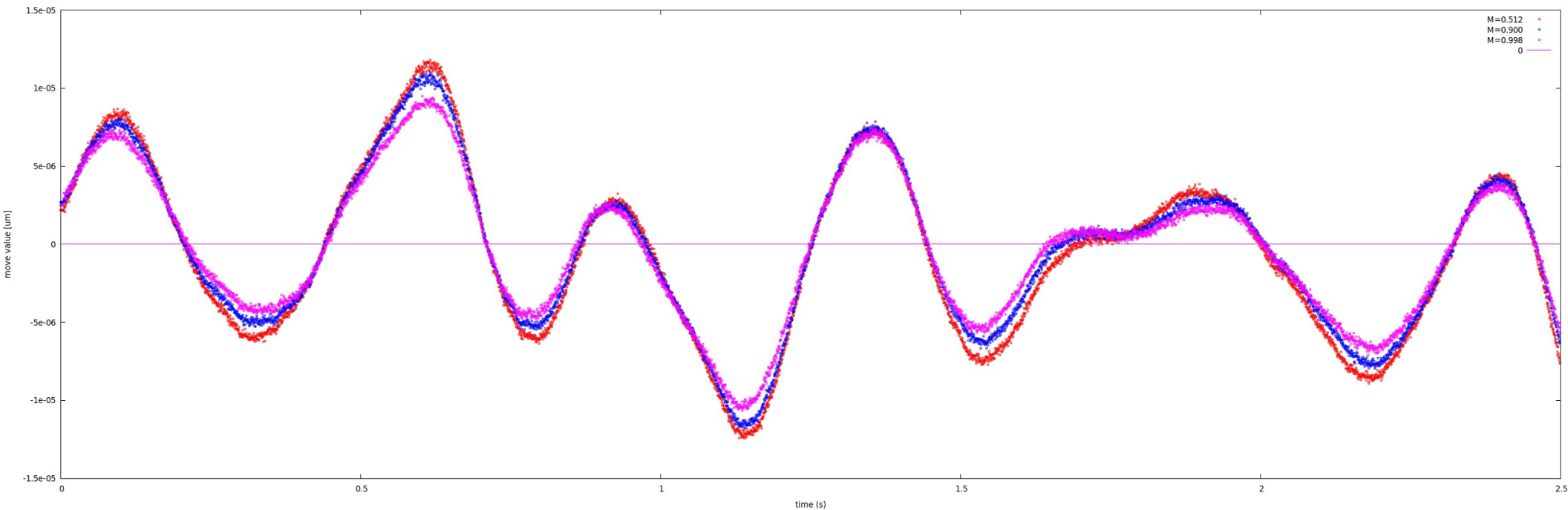
Open loop reconstruction: Changing mult factor (G=0.100)



Open loop reconstruction

Comparison between Mult factor

Small M over-estimates OL values
→ same effect as $G=0.0$ over-estimation



$M \ll 1$ tests rely mostly on WF residuals for OL estimation

$M \sim 1$ tests rely mostly on DM values for OL estimation

Test shown here uses full speed RM acquisition which underestimates RM by ~15% due to DM time-of-motion → reconstructed WFs from WFS are over-estimated by ~15%

Open loop reconstruction: noise propagation

$$\text{NOISE} = \sigma \times \text{sqrt}(1 + M^2 G^2 / (1-M^2))$$

