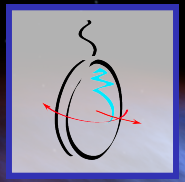
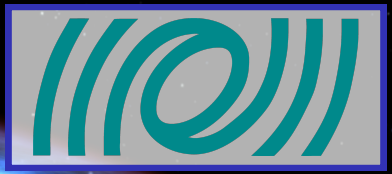
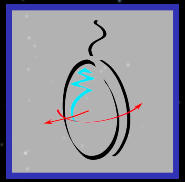






Lock Acquisition in Advanced Virgo (Part II): DC Readout, Noise Subtraction & Automation

—

Diego Bersanetti

INFN - Sezione di Genova

VIR-0405A-18

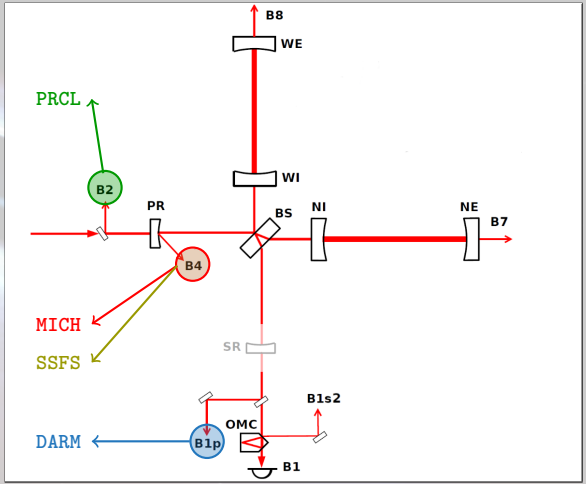
93rd JGW Seminar

Jun 15th, 2018

1. DC Readout Scheme
2. Noise Subtraction
3. Automation

Our Starting Point: ITF Locked at Dark Fringe

- ITF locked at dark fringe using *Variable Finesse* technique
- Main Automatic Alignment loop closed (DIFFp, PR, BS, COMMp)
- Longitudinal degrees of freedom locked:
  - DARM on B1p_56MHz
  - MICH on B4_56MHz_Q
  - SSFS on B4_56MHz_I
  - PRCL on B2_8MHz



Detection Schemes (1)

1st Generation: Heterodyne Detection

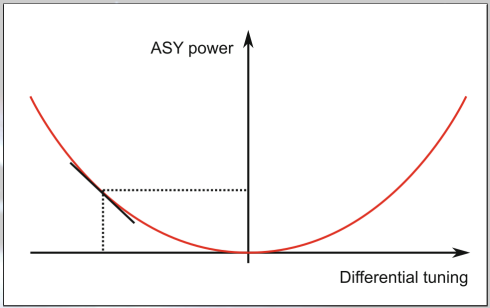
- The GW signals can be read out with the carrier field at the anti-symmetric port
- But we work in Dark Fringe, so we need a reference field at the anti-symmetric port to which the GW signal can beat against; then, we demodulate the beating signal and extract the GW information
- Solution: we add a *macroscopic* length difference in the Michelson: only the sidebands are affected (Schnupp asymmetry), but the interference (due to the carrier) is not
- We get a term linear in $h\left(t\right)$ and oscillating at the sideband frequency Ω:

$$P \simeq 8E_0^2 \mathcal{R} J_0\left(\Omega\right) J_1\left(\Omega\right) \frac{\mathcal{F}}{\pi} k_{\mathrm{L}} L h\left(t\right) \cos\left(\Omega t - \alpha\right)$$

Detection Schemes (2)

2nd Generation: Homodyne Detection

- We use instead the static DC power at the anti-symmetric port (DC Readout)
- We need a microscopic offset on MICH (or DARM) to have some light at the asymmetric port
- The offset is very small in order to keep the two cavities inside their resonance width

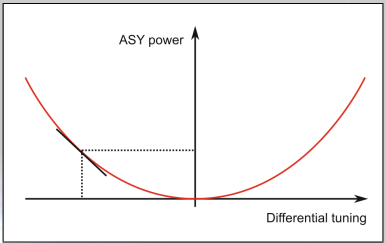


- The power sensed at the output will contain a static term linear in $h(t)$:

$$P \simeq 16\pi\mathcal{R}^2 E_0^2 J_0^2 \frac{L\Delta l}{\lambda_L^2}\mathcal{F}h(t)$$

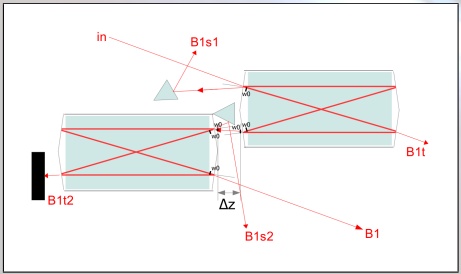
Detection Schemes (3)

- The static carrier field acts as the phase reference against which the carrier GW signal beats to provide a readable output
- *Complication*: any spurious field reaching the photodiode will spoil the sensitivity, as it will increase the total power but *not* the *optical gain* and the *signal*
- The sidebands are an example of this spurious field!
- It's not possible to switch them off since they are needed to control all the other DOFs of the ITF
- We filter them out before the photodiode adding at the output port two additional short and high-Finesse cavities called the Output Mode Cleaners

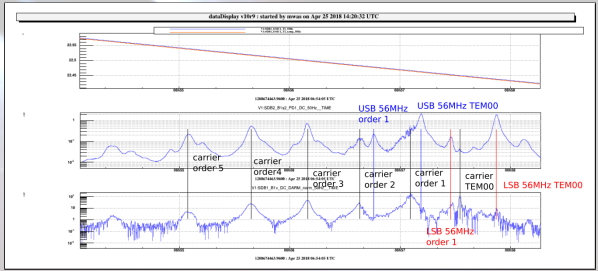


Output Mode Cleaners

- Two compact monolithic cavities, bow-tie shape
- Temperature stabilization and control with Peltier actuators



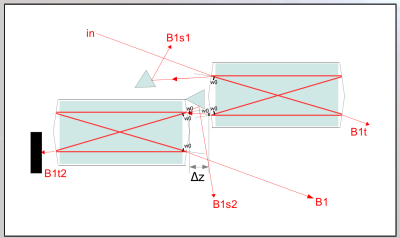
- Identification of resonances via temperature scan
- PZT actuators to lock the cavities
- Several B1∗ photodiodes available for DC Readout and monitoring purposes

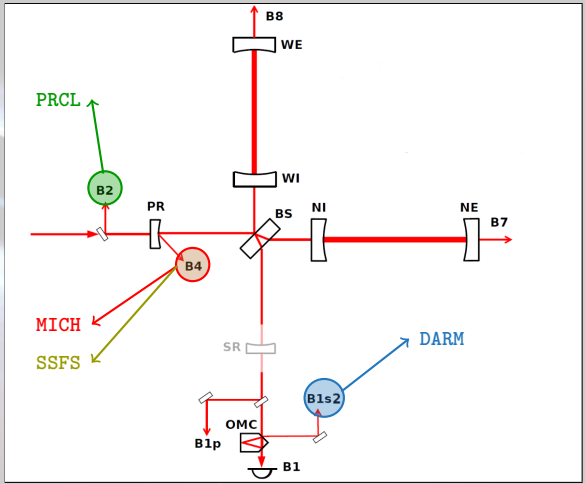


Credit: DET Team

Lock of the OMCs & DC Readout (1)

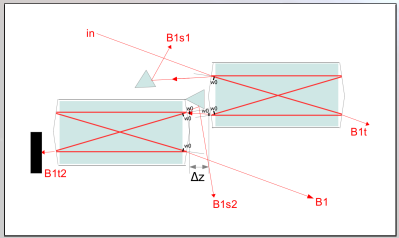
- Lock of OMC1:
  - we put the offset on DARM while still locked on RF
  - OMC1 is locked
  - we hand-off DARM to the (DC) signal reflected by OMC2 (B1_s2), which is kept out of resonance

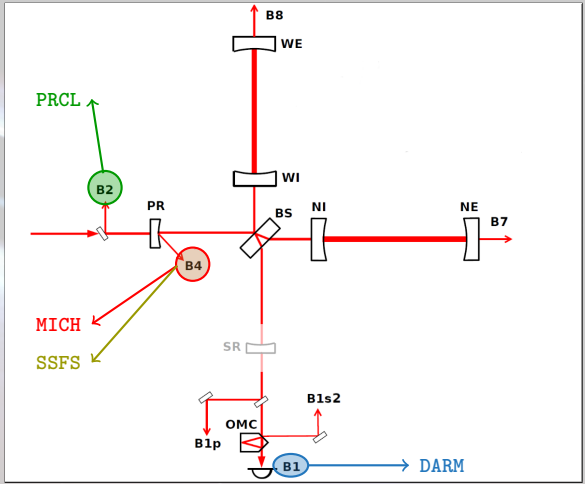




Lock of the OMCs & DC Readout (2)

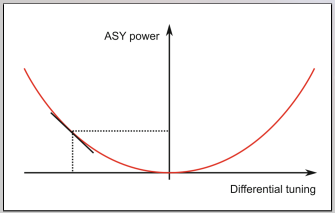
- Lock of OMC2:
  - while OMC2 scans for the resonance, the final detection photodiode `B1_DC` is already in the sensing matrix of DARM
  - OMC2 is locked
  - we remove `B1_s2` from the sensing, and DARM is locked on the designed sensor

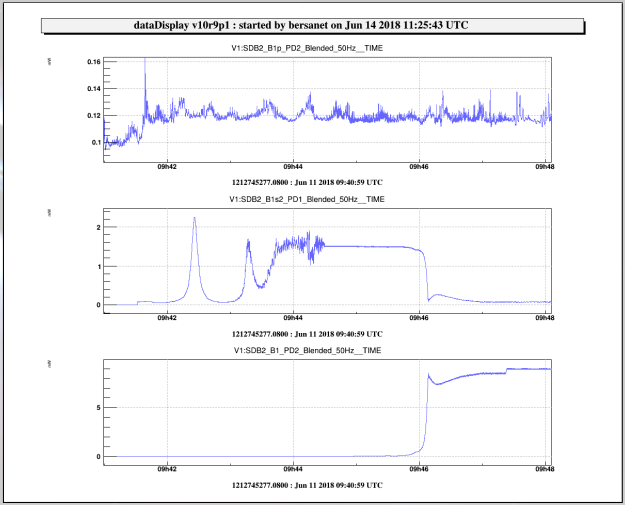




Lock of the OMCs & DC Readout (3)

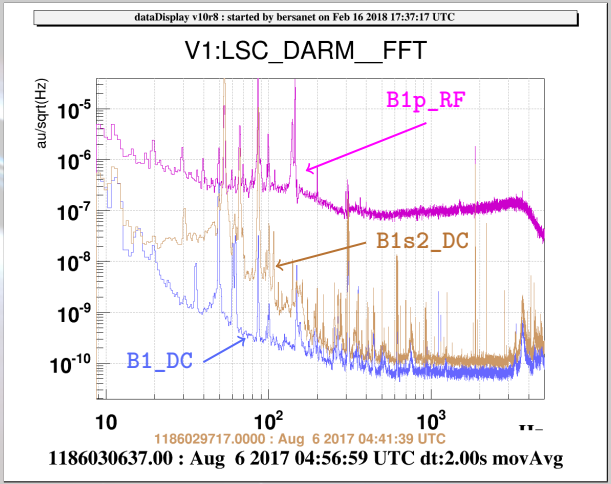
- The choice of the offset is optimized w.r.t. the trade-off between optical gain (slope) and noise (scattered light)
- The choice defines how much light reaches the anti-symmetric port
- Offset inter-calibrated w.r.t the different DARM sensors





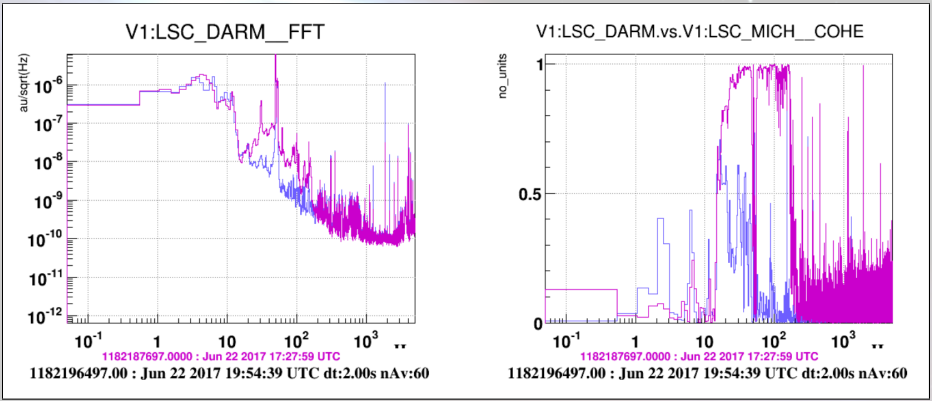
Lock of the OMCs & DC Readout (4)

- Big improvement in the DARM spectrum from RF to DC Readout
- Both B1_PD1 and B1_PD2 used to reduce noise
- Minor tweakings left:
  - increase of the UGF of DARM
  - additional Low Noise configuration for the mirror actuators
  - marionette reallocation from IMs to EMs
  - OMCs in Low Noise by reducing amplitude of dithering lines

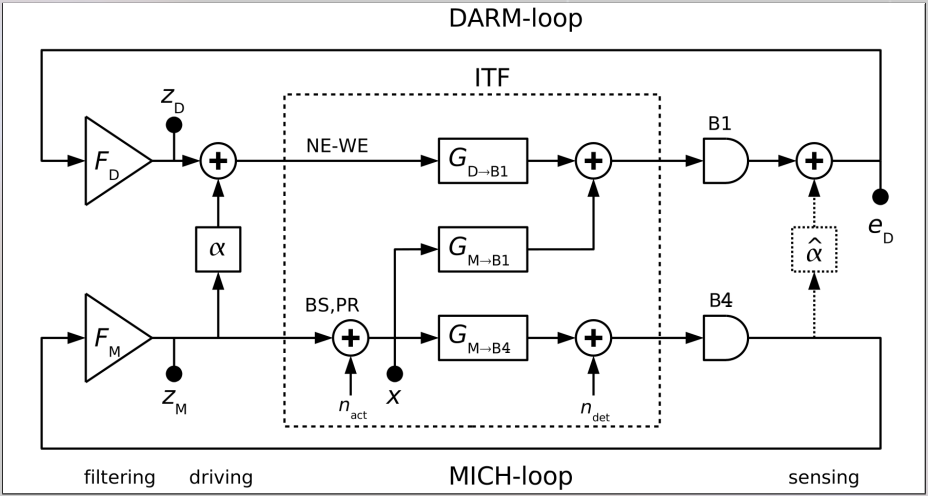


MICH → DARM Coupling

- Strong effect on DARM
- Frequency-dependent behaviour
- Most of the coupling is *linear*
- Online subtraction is possible



Alpha Technique: Mechanism



Alpha Technique: Definitions

- In principle the coupling factor is simply

$$\alpha = -\frac{G_{\mathrm{M}\to\mathrm{B1}}}{G_{\mathrm{D}\to\mathrm{B1}}}$$

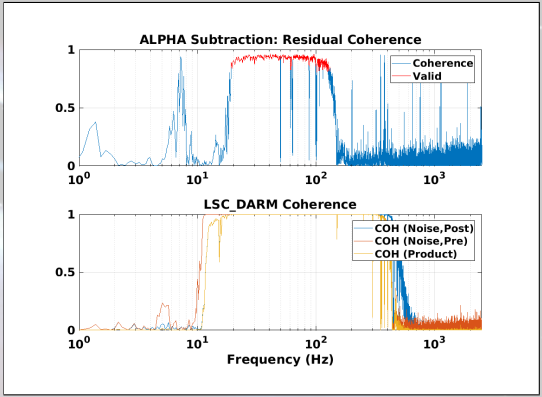
- But we cannot measure $G_{\mathrm{M}\to\mathrm{B1}}$ directly
- In the real ITF, we have instead:

$$\alpha_{\mathrm{new}} = \alpha_{\mathrm{old}} - \frac{\mathrm{TF}_{\mathrm{M}\to\mathrm{B1}}}{G_{\mathrm{Dcl}} \cdot \mathrm{TF}_{\mathrm{D}\to\mathrm{B1}}} = \alpha_{\mathrm{old}} - \frac{\mathrm{TF}_{\mathrm{M}\to\mathrm{B1}} \left(1 - \mathrm{TF}_{\mathrm{Dpost}\to\mathrm{Dpre}}\right)}{\mathrm{TF}_{\mathrm{D}\to\mathrm{B1}}}$$

Alpha Technique: Procedure

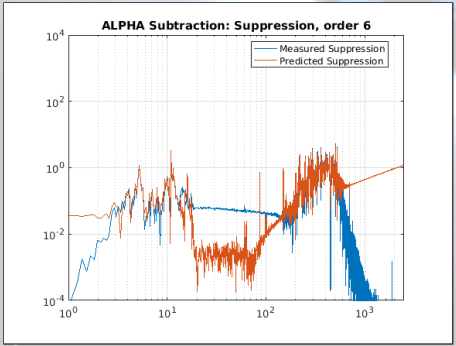
Procedure:

- Noise injections on both DARM and MICH
- Important: the DARM/MICH coherence should be high enough, but *without* saturating any of the actuators
- Calculate *offline* the new Alpha, by computing the TFs and fitting the new filter
- Important: Alpha is frequency dependent, so the frequency window and the frequency dependence of the weights are impacting
- Upload the new Alpha filter in the online software

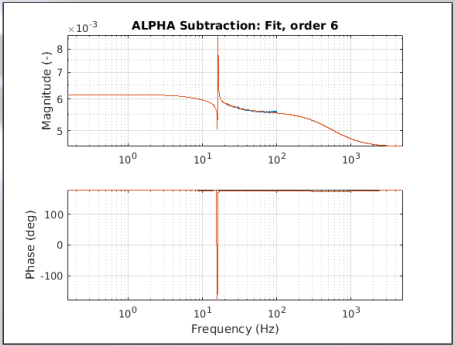


Alpha Technique: Evaluation

- Several fits are made for different orders
- The predicted new suppression is computed and compared to the current one

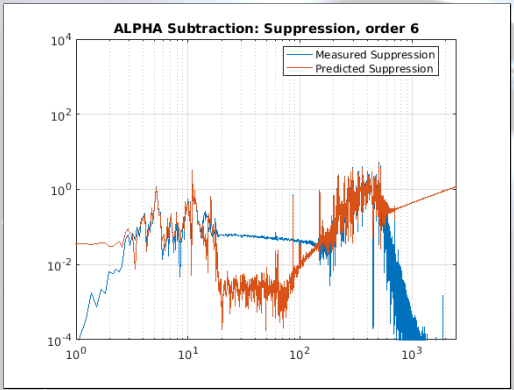


- Example of filter update after a change in the MICH loop made the subtraction under-performing

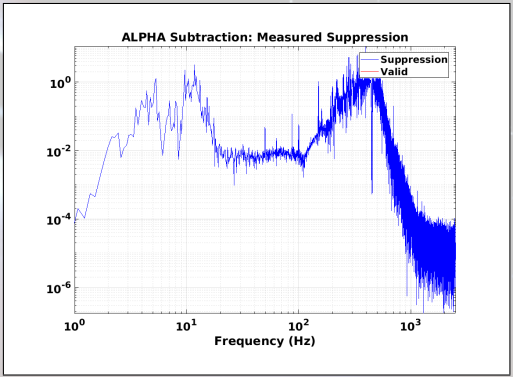


Alpha Technique: Validation

- With the new filter, another set of noise injections will validate the performance

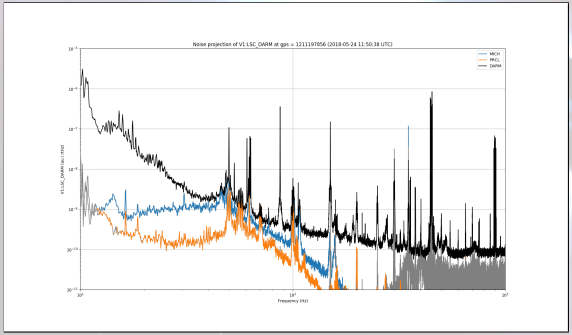


- Comparison between the predicted suppression and the measured one

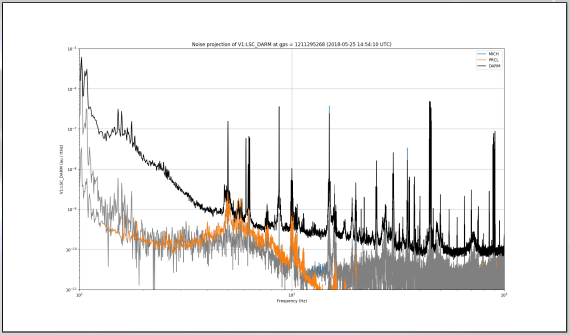


Alpha Technique: Longitudinal Noise Budget

- Contribution from MICH gets lower
- Coherence drops as well

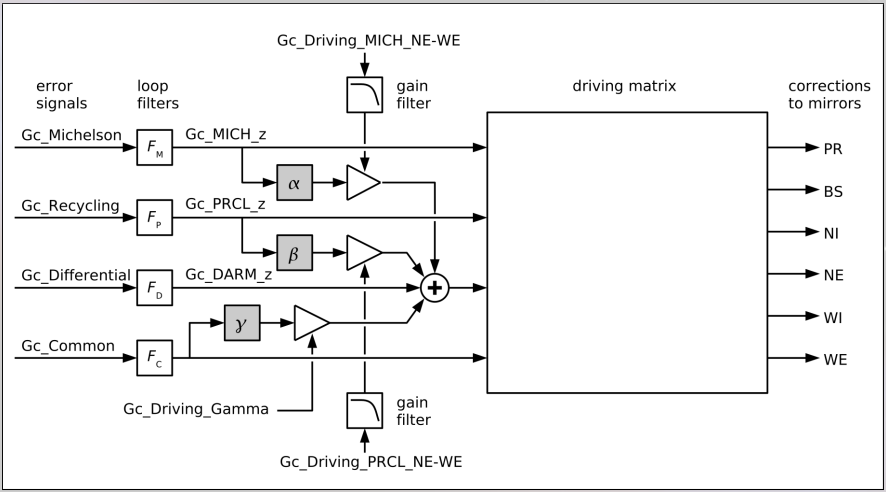


Old Alpha filter

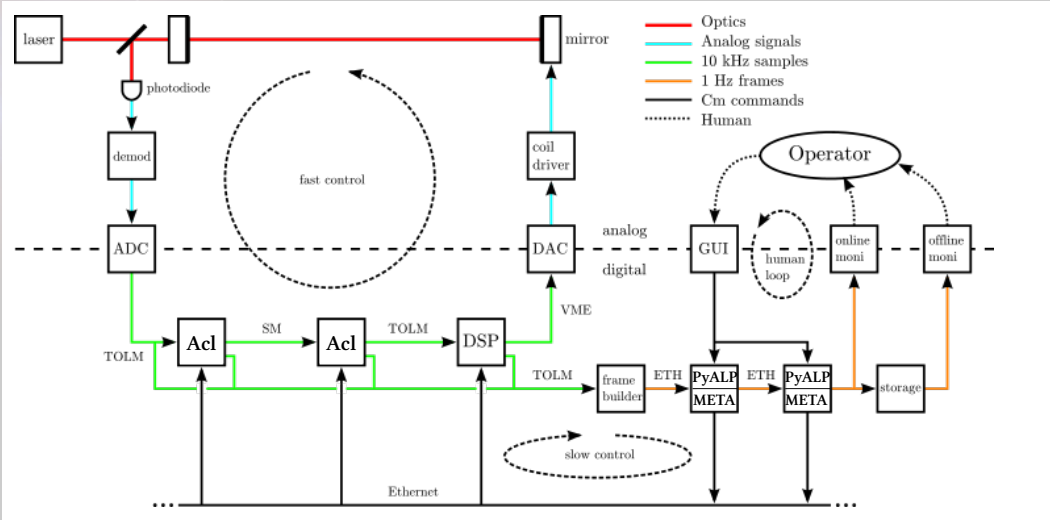


New Alpha filter

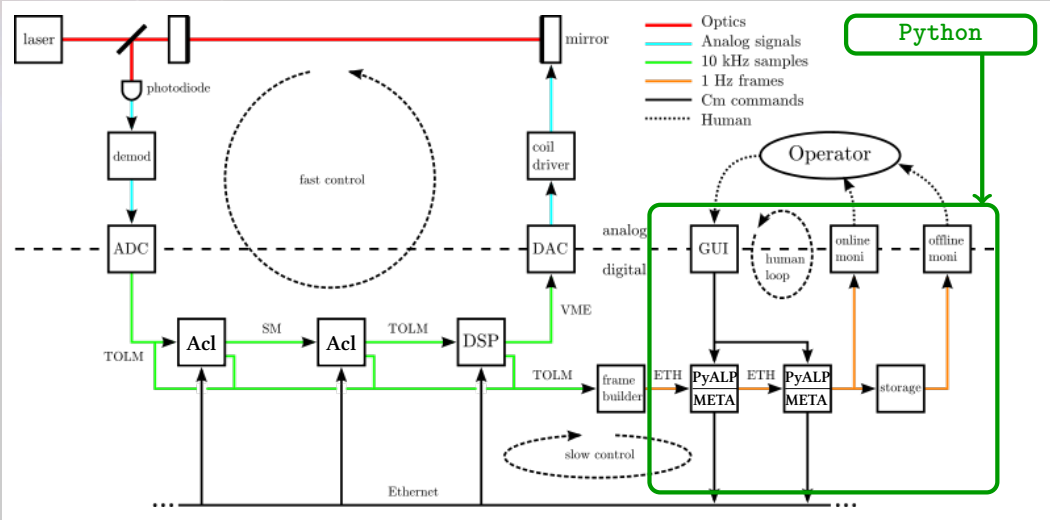
Alpha, Beta & Gamma



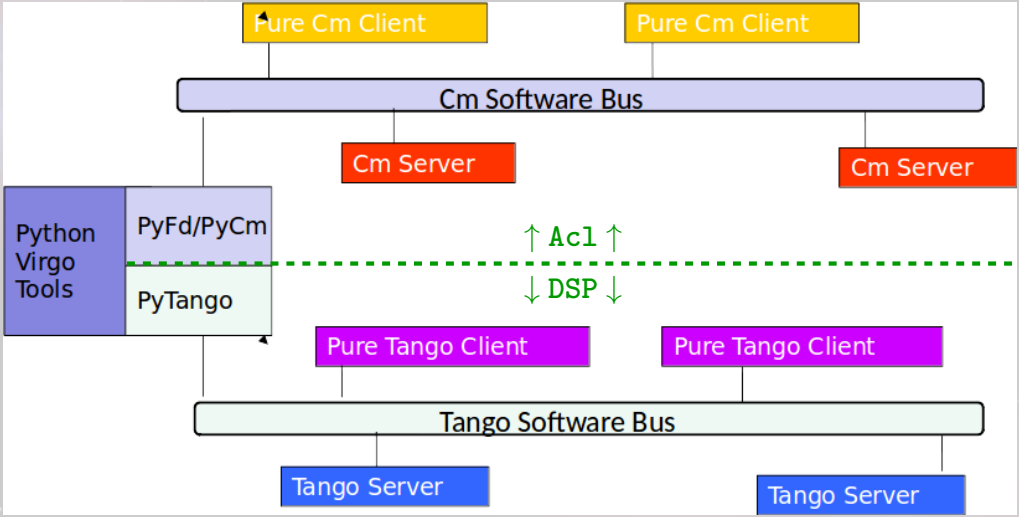
Control Structure (1): Fast and Slow Controls



Control Structure (1): Fast and Slow Controls

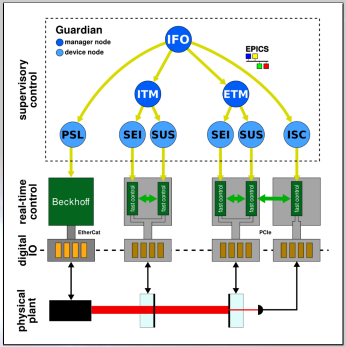


Control Structure (2): Middlewares (Cm & TANGO)



Automation in LIGO: `Guardian`

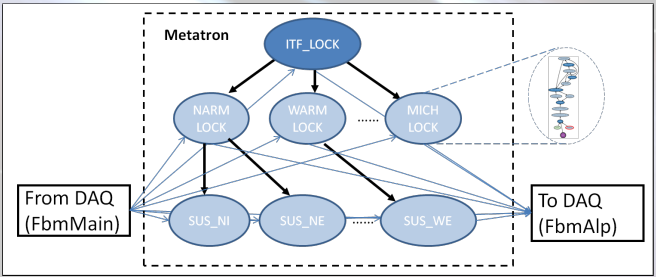
- `Guardian` is a **hierarchical**, **distributed**, **finite state machine**
- Individual automation **nodes** oversee well defined sub-domains of the full system, and they can be *managed* by other nodes
- Each node is a separate **daemon process**
- They are *state machine execution engines*: they load system modules that describe the state graph of the system and the code to be executed during each state
- Fully developed in `Python`: both the core program (`Guardian`) and the user apps (the nodes) are `Python` code



Paper & overview by J. Rollins:
https://arxiv.org/abs/1604.01456
https://dcc.ligo.org/LIGO-G1400016

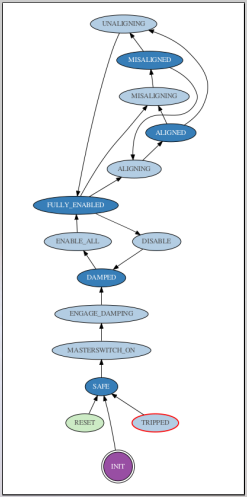
From Guardian to Metatron

- Guardian has been adapted and integrated in the Virgo environment as Metatron
- Built on top of the PythonVirgoTools library, it can communicate with all needed devices
- Uses the ConfigParser library and the .ini file format for managing configuration parameters
- Fully integrated in the DAQ chain



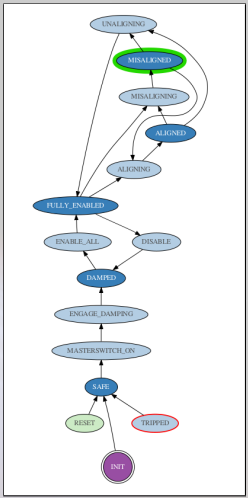
Metatron: State Graphs

- The **state graph** describes the accessible states of **single nodes**, and the allowable transitions between states (**edges**)



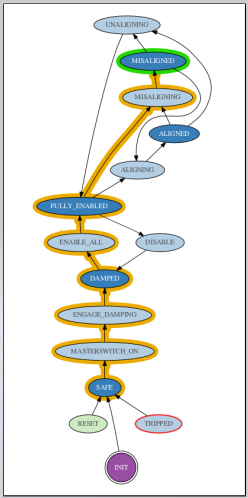
Metatron: State Graphs

- The **state graph** describes the accessible states of **single nodes**, and the allowable transitions between states (**edges**)
- Each node accepts **commands** in the form of a **state request**



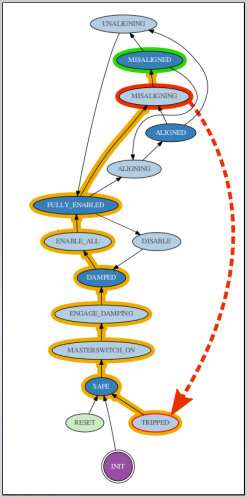
Metatron: State Graphs

- The **state graph** describes the accessible states of **single nodes**, and the allowable transitions between states (**edges**)
- Each node accepts **commands** in the form of a **state request**
- The **shortest path** from the current state to the requested state is computed, and **all the states** in-between are **executed**



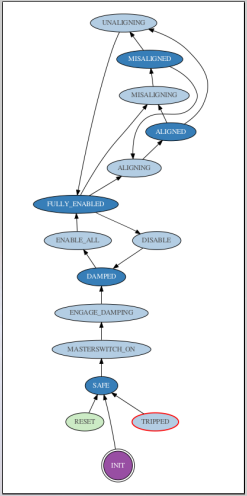
Metatron: State Graphs

- The **state graph** describes the accessible states of **single nodes**, and the allowable transitions between states (**edges**)
- Each node accepts **commands** in the form of a **state request**
- The **shortest path** from the current state to the requested state is computed, and **all the states** in-between are **executed**
- States may `return` a **jump target**, which is the name of another state to immediately "jump" to
- This **interrupts** the current path but, after the jump, `Metatron` recalculates the path back to the original request, and **continues**



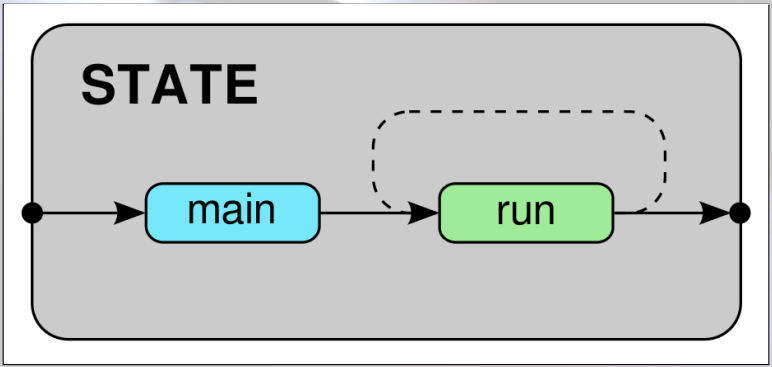
Metatron: State Graphs

- The **state graph** describes the accessible states of **single nodes**, and the allowable transitions between states (**edges**)
- Each node accepts **commands** in the form of a **state request**
- The **shortest path** from the current state to the requested state is computed, and **all the states** in-between are **executed**
- States may `return` a **jump target**, which is the name of another state to immediately "jump" to
- This **interrupts** the current path but, after the jump, `Metatron` recalculates the path back to the original request, and **continues**
- `Metatron` is a **finite state machine**: each state is a logically distinct block of code



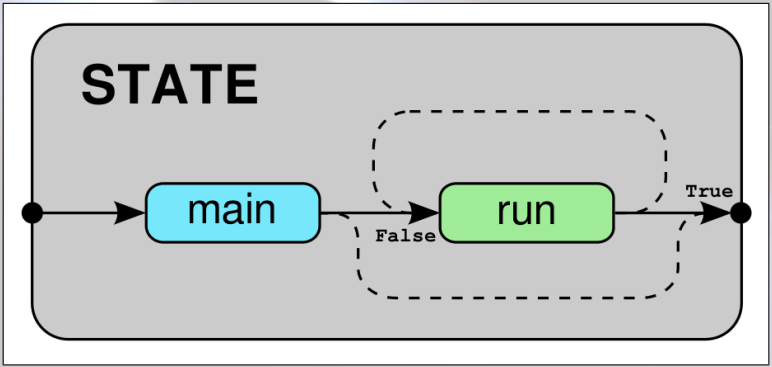
Metatron: States

- Two state methods:
  - `main()` : executed **once**, immediately when entering a state
  - `run()` : executed **in a loop**, useful to check for state/nodes completion conditions



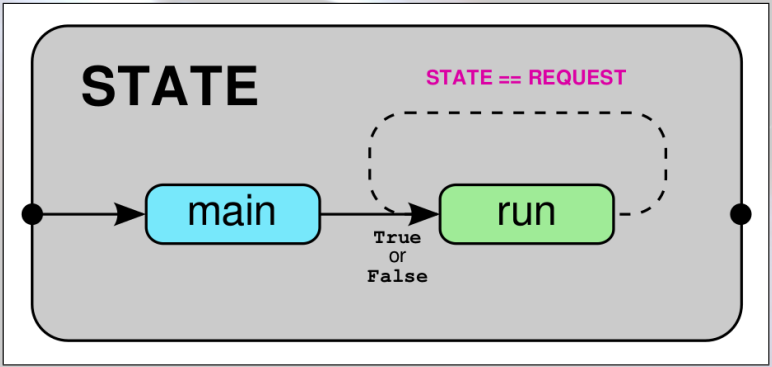
Metatron: States

- If/when either method returns `None` (default) or `False`, the `run()` method is executed
- If/when either method returns `True` the state completes and `Metatron` transitions to the next state (**edge transition**)



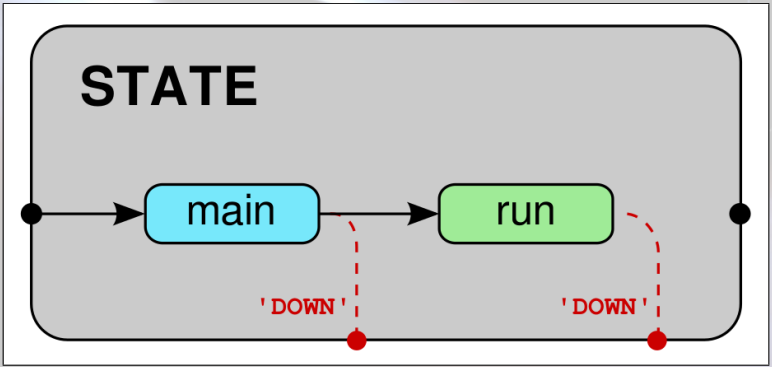
Metatron: States

- **Exception**: if this state was the **requested state**, the `run()` method will continue to execute even if it returns `True`
- This is useful for persistent checks (watchdogs and similar)



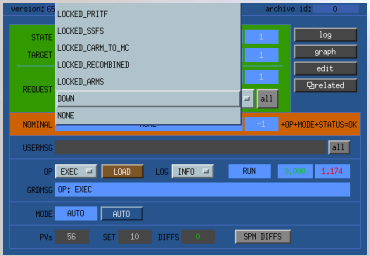
Metatron: States

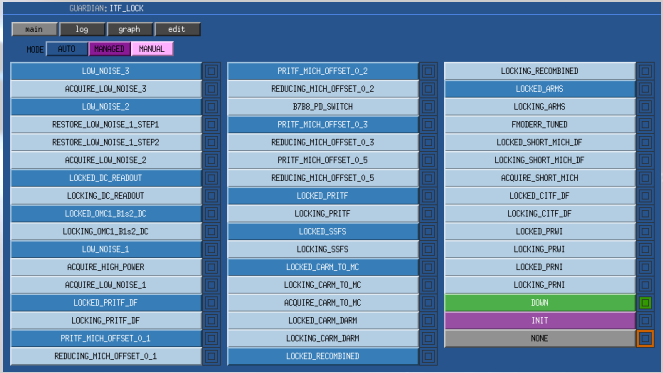
- Instead of `True` or `False`, either method can return the name of a state to initiate a **jump** transition (to the `DOWN` state in this case)



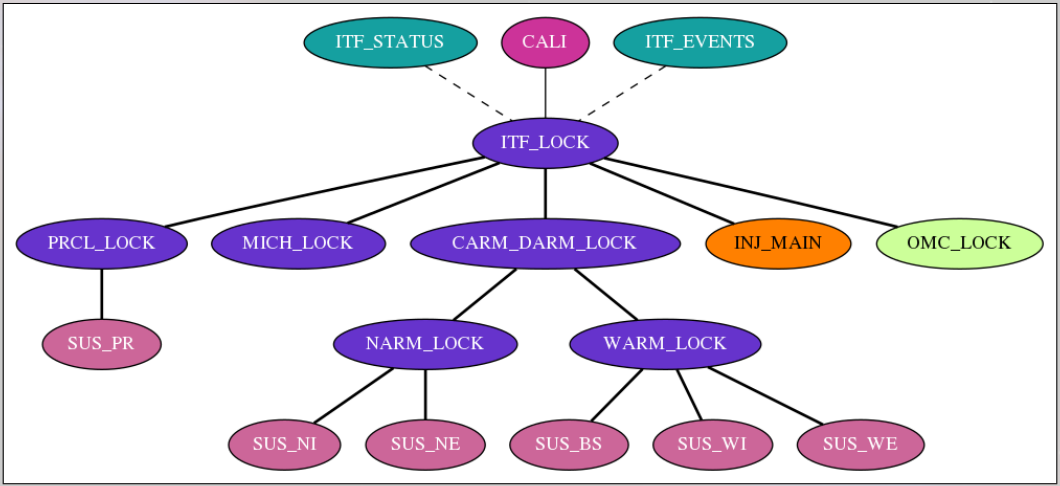
Metatron GUI: medm

- Main Lock Acquisition path in the drop-down menu
- the full list of states contains intermediate states and fancy configurations

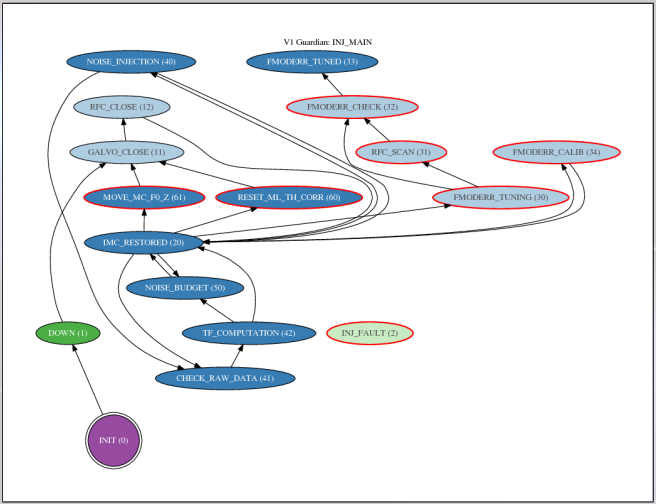




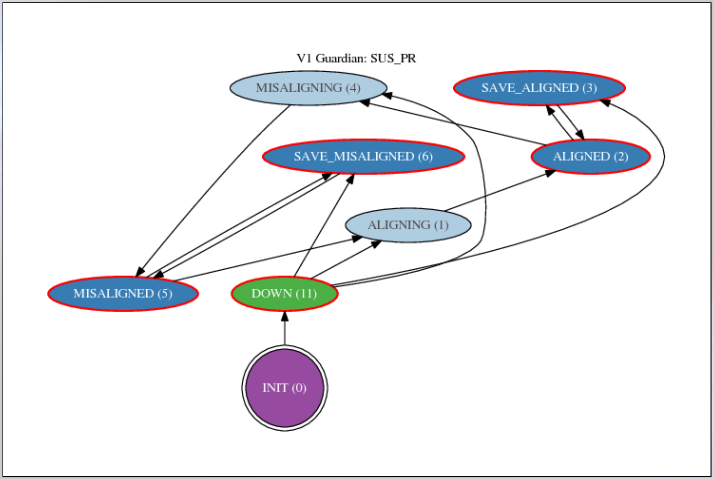
Metatron Node Hierarchy



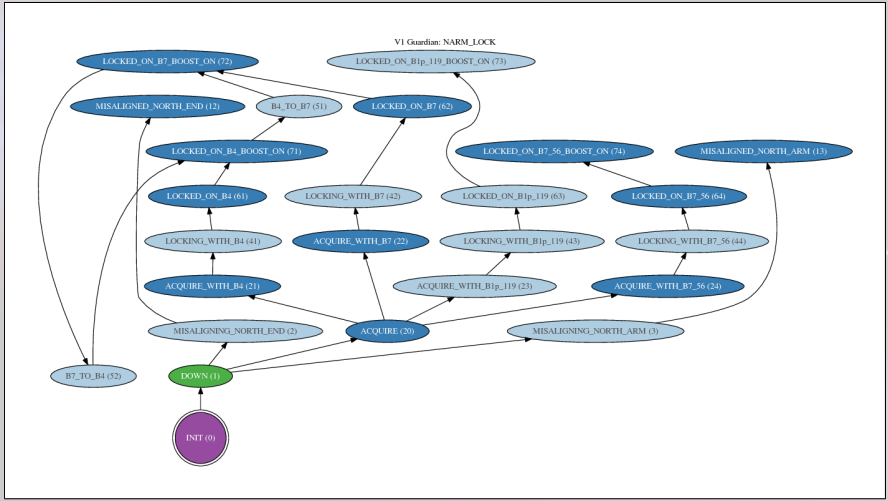
Metatron Node Example (1): Injection System Node



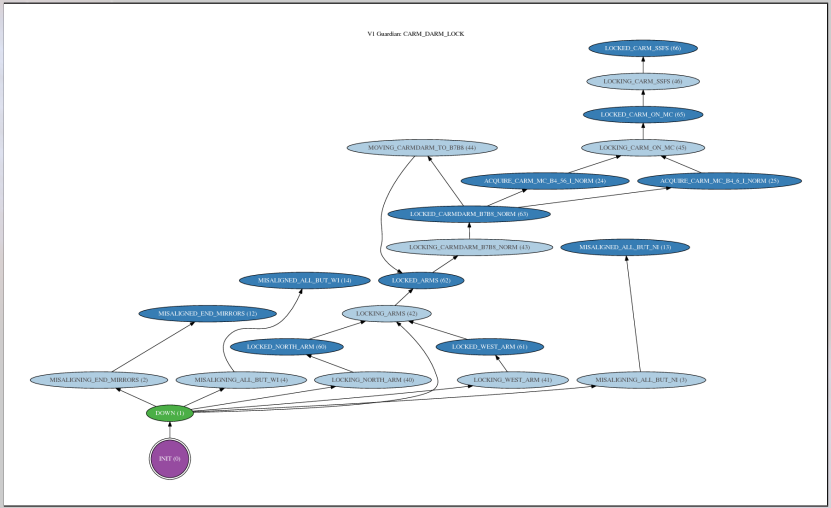
Metatron Node Example (2): Suspension Node



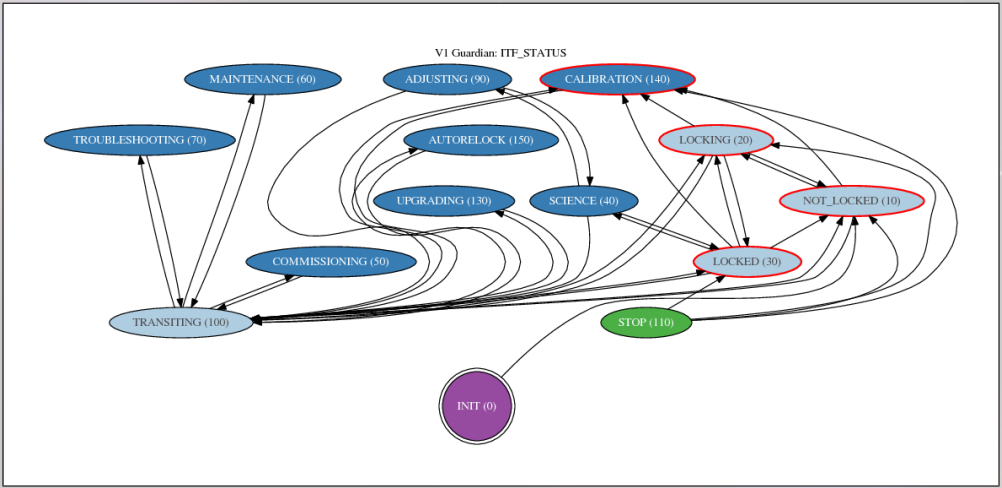
Metatron Node Example (3): Arm Locking Node



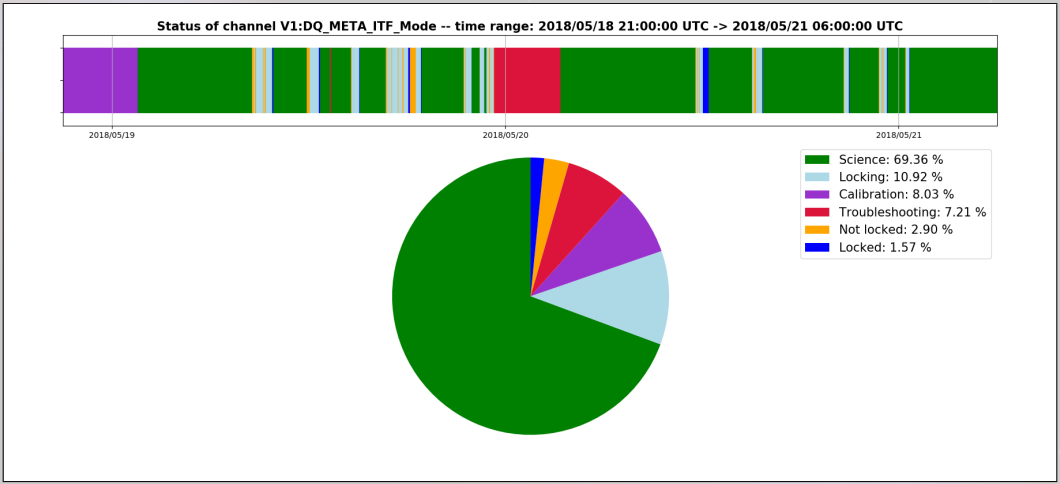
Metatron Node Example (4): CARM/DARM Locking Node



Metatron Node Example (5): Monitoring Node (1)



Metatron Node Example (5): Monitoring Node (2)




Thank You!