Simulation strategies employed in the development and maintenance of the Hobby-Eberly Telescope control system

Ramsey, Jason, Bryant, Randy, Drory, Niv, Elliott, Linda, Fowler, James, et al.

Jason Ramsey, Randy Bryant, Niv Drory, Linda Elliott, James Fowler, John Good, Gary J. Hill, Martin Landriau, Hanshin Lee, Ron Leck, Brian Vattiat, "Simulation strategies employed in the development and maintenance of the Hobby-Eberly Telescope control system," Proc. SPIE 10707, Software and Cyberinfrastructure for Astronomy V, 1070736 (6 July 2018); doi: 10.1117/12.2314471

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2018, Austin, Texas, United States

Simulation strategies employed in the development and maintenance of the Hobby-Eberly Telescope control system

Jason Ramsey[a], Randy Bryant[b], Niv Drory[a], Linda Elliott[a], James Fowler[b], John Good[a], Gary J. Hill[a], Lane Kolbly[a], Martin Landriau[c], Hanshin Lee[a], Ron Leck[a], and Brian Vattiat[a]

[a]McDonald Observatory, University of Texas at Austin, 2515 Speedway, C1402, Austin, TX, 78712-0259, USA
[b]Hobby-Eberly Telescope, University of Texas at Austin, USA
[c]Lawrence Berkeley National Laboratory, 1 Cyclotron Road Mailstop 50R5008, Berkeley, CA 94720, USA

ABSTRACT

The recently reinvented Hobby-Eberly Telescope (HET) has undergone a multi-year upgrade replacing the wide-field corrector, tracker, and metrology systems. The timelines for the production and delivery of the various upgrade components, combined with the desired turn around on deployment and integration, required that a majority of the core control system software functionality be implemented, and notionally tested, without access to the related hardware. We present the approaches to simulating the HET hardware systems, their evolution, the successes and shortcomings of the current level of capabilities, and ideas for improvements to simulation and integration testing.

Keywords: Hobby-Eberly Telescope, HET, Control Systems, Software, Computer Programming, Integration, Testing

1. INTRODUCTION

Post wide field upgrade[1–3] (WFU) the HET Telescope Control System (TCS) is a distributed set of Linux processes, TCS subsystems built of a common application framework,[4] ultimately binding a Python scripting layer to a variety of interfaces that set the telescope hardware components. TCS is comprised of 6 major subsystems: the primary orchestrator, the gantry tracker, an interface to the moving components of the focal surface and calibration lamps and actuators, metrology instrument readout, a wrapper around several legacy subsystems which include the dome, structure, and weather sensors and power management. Respectfully, these are referred to as tcs, tracker, pfip, pas, legacy, and apc. The power management subsystem interfaces with a variety of power distribution units, though is sometimes referred to as apc for historical reasons.

Subsystem	Interconnects	Hardware
tcs		n/a
tracker	sockets	dSPACE
pfip	EPICS, ModBus	cRIO, Phoenix Contact
pas	vendor APIs, sockets	AVT, PointGray, FLI
legacy	sockets	miscellaneous

Table 1: A condensation of the relationship of TCS subsystems to the hardware.

Designed from the same common framework, the CCD array management and readout application (camra), abstracts the VIRUS[5–7] and LRS2[8] spectrograph arrays. This is accomplished with two separate instances of camra, to avoid a common PCIe bus. These processes manage an array of over 40, with a goal of 80, readout controllers, with the LRS2 camra instance handling 2 of the spectrographs and VIRUS the remainder.

Software and Cyberinfrastructure for Astronomy V, edited by Juan C. Guzman, Jorge Ibsen, Proc. of SPIE
Vol. 10707, 1070736 · © 2018 SPIE · CCC code: 0277-786X/18/$18 · doi: 10.1117/12.2314471

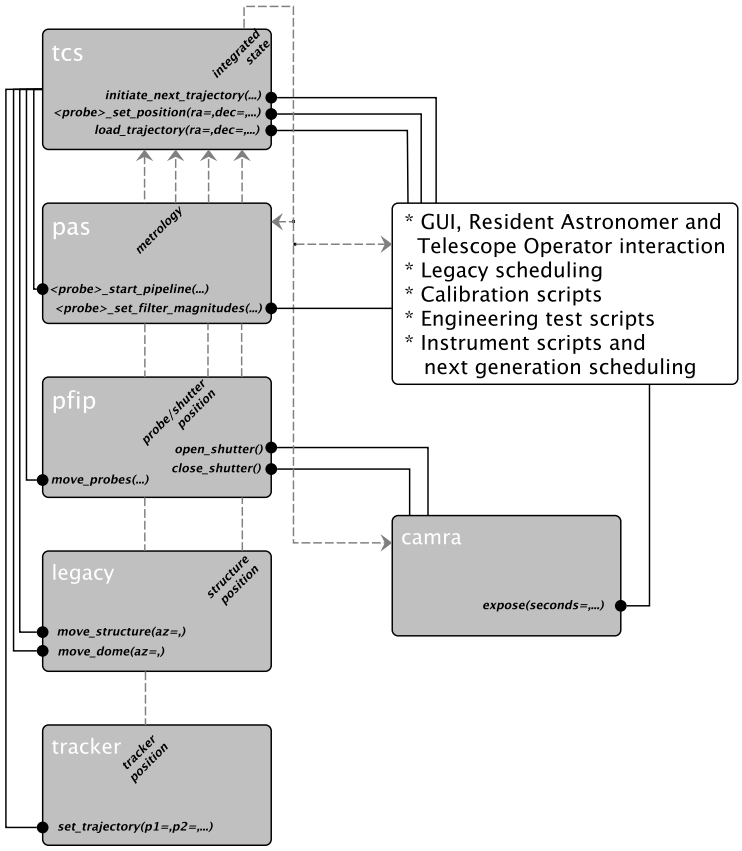


Figure 1: A diagram of a subset of the relationships between TCS and instrument subsystems, in gray, and typical orchestration layers summarized in the remaining block. The solid lines are RPC handler calls and the dashed are broadcast events. Note that certain method names used in the diagram have been normalized to a common standard while the corresponding source code is not.

A myriad of approaches to testing and simulation evolved from the needs of the individual sub-projects. Given the distributed nature of the subsystems, a testing suite that simplifies the orchestration of high level plumbing, regression, and analysis tests is crucial. A sample of a typical TCS orchestration is shown in Fig.1. The actions shown are supported by automatable offline TCS scripts.

While the TCS testing toolkit does a sufficient job of emulating a good portion of the hardware, a push button confidence has yet to be achieved. The scriptable nature of each subsystem makes test driver development trivial, while the successful emulation of hardware remains daunting in some cases. From simple netcat shell emulators to coding in emulation at the application layer to tcs event playback, the spectrum of testing strategies provides insight into many aspects of the control system. However, a more homogeneous approach to testing will be required for long term maintenance.

The operational expenses associated with a telescope of this scale require a minimization of downtime stemming from issues with software development and/or maintenance. Likewise, exhausting what would otherwise be valuable science time for testing and debugging purposes has immeasurable detrimental impact on the telescope, and its users. The next phase of TCS development, as the software moves into full 24/7 science operations, with multiple instruments already deployed and being actively commissioned, will require a more complete set of simulation capabilities for offline and automated testing.

2. HARDWARE EMULATION

2.1 Tracking

The base case for simulating a subset of the TCS is the successful emulation of the HET tracker hardware. The HET Tracker Motion Control System (TMCS),[9] a Matlab/Simulink model presented as a black box to the TCS software team, has only a few simple interfaces for moving the tracking mechanism and drip feeding trajectory

points to the tracker, a sliding window of 10 points at 1Hz. The interface to the dSPACE hardware, via TMCS, is a socket pair on both sides of the wire implementing transmit and receive channels conveying a watchdog timeout handshake, a status message from the hardware, and command/acknowledgement between tracker and TMCS.

A majority of the available development time for the tracker overlapped with the schedule for dismantling the tracking hardware, shipping it to HET, and reassembling in situ.[2,10–12] The challenge of a tightly constrained timeline for testing the tracker software, and related TCS integration, driven by the need to disassemble the tracking mechanism for transport to the observatory, was met with a simple Bash shell script and off-the-shelf Linux command line utilities.

The core of the TMCS emulator functionality comes from the netcat, http://nc110.sourceforge.net, utility. This is a socket wrapper that allows one to easily configure sockets from the command line, establishing I/O between remote channels on file descriptors. Leveraging netcat (`nc`), the tracker hardware emulator is reduced to the following command line:

```
$ nc -l -k ${ACCEPT} | validate | (while true; do nc -C localhost ${CONNECT}; done)
```

Reading left to right, the first `nc` establishes a socket listening on a port given by the `${ACCEPT}` variable and writing packets received to standard output. The standard output of the first `nc` is directed to the standard input of the shell function `validate`. The standard output of this shell function is then directed to the standard input of a while loop maintaining a persistent `nc` client instance connected via port `${CONNECT}` to the tracker software. The shell function providing the emulation, `validate`, was originally implemented simply to test and validate TMCS command strings from the tracker. This function evolved to provide the near full-featured emulation in use today.

The `validate` method implements a line fed filter transforming each line of information read from the accept socket into one or more lines of response to the tracker software. Each line received is echoed to standard error for logging and/or further chaining of test utilities. Additionally, a watchdog message is emitted on read timeout boundaries, rather than a concurrent realtime cycle. This sufficiently simulates the required 1Hz rate of the watchdog signal while assuming some flexibility in driving that signal at a higher and varied rate, in response to each command. The emulator supports the full command set of TMCS.

The usefulness of the TMCS emulator is challenged by the need to test the realtime interaction of the tracker and tcs subsystems while capturing the full state change response, both spatially and temporally. For trajectory position commands the emulator has an instantaneous response where the position and clock of the tracker are immediately set to that of the commanded position. This prevents the simulator from being of use in testing and debugging subtle timing issues in the transfer of information between tcs, tracker, and TMCS. Although the interpolation to realtime positions provided by TMCS is not replicated in the simulator, all of the tracker and tcs code paths related to tracker moves, aborts, and trajectory feed may be exercised in the absence of the tracker hardware.

2.2 Metrology

The pas subsystem implements readout and processing for metrology[13–15] instrumentation, providing an event stream from each data source to tcs. Metrology streams are filtered, transformed to the appropriate coordinate frame and aggregated in tcs for application to the trajectory points fed to TMCS via the tracker.

For testing, the metrology streams for the various cameras, including the tip-tilt, guide, and wavefront sensing, are synthesized from a sequence of images read from disk. This simulated metrology stream from pas is implemented via simple configuration changes which switch the image processing pipeline from reading frames from a camera to reading them from disk. These capabilities are sufficient enough for plumbing and limited integration testing, however, the ideal for this simulator would be to perturb the image sequence feed, as read from disk or perhaps synthesized, based on the current state of the integrated TCS. This approach to testing requires that the pas process be modified to implement the switching, which means that at some depth of stack not all code paths are tested without attaching to a live camera.

The tcs implements commands allowing for the injection of metrology events from any of the metrology streams at any desired rate. This allows for the metrology control loops in tcs to be tested independent of both the pas hardware as well as the pas subsystem.

```
dx=dy=dw=0
for i in range(1,101):
        tcs.WFS1_inject_metrology(dx=dx,dy=dy,dw=dw)
        dx,dy,dw=perturb_offsets(i,dx,dy,dw)
        time.sleep(cycle_time)
```

An instantaneous relative focal distance is read by pas, from the Distance Measuring Interferometer (DMI).[15] The DMI is simulated in a manner similar to the TMCS, with `nc`, manipulating sockets and a shell function responding to queries made by pas on a configurable polling interval. The initial motivation for the DMI simulator stemmed from the need to test the related metrology algorithms against noise and outliers. For that reason, the parameterization of the simulator drives a fake pseudo-random signal from the interferometer on a hard coded localhost port. As with the image feed, the DMI simulator would ideally respond to telescope state changes as well but this remains left for a future exercise. With the tracker position to mirror segment mapping available, the DMI simulator should be able to accurately represent primary segment transitions as well.

2.3 Focus and calibration instrumentation

Moving components of the guide, focus, calibration and primary shutter hardware, in addition to calibration sources, are wrapped by the pfip subsystem.[16,17] This requires pfip to juggle multiple protocols when communicating with the relevant hardware. Assumed complexity in emulating ModBus and/or EPICS led to a different approach in satisfying offline testing needs with pfip.

Current hardware emulation strategies for pfip maintain a configurable global "disable hardware" flag. This flag is used to short circuit communication with the underlying hardware. In some cases, pfip is coded to update presumed state providing a piecewise hardware emulation. The hardware segments of pfip covered by this strategy have thus far been defined by the needs of other subsystem's integration testing.

A modular approach to the implementation of these emulation features is satisfactory from the perspectives of current integration and plumbing tests. This approach requires that each new code path tested offline be properly adapted for testing. The strategy does not preclude the addition of new hardware interfaces, or changes to the existing interfaces, since it avoids the hardware and related interfaces altogether.

2.4 Legacy services

The legacy subsystem wraps services which were developed for the original HET, prior to the WFU. Interfaces for the dome, structure, dome shutter, and weather are all wrapped by legacy. With the structure critical to the setup and positioning algorithms in tcs, offline integration testing requires some degree of simulation.

For the hardware managed by legacy, each implementing a subtly different protocol, the socket spoofing approach employed for the tracker and DMI is leveraged. The success of this is currently limited to read only interaction with the structure and dome components. The structure emulator may be configured for any static azimuth, but does not yet support 2-way communication. While less than ideal, this still allows for integration testing with a predetermined and fixed azimuth for the duration of the execution of the simulator. The script may easily be called iteratively.

The dome and dome shutter are simulated via netcat shell script as well, though that effort has yet to be vetted. Integration testing short circuits commanded moves of the dome, for now, and the actuation of the dome shutter is not handled in any automated fashion, but left to the telescope operator.

2.5 CAMRA

The camra leverages the vendor API and kernel driver to communicate with readout CCD controllers over a PCIe bus.[18] The readout software was required early in tcs subsystem development due to a need to verify the functionality of the VIRUS spectrographs already well under production. With complications related to readout hardware stability, and sparse hardware availability in the lab, there was an immediate need for offline testing that supported decoupling bugs in software from bugs in hardware.

The massively multiplexed hardware architecture of the VIRUS complicates any efforts to mimic the hardware and readout. Meeting this challenge, the camra software was designed to simply wrap all calls into the vendor API and catch any error generated. If flagged for "debug" mode, camra will catch the exception generated upon opening the PCIe bus and produce a hardware state data structure that reflects a dummy readout system spanning 8 controllers. The data structure is then used to synthesize an array of 1 multiplexer (mux) and 8 readout controllers, 16 CCDs, or a total of 32 simulated readout amplifiers. In this mode, the software remains up in the absence of hardware and all code paths can be exercised down to terminal calls into the wrapped API. This is similar to the approach taken with pfip in Sec. 2.3, but more wholistic.

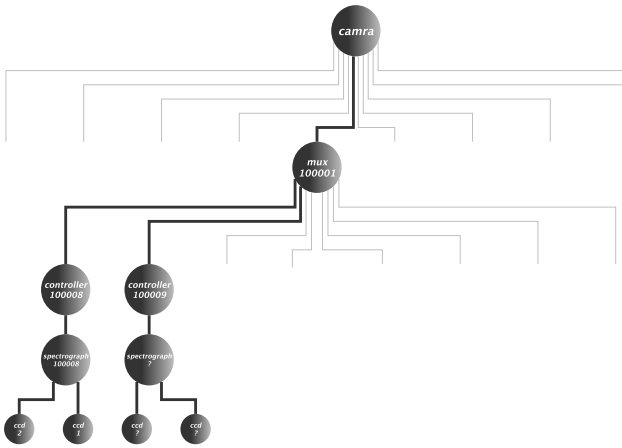


Figure 2: In testing, we fabricate a VIRUS-like array with a single mux and 2 dummy controllers, one of which is fully configured. This is the baseline used in testing new features that interact with the hardware and/or related configuration. The empty branches represent the scale of the eventual hardware for the VIRUS array. Test cases may easily be scaled up to the full hardware footprint, or beyond.

Standard sentinel values are used for fields that have no meaning in the absence of a configuration that provides the implicit mapping. See the log in Listing 1 showing a small portion of the camra startup associated with a test run in the absence of hardware. In this configuration we can fully interact with camra, including exposures that generate FITS files on disk for each of the 8 simulated amplifiers. With the other subsystems up and an active trajectory, we can generate fully populated headers as well.

Listing 1: Log messages from camra running on a development laptop, void of readout cards. In this case we have the array coming up with 8 controllers, two of which match supplied configurations. The first configuration, for controller 100008, specifies two parameters that trigger the setting of clock voltages, so we see the exceptions from those attempts. The second controller being configured is missing several identifiers so we see the defaulting of these to sentinel values. Note that time and source file components of the log message have been removed for the sake of readability.

```
- Exception opening hardware: ( CArcPCIe::FindDevices() ): No device bindings exist! Make sure an ARC, Inc
    PCIe card is installed!
- Allocating controller 1 of 0.
- Initializing controller 100008 at index 0 in mux 100001.
- ( CArcSys::Command() ): No devices are open!
- Setting CCD 2 reset_gate_high clock voltage to 6.000000 Volts with correction coefficients of m=0.999000,b
    =0.001000, an effective ADU count of 188.
- Exception applying clock voltages for CCD 2 on controller 100008: ( CArcSys::Command() ): No devices are
    open!
- ( CArcSys::Command() ): No devices are open!
- Setting CCD 1 reset_gate_high clock voltage to 6.000000 Volts with correction coefficients of m=0.999000,b
    =0.001000, an effective ADU count of 188.
- Exception applying clock voltages for CCD 1 on controller 100008: ( CArcSys::Command() ): No devices are
    open!
- Initialized controller 100008 with left CCD 2 and right CCD 1.
- Allocating controller 2 of 0.
- Initializing controller 100009 at index 1 in mux 100001.
- No identifier given for IFU on controller 100009, assigning 200017.
- No identifier given for spectrograph on controller 100009, assigning -999999.
- No slot given for IFU on controller 100009, assigning 300017.
- No identifier given for left CCD on controller 100009, assigning 100018.
- No identifier given for right CCD on controller 100009, assigning 100019.
- No characterization controller ID found for spectrograph -999999 on controller 100009.
```

Another testing strategy employed by camra is compile time shutter failure triggering. The potential of hours long exposures opens the door to a variety of failures, between shutter open and close. Additionally, the

breadth of coordinated activities in TCS associated with an instrument exposure requires special handling of the tear down from an exposure where a shutter delay, fault, or failure occurs. For this reason, simple directives are added to the relevant code paths to cause the software, in the absence of a live connection to pfip and the underlying hardware, to fail. This allows the camra handling of the failure to be tested and vetted without complex coordination and scripting involving the hardware. The nearest alternative, catching hardware in the act of failure, is rarely an efficient approach to troubleshooting. This testability feature is easily migrated to runtime configuration, if necessary.

Listing 2: The camra client side of the primary shutter open command.

```
void PFIPClient::open_shutter( void )
{
    tthread::lock_guard<tthread::mutex> lock(mutex_);
    TLOG_DEBUG( "Opening PFIP shutter." );

#ifdef __SIMULATE_SHUTTER_OPEN_SUCCESS
    return;
#endif

#ifdef __SIMULATE_SHUTTER_OPEN_FAILURE
    tthread::this_thread::sleep_for( tthread::chrono::seconds(1) );
    CAMRA_EXCEPTION( "Test exception from shutter open." );
#endif

    if( active_ )
        delete send( "OpenShutter", NULL );
}
```

Array hardware remains sparse in the development lab, with any complete sets being installed at HET as soon as they are characterized.[19] The scale at which some new features must be tested requires greater than 8 readout controllers, seven more than are typically available in the lab. Testing whether a configuration is properly applied to each of hundreds of CCDs in the array is best accomplished through simulation. The sheer number of parameters and recent additions to the camra configuration hierarchy require a great deal of efficiency in the testing of the new logic and interfaces. The hardware spoofing approach has proven highly effective for camra testing and enabled minimal iteration on maximal feature additions.

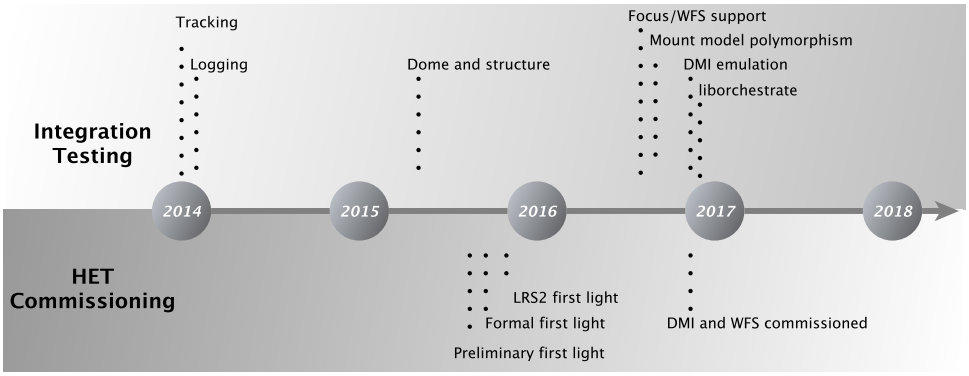
3. ORCHESTRATION AND INTEGRATION

One key apparatus in the WFU software effort has been a simple Python orchestration library and integration test script, and the related command line interfaces. This utility, through a concise command line parameterization, spawns the tcs subsystems, including related hardware emulators, and orchestrates a sidereal trajectory, by default. The script is configurable to include launching visualization utilities, the TCS GUI, and constraining the various actions taken during the orchestration. The script may be flagged to skip loading and running the default trajectory, but rather sit in an idle state. This is useful for debugging, where one may attach to any of the tcs subsystems with gdb, or kill any spawned by the script and re-launch them from an IDE of choice, maintaining the integration of the remainder of the subsystems.

3.1 Development cycles

The integration test script has evolved over two major cycles. Initially, we simply needed an easy method for testing the trajectory feed from tcs to tracker to TMCS while the tracker hardware remained offline for transport and assembly. What began as one of many wrapper scripts around a momentary testing need quickly evolved into a tcs subsystem wiring and orchestration manager. Unfortunately, we have not yet captured, in automated form, the series of integration tests needed to fully cover the code.

In the second phase, the script was handed off to an undergraduate student for "Pythonization". This resulted in a new subprocess management library and generalized interface to launching subsystems. The new intellect in subprocess management cleaned up numerous issues with orphaned processes making the script viable for automated integration testing. Flexibility in subsystem logging verbosity and additional fine tuning of the trajectory parameters were added enabling directed testing of trajectory calculation and coordinate transform libraries.



3.2 Features

Pointing options supported by the integration test are limited to sidereal trajectories, for now. These allow for setting the right ascension, declination, azimuth, equinox and the direction of the requested track. Altitude is coupled to azimuth given the HET mount. Offsets in either right ascension or declination may be applied. These are useful for analyzing response to grid pointing or simply moving the trajectory in time.

The console log of the integration script logs the actions taken by the script but may be augmented, via command line flagging, to include the console log output of any of the wrapped subsystems. This is useful when debugging a particular subsystem's responses to the test script's actions. The script supports enabling plotters that display tracker position, both as linear plots and OpenGL display. The OpenGL display is implemented as a TCS subsystem, consuming the same event stream as other subsystems.

A command line option allows the user to enable the TCS database. This results in all event traffic being recorded to a trivially queried SQLite database. This feature is used to generate Figures 3 and 4. An additional option enables the RPC message traffic between subsystems to be captured in the database as well. By extension of options to tcs, the script also supports disabling various mount models.

4. CURRENT WORK

4.1 ρ-offset

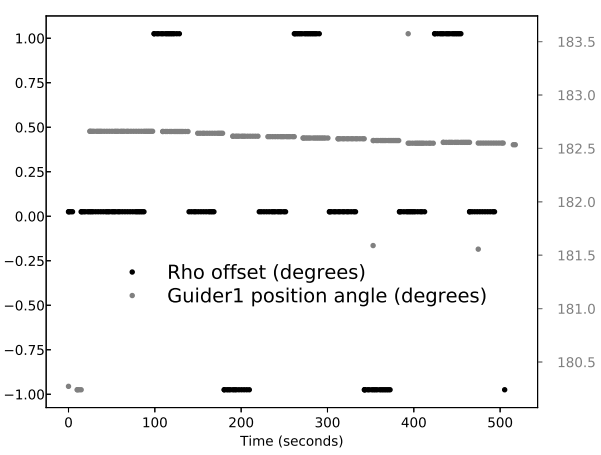
The derotation stage of the HET is referred to by ρ, in the tracker coordinate frame. This derotation is applied along the trajectory to keep the parallactic angle of the observation fixed and maintain guide and wavefront sensors in a static configuration, with respect to the sky, at 8 to 11′ off axis.

Current efforts are underway to implement an offset in TCS, where the natural trajectory ρ is offset by some number of degrees. This is expected to enable repeat observations of blind setups by maintaining a consistent parallactic angle over inconsistencies in the structure setting capabilities. Multiple subsystems are involved in the coordination of this functionality. The offset is implemented as a register in tcs, which is applied additively to the commanded ρ of the current trajectory. The moving probes are positioned, and their state read, by the pfip subsystem. The probes are moved sympathetic to the applied offset in order to maintain the same field, before and after the offset.

Geometrically, the positional information is not tied to the sky but rather the mechanical frame of the pfip. However, the initial implementation of the pfip related coordinate transforms included a zero point which partially tied the ideal frame of the pfip mechanics to the tracker via ρ. Furthermore, this inconsistency was not consistent between subsystems.

Using the integration test script to bootstrap the software and capture results in a database, and a simple command line loop to drive the test, we apply a $\pm 1^\circ$ offset in ρ. There is roughly a 30 second dwell time at each offset. From this, two metrics are used to provide initial insights into the implementation and performance of

Figure 3: An alternating ρ-offset of $\pm 1^\circ$ is applied along a sidereal trajectory. The initial discrepancy between the commanded right ascension and declination is effectively zero. We see a drift rate of $\sim$.01" per-minute of time along the zero valued offset. Note that this systematic is present in the simulated system, so we know that these are not components of any physical error. Though this appears as though we were actively guiding on an error that changes with ρ, the metrology systems were disabled for this test run. We see that the offset, on the scale of 1°, induces an initial $\sim$.5" error in the inverse of the commanded pointing, and the error goes with the sign of the offset. Also note the flip in the sign of the drift.



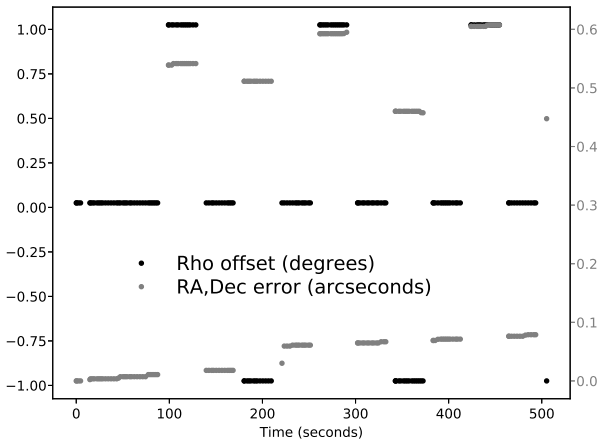


Figure 4: As the offset is applied, we see that the guide probe's reported position angle remains consistent, less the drift. The dropouts in guide probe position angle are due to a squelching of the reporting while the offset is being applied and each probe assembly is rotated to compensate for the change in derotation of its mount. We see some leakage of the probe position, in the outliers from the trend approaching 182.5°, which must be dealt with prior to taking the new functionality on-sky for science. These are at approximately 350, 400, and 475 seconds into the run. These are samples of probe position from mid-move and the corresponding image from the probe must be ignored by the metrology. However, the presence of those samples in this plot indicates they would be associated with valid metrology frames in real-time.

the new feature. We see that there is an inherent drift to the pointing inversion and that the application of the offset is not properly accounted for. Figure 3 shows that we are consistent with the reported pointing, less the drift, before applying the offset. However, there are clear problems with the non-zero offset. Secondly, shown in Figure 4, the reported position angle for the movable probes must remain consistent over the compensatory moves.

Ignoring the outstanding issues with the implementation, since they were at a manageable level and could be guided out, the new code was taken on sky during a recent engineering period. In spite of offline testing capabilities, we immediately found a sign bug thanks to the feedback from the HET. The sign bug came in a pair and, though we corrected one in situ to achieve the proper pointing, we only found the second after rerunning the integration testing and noticing that the reported guider position angles oscillated by twice the applied offset.

Without the perspective given by the test utilities, we would now be in the unfortunate position of mixing these systematics into the next iteration on empirical pointing error models. Regardless of the eficacy, these tests remain quite manual in terms of the amount of coding that goes in to generating the picture. Ideally we will soon automate the daily reporting of these frames of reference.

4.2 Probe move semantics

Slow moving guide and wavefront sensing probes, being adjusted to compensate for changes to components with much faster responses, i.e. the tracker, present an interesting problem for user feedback and interaction. One example is the offset described in the previous section. While the probes are slowly moved to compensate for a change in the tracker, the operators may still be presented with a prior guide frame. They may even attempt to

make an offset relative to that frame. This is currently supported by queueing the subsequent move and waiting for the pending rotation to finish. Originally, the subsequent offset would have resulted in an exception response.

Given this subjective requirement, numerous iterations have been made on the related code paths in order to increase operator efficiency and clarify the instantaneous state of the system during these offsets and moves. Behavioral issues of this nature require operator interaction for full testing and development, but integration testing methods ensure that the simple details of new functionality make basic sense, prior to consuming valuable telescope or operator time.

4.3 Pointing library refactoring

The initial pointing code, held over from the legacy telescope controls, and whittled on by newcomers to the WFU project, had numerous maintainability issues. These were primarily due to a partial port from Fortran to C to C++, over many years and through many developers.

Deciding to fix the implementation, to remove hundreds of lines of code from TCS and cleanup the API, we proceeded cautiously. Comparing the output of calculations, from both the library's test framework and runtime queries of tcs, we found that the calculated tracks were equivalent, before and after the refactoring work. So the changes were deployed.

For an entire science period, approximately three weeks around new moon, we saw an increase in mean pointing error on the order of 10", with the standard deviation of the error going from 2.9 to 7.6". We had missed something simple. Though the instantaneous trajectory calculations were equivalent, we had missed properly preserving some internal state within the pointing library. This internal state was used to properly recalculate a trajectory from an existing track and deltas in right ascension, declination, altitude, and/or azimuth. This recalculation is applied at the start of a track to correct for any error in structure azimuth, after settling, so each time this correction was made an error commensurate with the target azimuth would be seen during setup.

This bug manifested only for azimuth recalculations, so on-sky offsetting through the GUI, by deltas of right ascension and declination with fixed azimuth, still performed as expected. The bug was subtle and only caught after parallel efforts to refine pointing models and analyze setup times showed the approximately month long spike in pointing error. Testing this is trivial, in hindsight, where the results of the recalculation for a change in azimuth are compared. This clearly points out deficiencies in the current test budgeting and strategies.

5. LESSONS LEARNED AND FUTURE WORK

Throughout TCS development, some strategies for testing have proven more efficient than others. Subsystems that implemented some sort of testing during initial development proved, subjectively at a minimum, to be more amenable to change, supporting a more agile development model. There are a few key places where a redesign of the offline approach to testing might be beneficial.

The PFIP server implements EPICS and ModBus APIs for hardware interconnect. These may be refactored such that they support flagging to short-circuit interaction with the hardware. This would allow a developer supplied callback driven layer to synthesize hardware state in a generalized manner, for any components that might need to communicate over EPICS or ModBus. Application code could be written without special code paths for handling the offline use case.

The output database and/or console log of the integration test utility can be used to compare different runs of the test script. This may be implemented by rewriting the time stamps with the earliest referenced to zero. This would support gold standard testing and full regression testing for any code paths spanned by the test script.

Analysis scripts are particularly challenging, with each engineer and scientist favoring their own environment. Expanding the TCS database bindings to include Matlab will make some efforts easier. A framework is needed, to streamline scientist's and engineer's analysis efforts into a daily snapshot of where the HET has been.

6. CONCLUSION

Hardware emulation and integration testing for HET are far from complete. In early development, the need for common approaches to testing was outweighed by the acceptance of an evolutionary approach to integration, favoring unit functionality. The present capabilities provide an efficient normalization of the various approaches. Maintaining the integration testing capabilities at a level beyond the present feature set, and coding to the test, prove to be invaluable and must remain a top priority moving into TCS maintenance mode.

ACKNOWLEDGMENTS

HETDEX is run by the University of Texas at Austin McDonald Observatory and Department of Astronomy with participation from the Ludwig-Maximilians-Universität Mnchen, (LMU) Max-Planck-Institut für Extraterrestriche-Physik (MPE), Leibniz-Institut für Astrophysik Potsdam (AIP), Texas A&M University (TAMU), Pennsylvania State University (PSU), Institut für Astrophysik Göttingen (IAG), University of Oxford, Max-Planck-Institut für Astrophysik (MPA) and The University of Tokyo. In addition to Institutional support, HETDEX is funded by the National Science Foundation (grant AST-0926815), the State of Texas, the US Air Force (AFRL FA9451-04- 2-0355), by the Texas Norman Hackerman Advanced Research Program under grants 003658-0005-2006 and 003658-0295-2007, and by generous support from private individuals and foundations. We thank the staffs of McDonald Observatory, HET, AIP, MPE, TAMU, IAG, Oxford University Department of Physics, the University of Texas Center for Electromechanics, and the University of Arizona College of Optical Sciences for their contributions to the development of the HET WFU and VIRUS.

REFERENCES

[1] Hill, G. J., Drory, N., Good, J., Lee, H., Vattiat, B., Kriel, H., Bryant, R., Elliot, L., Landriau, M., Leck, R., Perry, D., Ramsey, J., Savage, R., Allen, R. D., Damm, G., DePoy, D. L., Fowler, J., Gebhardt, K., Haeuser, M., MacQueen, P., Marshall, J. L., Martin, J., Prochaska, T., Ramsey, L. W., Rheault, J.-P., Shetrone, M., Schroeder Mrozinski, E., Tuttle, S. E., Cornell, M. E., Booth, J., and Moreira, W., "Deployment of the Hobby-Eberly Telescope wide field upgrade," *Proc. SPIE* **9145**, 914506–914506–19 (2014).

[2] Hill, G., Drory, N., Good, J., Lee, H., Vattiat, B., Kriel, H., Ramsey, J., Randy Bryant, R., Elliot, L., Fowler, J., Landiau, M., Leck, R., Odewahn, S., Perry, D., Savage, R., Schroeder Mrozinski, E., Shetrone, M., Damm, G., Gebhardt, K., MacQueen, P., Martin, J., Armandroff, T., and Ramsey, L., "The Hobby-Eberly Telescope wide-field upgrade," *Proc. SPIE* **9906-5** (2016).

[3] Hill, G., Drory, N., Good, J., Lee, H., Vattiat, B., Kriel, H., Ramsey, J., Bryant, R., Fowler, J., Landriau, M., Leck, R., Mrozinski, E., Odewahn, S., Shetrone, M., Westfall, A., Terrazas, E., Balderrama, E., Buetow, B., Damm, G., MacQueen, P., Martin, J., Martin, A., Smither, K., Rostopchin, S., Smith, G., Spencer, R., Armandroff, T., Gebhardt, K., and Ramsey, L., "Completion and performance of the hobby-eberly telescope wide field upgrade," *Proc.SPIE* , 10700–20 (2018).

[4] Ramsey, J., Drory, N., Bryant, R., Elliott, L., Fowler, J., Hill, G. J., Landriau, M., Leck, R., and Vattiat, B., "A control system framework for the hobby-eberly telescope," *Proc.SPIE* **9913**, 9913 – 9913 – 9 (2016).

[5] Hill, G., Tuttle, S., Vattiat, B., Lee, H., Drory, N., Kelz, A., Ramsey, J., DePoy, D., Marshall, J., Gebhardt, K., Chonis, T., Dalton, G., Farrow, D., Good, J., Haynes, D., Indahl, B., Jahn, T., Kriel, H., Montesano, F., Nicklas, H., Noyola, E., Prochaska, T., Allen, R., Blanc, G., Fabricius, M., Landriau, M., MacQueen, P., Roth, M., Savage, R., and Snigula, J., "VIRUS: first deployment of the massively replicated fiber integral field spectrograph for the upgraded Hobby-Eberly Telescope," *Proc. SPIE* **9908-54** (2016).

[6] Tuttle, S. E., Hill, G. J., Lee, H., Vattiat, B., Noyola, E., Drory, N., Cornell, M., Peterson, T., Chonis, T., Allen, R., Dalton, G., DePoy, D., Edmonston, D., Fabricius, M., Haynes, D., Kelz, A., Landriau, M., Lesser, M., Leach, B., Marshall, J., Murphy, J., Perry, D., Prochaska, T., Ramsey, J., and Savage, R., "The construction, alignment, and installation of the virus spectrograph," *Proc.SPIE* **9147**, 9147 – 9147 – 13 (2014).

[7] G.J., H., Kelz, A., Lee, H., MacQueen, P., Peterson, T., Ramsey, J., Vattiat, B., DePoy, D., Drory, N., Gebhardt, K., Good, J., Jahn, T., Kriel, H., Marshall, J., Montesano, F., Tuttle, S., Balderrama, E., Chonis, T., Dalton, G., Fabricius, M., Farrow, D., Fowler, J., Froning, C., Haynes, D., Indahl, B., Martin, J., Montesano, F., Mrozinski, E., Nicklas, H., Noyola, E., Odewahn, S., Peterson, A., Prochaska, T., Shetrone, M., Smith, G., Snigula, J., Spencer, R., and Zeimann, G., "Virus: status and performance of the massively replicated fiber integral field spectrograph for the upgraded hobby-eberly telescope," *Proc.SPIE* , 10702–56 (2018).

[8] Chonis, T. S., Hill, G. J., Lee, H., Tuttle, S. E., Vattiat, B. L., Drory, N., Indahl, B. L., Peterson, T. W., and Ramsey, J., "Lrs2: design, assembly, testing, and commissioning of the second-generation low-resolution spectrograph for the hobby-eberly telescope," *Proc.SPIE* **9908**, 9908 – 9908 – 28 (2016).

[9] Beno, J. H., Hayes, R., Leck, R., Penney, C., and Soukup, I., "Hetdex tracker control system design and implementation," *Proc.SPIE* **8444**, 8444 – 8444 – 15 (2012).

[10] Good, J., Booth, J., Cornell, M. E., Hill, G. J., Lee, H., Savage, R., Leck, R., Kriel, H., and Landriau, M., "Laboratory performance testing, installation, and commissioning of the wide field upgrade tracker for the Hobby-Eberly Telescope," *Proc. SPIE* **9145**, 914546–914546–11 (2014).

[11] Good, J. M., Hill, G. J., Landriau, M., Lee, H., Schroeder-Mrozinski, E., Martin, J., Kriel, H., Shetrone, M., Fowler, J., Savage, R., and Leck, R., "HET Wide Field Upgrade Tracker System Performance," *Proc. SPIE* **9906-167** (2016).

[12] Good, J., Leck, R., Ramsey, J., Drory, N., Hill, G., Fowler, J., Kriel, H., and Landriau, M., "Mechanical systems performance of the het wide-field upgrade," *Proc.SPIE* , 10700–143 (2018).

[13] Lee, H., Hill, G. J., Cornell, M. E., Vattiat, B. L., Perry, D. M., Rafferty, T. H., Taylor, T., Hart, M., Rafal, M. D., and Savage, R. D., "Metrology systems of hobby-eberly telescope wide field upgrade," *Proc.SPIE* **8444**, 8444 – 8444 – 14 (2012).

[14] Lee, H., Hill, G., Drory, N., Ramsey, J., Bryant, R., and Shetrone, M., "Wavefront sensing for active alignment control of a telescope with dynamically varying pupil geometry: theory, implementation, on-sky performance," *Proc.SPIE* , 10706–150 (2018).

[15] Lee, H., Hill, G., Drory, N., Vattiat, B., Ramsey, J., Bryant, R., Shetrone, M., Odewahn, S., Rostopchin, S., Landriau, M., Fowler, J., Leck, R., Kriel, H., and Damm, G., "New hobby eberly telescope metrology systems: design, implementation, and on-sky performance," *Proc.SPIE* , 10700–78 (2018).

[16] Vattiat, B., Hill, G. J., Lee, H., Moreira, W., Drory, N., Ramsey, J., Elliot, L., Landriau, M., Perry, D. M., Savage, R., Kriel, H., Häuser, M., and Mangold, F., "Design, alignment, and deployment of the Hobby-Eberly Telescope prime focus instrument package," *Proc. SPIE* **9147**, 91474J–91474J–12 (2014).

[17] Vattiat, B., Hill, G. J., Lee, H., Moreira, W., Drory, N., Ramsey, J., Elliot, L., Landriau, M., Perry, D. M., Savage, R., Kriel, H., Huser, M., and Mangold, F., "Design, alignment, and deployment of the hobby eberly telescope prime focus instrument package," *Proc.SPIE* **9147**, 9147 – 9147 – 12 (2014).

[18] Hill, G. J., Tuttle, S. E., Drory, N., Lee, H., Vattiat, B. L., DePoy, D. L., Marshall, J. L., Kelz, A., Haynes, D., Fabricius, M. H., Gebhardt, K., Allen, R. D., Anwad, H., Bender, R., Blanc, G., Chonis, T., Cornell, M. E., Dalton, G., Good, J., Jahn, T., Kriel, H., Landriau, M., MacQueen, P. J., Murphy, J. D., Peterson, T. W., Prochaska, T., Nicklas, H., Ramsey, J., Roth, M. M., Savage, R. D., and Snigula, J., "Virus: production and deployment of a massively replicated fiber integral field spectrograph for the upgraded hobby-eberly telescope," *Proc.SPIE* **9147**, 9147 – 9147 – 27 (2014).

[19] Indahl, B. L., Hill, G. J., Drory, N., Gebhardt, K., Tuttle, S., Ramsey, J., Ziemann, G., Chonis, T., Peterson, T., Peterson, A., Vattiat, B. L., Li, H., and Hao, L., "Virus characterization development and results from first batches of delivered units," *Proc.SPIE* **9908**, 9908 – 9908 – 16 (2016).