RMxAC, **53**, 52–58 (2021)
© 2021: Instituto de Astronomía, Universidad Nacional Autónoma de México
https://doi.org/10.22201/ia.14052059p.2021.53.14

VI Workshop on Robotic Autonomous Observatories (Mazagón, Huelva (Spain), September 30 - October 4, 2019)
Editors: A. J. Castro-Tirado, S. B. Pandey, & M. D. Caballero-García - DOI: https://doi.org/10.22201/ia.14052059p.2021.53.14

LBTO TCS SOFTWARE

Petr Kubánek[1]

RESUMEN

Se presenta una descripción general del software de control del Observatorio del Gran Telescopio Binocular. Resaltamos cómo el sistema es capaz de operar un robot complejo y multiejes, que resulta ser uno de los telescopios astronómicos mayores del planeta. Se detallan las interfaces del usuario observador y del operador del telescopio. Se concluye con los cambios ya realizados y planeados para los próximos años.

ABSTRACT

This article presents an overview of the Large Binocular Telescope Observatory's Control Software. It is focused on how the system is capable of driving a complex, multi-axis robot - which happens to be one of the largest astronomical telescopes. Observer and telescope operator user interfaces are discussed. The article concludes with changes already performed and planned for the next few years.

Key Words: telescopes

1. INTRODUCTION

The Large Binocular Telescope Observatory (Hill et al. 2008) *LBTO*, is an organization operating one of the largest optical telescopes. The telescope itself is located at Mount Graham International Observatory in southeast Arizona. The telescope's unique design - incorporating two borosilicate honeycomb mirrors mounted on a single mount, three possible focus configuration, active and adaptive optics, and state of the art instruments - requires unique control software. The package, referred to as the LBTO TCS *(Telescope Control System)*, consists of various libraries and binaries written usually in *C* or *C++* languages. TCS development spans over two decades, with thousands man-hours spend on the project. It's hard to measure the size of the resulting code, but it definitely consists of a few hundred thousand lines of code, if not a little over million.

2. LBTO

The LBTO design's most unique features - placing two big mirrors on a single mount - present a challenge for TCS.

The telescope optical assembly consists of three mirrors per side - 8.4m primary (**M1**), 91cm secondary (**M2**) and 50cm tertiary (**M3**). Secondary mirror comes in rigid and deformable (for Adaptive Optics) flavors.

2.1. *Focal stations*

The telescope design offers three distinctive focal station per side. There is a prime focus station above **M1**, utilizing only the primary mirror. Direct Gregorian focus is located below **M1** and needs **M2** deployed. Bent Gregorian foci, located on side of the **M1**, needs both **M2** and **M3** deployed.

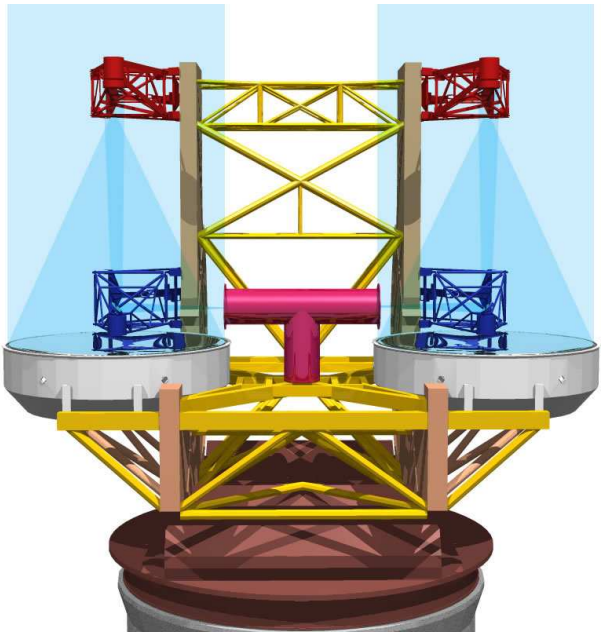

Fig. 1. LBT beams paths to the Bent Gregorian Focal Stations

2.2. *Hardware interlocks*

Due to the size the of observatory, staff and visitor safety is an issue. A part of the telescope can easily be moved from remote location, causing harm to any staff or visitor performing some work on the machine. Movement of those larger parts affecting

[1]Large Binocular Telescope Observatory, University of Arizona, 933 N Cherry Avenue, AZ 85721 Tucson, USA (pkubanek@lbto.org).

VI Workshop on Robotic Autonomous Observatories (Mazagón, Huelva (Spain), September 30 - October 4, 2019)
Editors: A. J. Castro-Tirado, S. B. Pandey, & M. D. Caballero-García - DOI: https://doi.org/10.22201/ia.14052059p.2021.53.14

human safety is protected by hardware interlocks. The commanding switches for interlocks are usually locked out with personal locks, allowing employees to safely disengage some of the telescope functions. Lock state is made available to TCS. Those locks are primary on low, hardware level, well tested with software that doesn't change often - achieving the level of safety needed for telescope operation.

2.3. *Active optics*

LBTO uses active optics to collimate telescope for seeing limited observations. Beam entering an AGW (Acquisition, Guiding and Wavefront sensing) unit located either in the instruments or before instruments attach to the telescope is used for both guiding and wavefront sensing. Calculated Zernikes showing image deformations are shiped to TCS. TCS calculates and executes changes on steerable mirrors (**M1**, **M2** and **M3**, depending on focal station).

2.4. *Sides co-pointing*

Both sides can be commanded to point independently, steering the side optical assembly. TCS has to check if the move is physically possible. Mirror support control allows for slight deviation of side pointing from mount pointing. If those calculations are disabled, excessive forces can be achieved on primary mirror support actuators. Those forces will break the actuators, causing the primary mirror to panic, loose collimation as it goes to rest on a spring coil supports. This in turn will lead to about 20 minutes of downtime, as primary mirror needs to be raised and new collimation performed.

3. TELESCOPE CONTROL SYSTEM (TCS)

TCS controls everything scientists don't care for - in other words, instruments are being controlled by their respective control software, which interact with the TCS. TCS takes care of the building, telescope mount, active optics including wavefront and guiding sensors, and various support subsystems - weather, hydrostatic bearings control and ballast tanks for telescope balancing, to name just a few.

3.1. *Modular design*

TCS consists of modules (or subsystems). Modules are separate programmes, running on TCS server machines. Modules can be started and stopped independently. A custom Remote Procedure Call (RPC) mechanism is provided for the modules to perform calls to another module.

Module functions are usually slitted into multiple threads. The modules runs in soft real-time, not necessary guaranteeing response to hardware events. All safety critical operations are performed on dedicated hardware, which TCS only monitors and commands.

3.2. *TCS Hardware*

TCS is running on servers located in one of the LBTO mountain server rooms. The machines in the so called TCS cluster are connected to each other through TCP/IP network. Hardware located throughout the telescope are connected to TCS servers through a TCP/IP network, using optical fibers to electrically isolate the hardware nodes on various levels of the building and telescope.

TCS servers run the Centos Linux distribution. Data is shared on the TCS servers either through a GlusterFS cloud file system, or as NFS mounts from the mountain NAS.

TCS mainly communicates with subsystems controlling physical movement of the mount and a PLC controlling most of the LBTO building and auxiliary electronics such as building shutters and lights. TCS also connects to DeltaTau UMACs (Universal Motion and Automation Controller) driving various motors on the telescope. Primary mirror collimation is controlled through two real time VxWorks crate and custom electronics in the mirror cell. Guiding and wavefront sensing CCDs are controlled through AZCam server.

3.3. *Reflective memory*

TCS was envisioned to take advantage of reflective memory hardware. Reflective memory allows multiple CPUs to read and write to a common memory block. This approach seems to be quite popular in VxWorks/WindRiver systems, among others. The original intent to use hardware card(s) to be shared among TCS machines was changed to instead use custom software. The software creates IPC shared memory, watches for memory changes recorded through a custom *C++* library, and propagates the changes in 1Kb UDP broadcast packets to the other machines in the TCS cluster. Every machine in the TCS cluster must run a process handling memory synchronization.

Reflective memory is used by TCS modules to write important data to be read by other modules. It's also accessed by GUIs to read data to be displayed.

Reflective memory was used on multiple machines forming the TCS cluster. That includes machines running TCS modules and machines running TCS GUIs. This approach, prone to problems with high traffic with GUIs machines blocking time for TCS machines updates, was reduced by running TCS only on just two servers. Observers and users interact with TCS mostly through GUIs running on TCS servers, which are X11 forwarded to their desktops.

VI Workshop on Robotic Autonomous Observatories (Mazagón, Huelva (Spain), September 30 - October 4, 2019)
Editors: A. J. Castro-Tirado, S. B. Pandey, & M. D. Caballero-García - DOI: https://doi.org/10.22201/ia.14052059p.2021.53.14

3.4. *Telemetry*

TCS produces a plenty of telemetry streams (Summers et al. 2016). Around 7200 data channels are written on different timescales by TCS alone. Streams are recorded at frequencies ranging from 1kHz to a few entries per day. Telemetry data are used to monitor telescope performance and for troubleshooting. Custom web interface written with Polymer WebComponents, using Plot.ly for plotting, is used to display the data. Various other way to access the data are available.

The telemetry was originally stored in MySQL database, but later changed to store in HDF5 data files. One or if necessary multiple files are created per day to keep HDF5 file size below 100 MB. About 10GB of data are produced every 24 hours when all TCS subsystems are running and recording telemetry.

4. TELESCOPE POINTING KERNEL (TPK)

TPK is a *C++* library written for telescope pointing. It provides its users a set of classes encapsulating the original *C* pointing kernel. (Terrett 2006) describes the special binocular telescope interface, which is part of TPK.

5. TCS SUBSYSTEMS

Noticeable TCS subsystems. Due to the dual nature of the telescope, some of the subsystems are doubled, one controlling the left and the other controlling the right side.

5.1. *Mount Control System Processing Unit (MCSPU)*

The MCSPU is the interface which sends PCS trajectories to the mount hardware and returns hardware responses. The MCSPU knows the telescope's position and trajectories in its axis coordinate space - azimuth, altitude and native rotation. MCSPU doesn't know a target Right Ascension, Declination and differential tracking - that's responsibility of PCS. The MCSPU is run on a special computer equipped with DSP I/O cards, responsible for interfaces to the actual mount hardware. Details can be found in (Ashby et al. 2006).

5.2. *Pointing Control System (PCS)*

The PCS handles requests for pointing. It transforms target sky spherical coordinates into mount's altitude and azimuth trajectories and ships them to the MCSPU controlling mount motors. It queries the MCSPU for the current encoder values and transforms those back to the sky position. And it provides function to transform sky position to focal plane position and back. PCS also handles requests for pointing to sidereal or non-sidereal targets.

5.3. *Instrument InterFace (IIF)*

The IIF coordinates communication between instruments and the TCS. An observer can choose a so called authorized instrument, one per telescope side. This instrument is then allowed to call methods which change telescope configuration (for example pointing requests).

Instruments interface with the IIF over ZeroC ICe (Henning et al. 2003). *C++*, *Java* and *Python* language bindings are currently supported. The interface provides a factory and access class. The factory allows for registration and access class creation. The access class is then used to execute calls to the IIF.

Requests for telescope movements can be made either on a single side (monocular), or coordinated for both sides as binocular commands. For binocular commands, both sides must send the command, performed only after data from both sides are received. The TCS checks co-pointing restrictions and reject the command if the restrictions will be violated.

5.4. *Guiding Control System (GCS)*

This subsystem is responsible for guiding and active optics collimation. It is able to take a full frame image with an AGW, find the guiding start, center telescope on the guiding star. After guiding star is centered, wavefront sensing can begin. Beam is fed to Shack-Hartman wavefront sensor, acquired image is analyzed and Zernike coefficients send to PSF for distributions to the telescope optics.

5.5. *Point Spread Function (PSF)*

Zernikes calculated by GCS are transformed into mirror displacement. Telescope operators can choose to displace only subset of the active mirrors, as well to introduce manual corrections, or scale the calculated Zernikes coefficients.

5.6. *Primary Mirror Control (PMC)*

The primary mirror is supported by pneumatic actuators and steered by six hard points, forming a hexapod. PMC acts only to display and command mirror operations. Due to safety requirements, actual mirror control logic is implemented in hard real time VxWorks/WindRiver computer and the control electronics itself.

The system is designed to transition to a so called panic, power less mode on any violations of safety constrains. The constrains are checked both in electronics controlling the mirror and in the VxWorks/WindRiver box commanding it.

VI Workshop on Robotic Autonomous Observatories (Mazagón, Huelva (Spain), September 30 - October 4, 2019)
Editors: A. J. Castro-Tirado, S. B. Pandey, & M. D. Caballero-García - DOI: https://doi.org/10.22201/ia.14052059p.2021.53.14

5.7. *Remaining subystems*

• *LSS* - Logging SubSystem, central point for TCS events, play sounds on critical events.

• *DDS* - Data Directory System, compliment to IIF.

• *ENV* - ENViromental subsystem, collect weather data.

• *ECS* - Enclosure Control System, controls enclosure, handles instrument alarms.

• *MCS* - interface to MCSPU.

• *OSS* - Optical Support Structure, control M2, M3 and swing arms, handles telescope reconfiguration (change of focal stations).

• *AOC* - Adaptive Optics Control, interface with adaptive optics supervisor.

6. TCS FUNCTIONS

6.1. *Environmental monitoring*

Data from multiple weather stations are recorded and displayed. They are used by the telescope operator to decide when to open and close the shutter doors. As the telescope is designed to by supervised by human operators all the time, weather doesn't trigger telescope automatic shutdown (nor opening).

6.2. *Moving telescope to a new target*

Setting a new target to the telescope is essentially a two step process:

1. **Set new target.** The target (together with guiding stars) can be specified using various coordinates.

2. **Preset the telescope.** Target polynomial trajectories are send to **MCSPU**. Optimal steps are performed depending on the preset mode.

TCS governs full telescope presetting, up to mirror collimation and alignment. The following modes, governing what will be performed, are available and can be commanded. Each successor mode performs previous steps plus the step described.

• **Static** points telescope to given location and holds there.

• **Track** open loop track the target.

• **Guide** acquire image with AGW guider, center guide star, start guiding.

• **Active** acquire images from the wavefront sensor and use them to update optics collimation.

• **AO** starts adaptive optics.

6.2.1. *Supported target coordinates*

As TCS supports off-axis guiding (and wavefront sensing), target data usually include both telescope and guiding target. TCS supports the following target coordinates:

• **Sidereal** usually J2000.0 right ascension - declination pair. Star proper motion, magnitude and color class (handy for guiding stars) can be added.

• **Non-sidereal** although J2000.0 pair with rates and start date can be provided, far superior is to use ephemeris file downloaded from the JPL Horizons(Giorgini 2015) service. This allow for far better, up-to-date pointing. PCS keeps care of updating telescope pointing to follow non-sidereal target, synchronizing telescope movements with GCS. Associated TCS packages provide interfaces to download ephemeris from JPL Horizons.

• **Azimuthal** and **Galactic** - those are truly rarely (if ever) used.

6.3. *Offsetting telescope target*

After preset is done, telescope can be commanded to offset. Offset can be one sided (monocular) or both sided (binocular). Binocular offsets are performed only when offset is received on both sides.

During offsetting TCS has to make sure all processed interfering with telescope pointing - guiding, wavefront sensing and AO offloads - are paused.

6.4. *Active Optics management*

TCS is responsible for active optics. That requires coordination among GCS, PSF, PMC, OSS and PCS subsystems.

• **GCS** acquires wavefront sensing images, fit Zernikes and send those to PSF

• **PCS** feeds PSF with elevation data, used for collimation model

• **PSF** coordinates active optics. It allows operators to modify Zernikes coefficients before using them to calculate mirror displacement. It also calculates expected displacements from collimation model, which are used to speed up collimation after presetting a new target

• **PMC** and **OSS** commands and monitors mirror displacement

7. USER INTERFACES

7.1. *Command line interface*

Various command line utilities are available. Those were developed both for TCS testing and operations, allowing operators and engineers to command the subsystems.

7.2. *Engineering NCurses base mount interface*

MCSPU is a separated subsystem, running on separate computer. It provides own interface, through which **MCSPU** function can be controlled. The interface is using nCurses (nCurses 2020) to draw control screens in a terminal.

VI Workshop on Robotic Autonomous Observatories (Mazagón, Huelva (Spain), September 30 - October 4, 2019)
Editors: A. J. Castro-Tirado, S. B. Pandey, & M. D. Caballero-García - DOI: https://doi.org/10.22201/ia.14052059p.2021.53.14

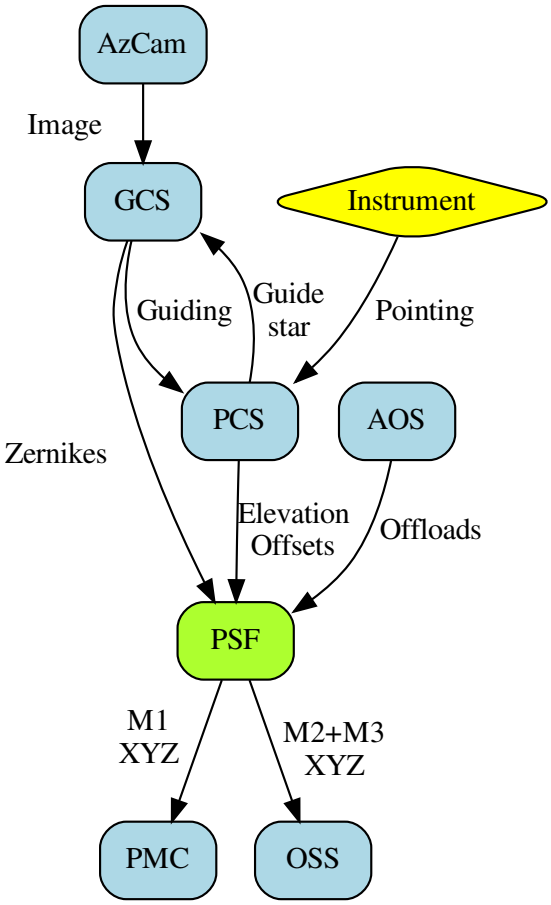

Fig. 2. Active Optics interactions

7.3. *Qt Graphical User Interfaces*

Most of the TCS functions is controlled from Qt GUIs. The GUIs access TCS reflective memory to retrieve current telescope data. TCS RPCs are used to issue commands to various hardware subsystems.

As the instruments are responsible for moving the telescope, TCS GUI lack functions presetting and offsetting the telescope.

8. SOFTWARE PITFALLS

Multiple software engineers and scientists collaborated on the design, coding, testing, deployment and maintenance of the TCS source code. The project progressed through classical pre-deployment, where some time was allocated for system design and architecture, towards classical hot-fixing of the current problems and development of the now needed functionality. As I joined the team at the time when the scientists were to some degree satisfied with the state of the software, I have had the privilege of investigating its various dark corners and improved them in the process.

8.1. *Multi thread programming*

As already mentioned, TCS programmes are using multiple threads to crunch the numbers and handle various communication. Proper locking is then needed to prevent know race conditions either while working with shared resources or calling know thread unsafe (system) calls. C++11 *STL* helps significantly with the task, allowing for use of Resource Acquisition Is Initialization (*RAII*) approach for thread lock management.

Developers using C++ Standard Template Library (*STL*) are usually wrongly assuming the container manipulations are thread safe - when in fact they usually aren't.

Another not so well know problem comes when cancelling a thread [2] occurs while the thread executes in *try..catch all* block. As cancelling a thread essentially throws an exception, the exception must be re-thrown - otherwise the process will be core dumped.

8.2. *Limited understanding of system calls*

System calls are tricky. Developer calling them need to understand properly how they function, interact with the running environments and what happens on possible exceptions. It's hard to debug a process where system call return values aren't tested and system call errors aren't properly reported, particularly if under usual circumstances the code just work.

8.3. *Tendency towards developing everything from scratch*

Particularly early years of TCS developments were plagued by designers and programmers tendency to develop everything from scratch, not relying and integrating existing components. The sentiment can be understood given late 90s landscape, when standard tools were either not existing or available as closed source, binary only packages.

Of course a lot of those decisions backfired - the most prominent example beeing spot (blobs) detection. Various algorithms were developed and tested to detect stars for guiding and wavefront sensing, before SeXtractor (Bertin et al. 1996) was integrated and used.

8.4. *Large code blocks*

Some of the code is poorly split into functions. The code then features a long blocks, making testing, debugging and enhancing the code difficult. Functions and methods with over 1000 code lines aren't uncommon.

[2] using *pthread_cancel* call

VI Workshop on Robotic Autonomous Observatories (Mazagón, Huelva (Spain), September 30 - October 4, 2019)
Editors: A. J. Castro-Tirado, S. B. Pandey, & M. D. Caballero-García - DOI: https://doi.org/10.22201/ia.14052059p.2021.53.14

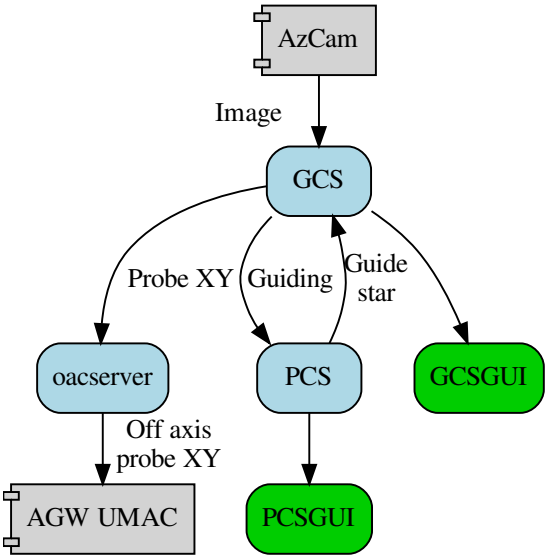

Fig. 3. Current GCS and its dependencies

8.5. *Copy & paste programming*

This is related to the previous problems. Instead of making functions and passing arguments, the developers decided to copy & paste code, and modify some of the parameters. This just makes code longer, harder to read, harder to modify[3].

9. CURRENT AND FUTURE DEVELOPMENTS

Here are presented a few topics I started and are either ready to use or are reaching this point.

9.1. *Code improvements*

C++11 features can be now used in the code. *C++11* allows for in-situ lambda functions, improving code readability. Qt was upgraded to 4.9 release, using system installed libraries (instead of custom Qt installation), and plans are laid for Qt5.

9.2. *Testing*

Testing classes, using Catch2 (Catch2 2020) C++ header only library, are regularly created for new development.

Test scripts, using either IIF or TCS commands, are created for setting up telescope (either simulated or real) into a know trouble position.

9.3. *Generic command line processing class*

CliApp class was created to handle need of various command line programmes. It's a lean, C++11, header only class. The class uses GNU Readline (GNU Readline 2020) for history management.

[3]as if problem is detected in one block, the other similar blocks must be changed as well

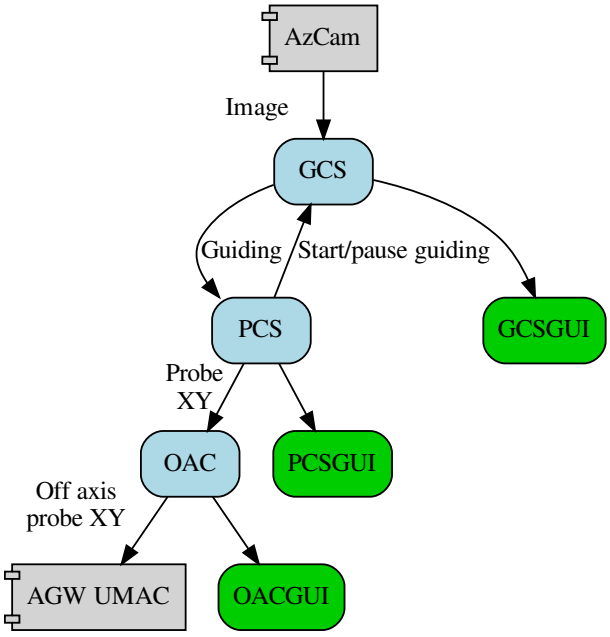

Fig. 4. Proposed GCS interactions

Commands which are provided, together with their allowed arguments and help text, are passed to *CliApp* constructor. Methods associated with commands can then be called either from command line or from an interactive console.

9.4. *New GCS subsystem*

GCS subsystem is most likely the worst of all TCS modules - features really long functions blocks[4] and mix together both simulated and on sky operations.

Current GCS controls AGW stages through a process called *oacserver*. *Sun V* RPCs (Sun Microsystems 1988) are used for commanding the oacserver. That makes direct communication with stages motion control hardware difficult. Adding new features to *oacserver* is difficult due to its legacy C only design.

New architecture replaces *oacserver* with a TCS module called *OAC*. This will communicate directly with motion stages. As the same motion stages are used for active optics control, *OAC* will benefit in usign the proven code.

The following steps are ready to be use:

- class for FITS handling
- class for communication with guiding and wavefront sensing CCDs
- class for communication with AGW stages

[4]2k lines of code aren't uncommon

VI Workshop on Robotic Autonomous Observatories (Mazagón, Huelva (Spain), September 30 - October 4, 2019)
Editors: A. J. Castro-Tirado, S. B. Pandey, & M. D. Caballero-García - DOI: https://doi.org/10.22201/ia.14052059p.2021.53.14

• command line interface for CCDs and AGW stages, verifying design of the former
• simulator (written in Python), simulating CCD control programme
• TCP control and readout for One Line Filesystem interface, used by legacy AGW housekeeping units, mimicking new housekeeping units interface
• SeXtractor (Bertin et al. 1996) - C++ interface class

The next steps are needed to reimplement GCS:

• class for displaying FITS file in QtWidget, together with scaling functions
• guiding algorithm
• wavefront sensing image analysis - where an external script is preferred to C/C++ code logic
• new OAC and GCS modules
• new OACGUI and GCSGUI

10. CONCLUSION

This presents the LBTO TCS. As the observatory transits from an engineering phase, where its various aspects were designed, deployed, (redesigned and redeployed) towards full scientific operations, the results shall contribute to breakthrough discoveries.

REFERENCES

Ashby, D. S., McKenna, D., Brynnel, J. G., et al. 2006, SPIE, 6274, 23
Bertin, E. & Arnouts, A&A117, 393-404
Cath2, available on `https://github.com/catchorg/Catch2`
Giorgini, J. D. 2015, in IAU General Assembly, volume 22, 225293, `https://ssd.jpl.nasa.gov`
GNU readline library, available on `https://tiswww.case.edu/php/chet/readline/rltop.html`
Henning, M. & Spruiell, M. 2003 in ZeroC Inc. Revision
Hill, J. M., Green, R. F., Slagle, J. H., et al. 2008, SPIE, 7012, 03
nCurses, available on `https://www.gnu.org/software/ncurses`
Summers, K. R., Summers, D. M, Biddick, C., & Hooper, S. 2016 in Proc. SPIE 9913, Software and Cyberinfrastructure for Astronomy IV
Sun Microsystems, RFC1050, available on `https://tools.ietf.org/html/rfc1050`
Terrett, D. L. 2006, SPIE, 6274