Guardian "core" status and roadmap

Jameson Graef Rollins

aLIGO integration meeting
January 16, 2015

LIGO-G1500065

Guardian

Current core installations

L1

Core r1063 (top node: r1134)
EZCA r274

Newer version for L1 top node to accommodate passive node monitoring improvements for ER6.

H1

Core r1083
EZCA r322

New release ready for deployment in ∼2 weeks, includes the following new features:

New features

request any state in graph

Requests no longer limited to states that have been labeled `request=True`; any state in the system can be requested. The requestable/non-requestable distinction will continue to have value in highlighting "end point" states and exposing them in the main control interface.

Working on new interface to allow easier selection of arbitrary states.

New features

edge weights

Edges can now be assigned arbitrary "weights". The default weight is 1. Execution paths are chosen based on the lowest total path weight (sum of all edge weights) to the requested state.

Current path calculation behavior is unchanged if no weights are specified.

fix handling of IFO-rooted channels

These should no longer produce channel access errors or memory leaks.

New features

execution times recorded

Execution time of main and run methods is recorded, both as elapsed time and total execution time of last method. Provides user feedback for states that are taking overly long to execute.

improved manager interface

Managers register their names with their subordinates. `MANAGED` is a separate flag (no longer a system `MODE`). Managers can now detect if a subordinate has been hijacked and can fault accordingly.

New features

dynamic, automatic code archiving

All system user code is automatically committed to a system code archive at initialization and after RELOAD. Provides a permanent record of all running code at any given point in time, regardless of USERAPPS commit status. Allows ability to view graphs of currently running system, versus just those of development code.

New features

notification message cycling

Up to ten `notify()` messages will be queued and auto rotated in the USERMSG message box. Messages are cleared if the notification didn't occur in the last cycle.

state decorators can return False for the state

State decorator pre-exec functions can now bypass further state execution if they return False, not just when they True or a jump target.

Shouldn't affect deployed decorators, but need to check.

New features

non-redirectable "protected" states

States can be labeled `redirect=False` to indicate that the state can not be redirected to a GOTO target until the state returns True. This is intended for "fault" states that should not be left until a fault condition clears. This replaces the current flaky protection heuristic based on jump history.

BEHAVIOR CHANGE: All states that should be protected (e.g. fault states) will need the new `redirect=False` flag to be set.

New features

requests no longer rejected based on lack of path

Requests are no longer rejected if there is no direct path to the requested state. All requests are accepted. If there is no path to the requested state, the target will be set to the current state and the system will not advance from the current state. Needed for monitor-only nodes.

BEHAVIOR CHANGE: Requesting an inaccessible state while the system is traversing a path will now cause the system to stall. This is ameliorated by making sure all graphs are well-connected (as I believe all currently deployed system graphs are). Could instead have nodes continue to last valid request.

New features

EZCA library now records all set points

The `ezca` interface now records all channel access writes to an internal table. Nodes can be told to automatically check all set points during execution and either alarm or fault upon detection of external changes.

This only works if there are no external channel access calls in e.g. external "`subprocess`" scripts. All `ezca` calls need to use internal Epics object. Need to take accounting of where external scripts are still being utilized.

Issues remain: Full coverage will require synchronization of setpoints with any BURT snapshots or front end settings definition files to be loaded. Almost certainly requires declaring all setting channels at load time, e.g. in a module attribute.

Settings monitoring

The union of front end and guardian settings monitoring would cover *almost all* channels in the system. Some channels would remain uncovered:

- **Beckhoff** May be possible to port front end settings monitoring sequencer code into TwinCAT IOC.
- **TCS HWS IOCs**
- others?

Could deploy monitor-only guardian nodes to watch whatever channels remain.

Still needed

- `ezca` writes are slow
- better NDS access management
- anything else?

Improvements to SUS alignment offset handling

Issues with SUS alignment offsets

- no way to know if current offsets correspond to ALIGNED or MISALIGNED state
- relies on remembering to save offsets to text files that are not well managed. If user forgets to save, alignment can be lost.
- no way to track ALIGNED vs. MISALIGNED offset values over time.

Proposal to address the issues

ECR E1400107, integration bugzilla 746

Store ALIGNED and MISALIGNED offset values in separate EPICS records, with switch to switch between them.

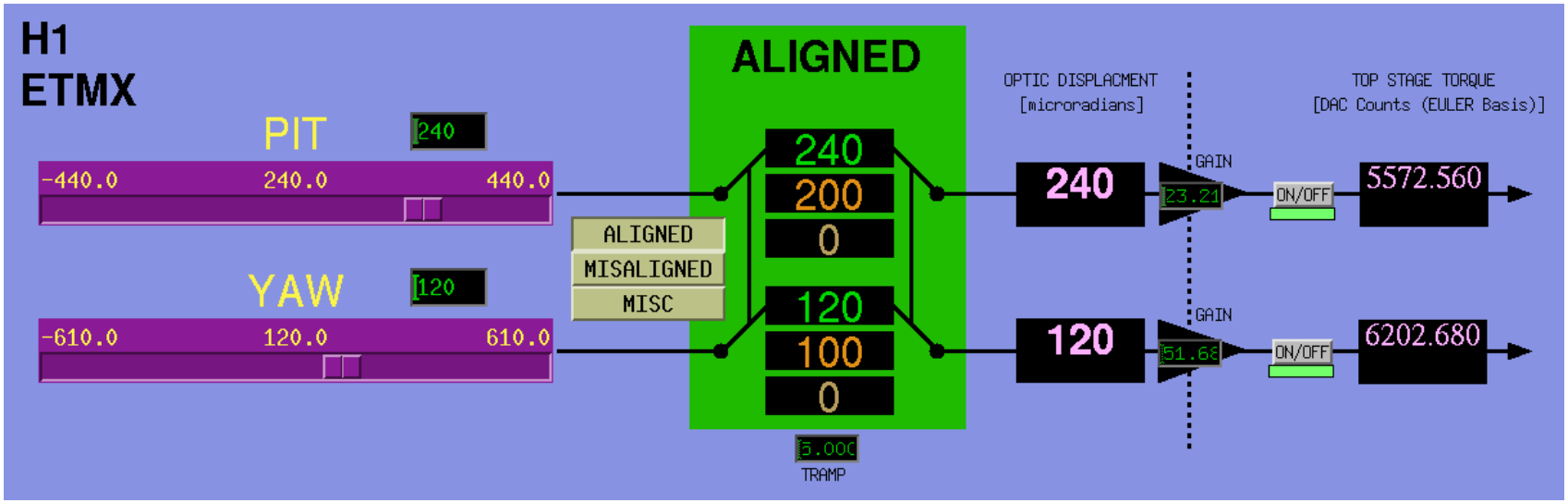
- Setting offsets would be done on a per-state basis.
- Offsets for each state would always be up-to-date, and stored separately in frames. **Would allow for previously unavailable trending of ALIGNED offset values**.
- SUS guardians could have ALIGNED and MISALIGNED states that would simply flip the alignment switch appropriately, with no worry that it would restore to out-of-date values.
- Simply select MISALIGNED to go to known good misalignment value, without having to touch offsets. Trivially restore to last ALIGNED value.

Proposed changes

Changes to be made:

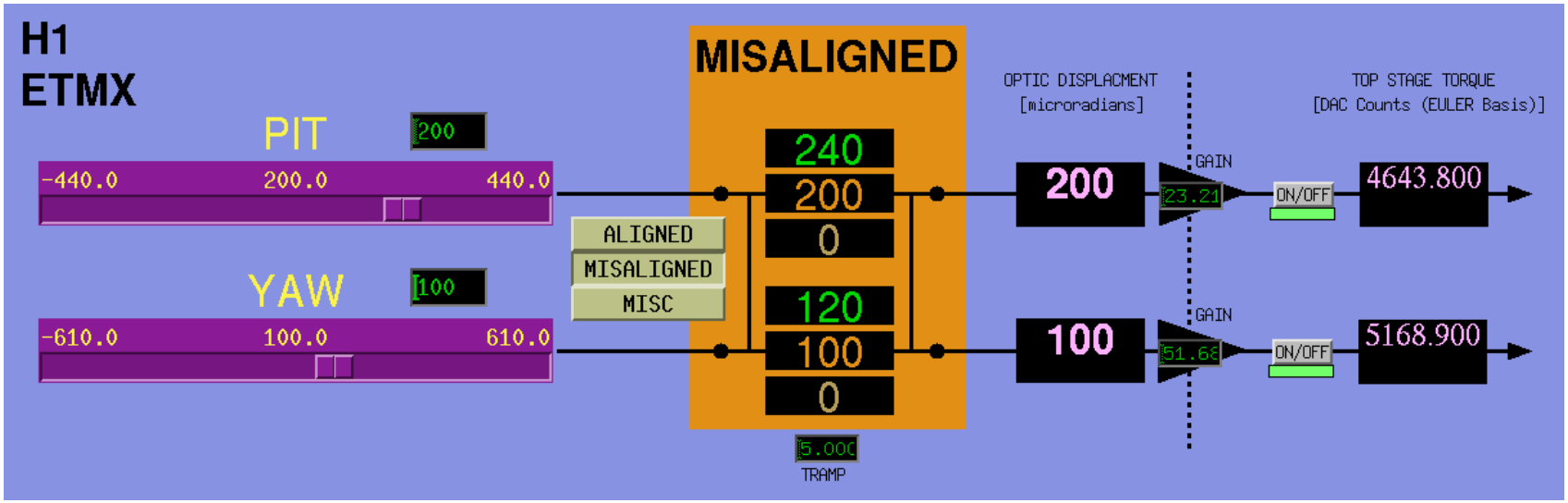
- Break out OPTICALIGN front-end logic into separate common OPTICALIGN library block for all suspensions (should be done anyway), with corresponding common OPTICALIGN MEDM screen.
- Add new PIT/YAW records for alignment states (ALIGNED/MISALIGNED/MISC), with a ganged ramping switch to handle switching PIT and YAW offsets simultaneously.
- Implement switch logic that writes appropriate offset into common OFFSET slider upon switching, and records current slider value into appropriate alignment record.
- Modify all SUS models to point to new OPTICALIGN library block.

MEDM interface



A single channel/slider can be used to control the current offset value, with a switch to select which alignment state the current value corresponds to.

MEDM interface



A single channel/slider can be used to control the current offset value, with a switch to select which alignment state the current value corresponds to.

Discussion

- Need way to recover last alignment after unexpected shutdowns (power outage, etc.). Make functions to recover values from frames or conlog.
- Do we need more than three alignment states (ALIGNED, MISALIGNED, MISC)?
- Do MISLIGNED values ever need to be set relative to ALIGNED values?