Asynchronous object-oriented approach to the automation of the 0.8-meter George Mason University campus telescope in Python

Michael Reefe,[a,*] Owen Alfaro,[a] Shawn Foster,[b] Peter Plavchan,[a,b] Nick Pepin,[a] Vedhas Banaji,[c] Monica Vidaurri,[d,e] Scott Webster,[a] Shreyas Banaji,[a,f] John Berberian,[a,g] Michael Bowen,[a] Sudhish Chimaladinne,[a,h] Kevin Collins,[a] Deven Combs,[a] Kevin Eastridge,[a] Taylor Ellingsen,[a] Mohammed El Mufti,[a] Ian Helm,[a] Mary Jimenez,[a] Kingsley Kim,[a,h] Natasha Latouf,[a] Patrick Newman,[a] Caitlin Stibbards,[a] David Vermilion,[a,e] and Justin Wittrock[a]

[a]George Mason University, Department of Physics and Astronomy, Fairfax, Virginia, United States
[b]Rochester Institute of Technology, Department of Physics and Astronomy, Rochester, New York, United States
[c]Columbia University, New York, United States
[d]Howard University Department of Physics and Astronomy, Washington, DC, United States
[e]NASA Goddard Space Flight Center, Greenbelt, Maryland, United States
[f]Flint Hill School, Oakton, Virginia, United States
[g]Woodson High School, Fairfax, Virginia, United States
[h]Thomas Jefferson High School, for Science and Technology, Alexandria, Virginia, United States

Abstract. We present a unique implementation of Python coding in an asynchronous object-oriented programming (OOP) framework to fully automate the process of collecting data with the George Mason University (GMU) Observatory's 0.8-meter telescope. The goal of this project is to perform automated follow-up observations for the Transiting Exoplanet Survey Satellite (TESS) mission, while still allowing for human control, monitoring, and adjustments. Prior to our implementation, the facility was computer-controlled by a human observer through a combination of webcams, TheSkyX, Astronomy Common Object Model Dome, MaxIm DL, and a weather station. We automate slews and dome movements, charge-coupled device exposures, saving flexible image transfer system images and metadata, initial focusing, guiding on the target, using the ambient temperature to adjust the focus as the telescope cools through the night, taking calibration images (darks and flats), and monitoring local weather data. The automated weather monitor periodically checks various weather data from multiple sources to automate the decision to close the observatory during adverse conditions. We organize the OOP code structure so that each hardware device or important higher-level process is categorized as its own object class or "module" with associated attributes and methods, with inherited common methods across modules for code reusability. To allow actions to be performed simultaneously across different modules, we implement a multithreaded approach in which each module is given its own CPU thread on which to operate concurrently with all other threads. After the initial few modules (camera, telescope, dome, and data I/O) were developed, further development of the code was carried out in tandem with testing on sky on clear nights. We achieve our goal of a fully automated nightly observation process. The code, in its current state, was tested and used for observations on 171 nights, with more planned usage and feature additions in the future. [DOI: 10.1117/1.JATIS.8.2.027002]

Keywords: exoplanets; astrophotometry; automation; optical telescopes; Python; Transiting Exoplanet Survey Satellite.

Paper 21133G received Oct. 22, 2021; accepted for publication Jun. 3, 2022; published online Jun. 28, 2022.

*Address all correspondence to Michael Reefe, mreefe@gmu.edu

1 Introduction

The transit method became the most dominant method for discovering new exoplanets with the launch of Kepler in 2009[1] and, more recently, the Transiting Exoplanet Survey Satellite (TESS) mission in 2018.[2] By observing the dips in star brightness as an exoplanet passes, or "transits," in front of its host star, we can acquire knowledge about exoplanet properties such as the radius, period, orbital eccentricity, and atmospheric transmission properties, as well as in some cases additional planets in the same system.[3] Although TESS has found over 4000 exoplanet candidates during its first 2 years of scientific operation, monitoring ~500,000 relatively nearby and bright stars, follow-up observations remain a key priority for validation or rejection of these candidates. These observations have been able to identify everything from hot Saturns[4] to warm Jupiters,[5] other Jovians,[6] and even Earth-like planets in the habitable zone,[7] and they may be coordinated using frameworks such as NASA's Exoplanet Archive.[8]

Follow-up observations of transiting exoplanet candidates adhere to a regimented set of procedures. This structure has arisen to efficiently maximize the scientific return from observatory resources, particularly the limited time on larger 3 to 10-meter class telescopes, to obtain precise radial velocities and exoplanet mass measurements with high-resolution spectrographs. We screen candidate exoplanets with photometric imaging on smaller meter-class telescopes to confirm that transits occur when they are expected, with a duration and change in brightness (transit depth) consistent with the TESS observations, and that these transits are associated with the expected star. This allows us to identify false positives, which can be caused by dips in stellar brightness from a variety of eclipsing binary scenarios that are not transiting exoplanets, including blended eclipsing binaries, nearby eclipsing binaries, grazing eclipsing binaries, and other systematics (e.g., the kilodegree extremely little telescope follow-up network (KELT-FUN)[9] and analysis of false positive probabilities in Kepler[10]). In particular for the NASA TESS mission, the pixel scale on the sky of the on-board cameras is 22 arcseconds per pixel, with light from each star spread over several pixels (defining the point spread function or PSF).[2] This was an intentional design choice to maximize the total sky coverage of the mission's four science cameras (each with over 6 times the areal sky coverage of Kepler), but this also meant that many candidate host stars were observed in groups of pixels that contained many relatively fainter background stars in addition to the relatively brighter target stars, resulting in a fraction of TESS candidates that are false positives.[11,12] Only follow-up observations with finer angular resolutions (pixel scales on the order of 1 arcsecond per pixel or less) could discern whether the dips in brightness observed by TESS are associated with the individual targeted stars, with the expected chromaticity and ephemerides, or nearby fainter false-positive stars that fell within the same TESS pixels.

The need for photometric follow-up monitoring of TESS mission candidates drives a need for automation, given the volume of data collection required. Many software tools already exist for the automation of telescope and observatory control, both for ground-based follow-up observations and for the exoplanet searches themselves, including wide-field transit survey searches and targeted spectroscopic follow-up monitoring. A noncomprehensive list of past ground-based observatory automation efforts is listed in Table 1, which includes examples of wide-field photometric exoplanet transit searches on submeter class telescopes, follow-up observational facilities with moderately sized telescopes comparable to the GMU telescope, and fully automated large-scale (>1 m) or multitelescope array facilities performing both photometric and spectroscopic follow-up observations.

Drawing upon this prior automation work for inspiration, and in particular the MINERVA-North Python-based framework (https://github.com/MinervaCollaboration), we pursued full automation of exoplanet follow-up photometric observations with the 0.8-m Ritchey–Chrétien telescope at George Mason University (GMU). Apart from the scientific goals, our motivations for undertaking this project are primarily grounded in improving the observational efficiency and data quality from the GMU telescope. With the variability of weather conditions in Fairfax, Virginia, United States, it can be challenging for human observers to make optimal use of a meter-class telescope, particularly on nights that are intermittently clear. Additionally, meter-class telescopes are not as well staffed or resourced as larger telescope facilities. The GMU observatory in particular is located and operated by an educational institution,

Table 1 A comparison of past automated observatories that have been semiautomated or fully automated. The scientific goals of the observatory are enumerated in the second column, and the number and size of telescopes at each facility is in the third column. References to each project are included in the first column.

Telescope/facility	Type	Number × size
MEarth[13]	Transit survey	–
KELT[14]	Transit survey	1 < 0.5 m
HATNet[15]	Transit survey	6 × 0.18 m
TrES[16]	Transit survey	3 < 0.5 m
WASP[17]	Transit survey	1 < 0.5 m
DEMONEX[18]	Follow-up transits and TTVs	1 × 0.5 m
Robo-AO[19,20]	Follow-up AO	1 × 2.2 m
OWL@OUKA[21]	Follow-up transits	1 × 0.6 m
MINERVA-North[22]	Follow-up transits and spectroscopy	4 × 0.7 m
MINERVA-Australis[23]	Follow-up transits and spectroscopy	4 × 0.7 m

where observations are primarily carried out by undergraduate and graduate students, and telescope time must be allocated for classes and tours, which take precedence over research observing. However, observing during late nights, weekends, holidays, and/or academic breaks may not be in the best interests of the students. By implementing an automated observing routine, we not only minimize these challenges but also create a uniform organizational system for data folder hierarchies and naming conventions and increase the volume of data available for students to reduce and gain experience in analyzing.

We present herein an asynchronous object-oriented approach to achieve the goals outlined above in Python, extending the work of past automated observatories, without compromising the educational usage of the telescope. We develop our own custom Python software optimized for both our particular Observatory hardware setup and science observation goals, which primarily consist of repeated, staring photometric observations for most of a night. We opt for an asynchronous approach to further increase the operational efficiency by enabling actions to be performed in parallel when appropriate. For example, this maximizes the safety of automated observations by allowing weather conditions to be constantly monitored in the background and to close up the observatory during inclement weather, even if this occurs during a long exposure sequence. Python was chosen due to its existing libraries and ease of use. The slower run times associated with interpreted languages do not significantly increase overhead during our observations because our main computational bottleneck is often hardware response times, and we utilize the compiled numpy[24] and numba[25] packages for more computationally intensive tasks.

In Sec. 2, we briefly overview the hardware setup and specifications of the GMU Observatory and how the facility has previously been manually controlled. In Sec. 3, we present a description of each module of the software and its development and testing process, starting with a general overview of the code structure, then explaining each module in more detail, and ending with a description of our testing process. In Sec. 4, we present an analysis of the performance of our base and higher-level automated systems, notably the automated focusing and guiding routines, and show examples of reduced data gathered using the software. In Sec. 5, we present a brief discussion and our conclusions, including planned future work on the project.

2 Hardware

GMU's Ritchey–Chrétien telescope (RC32-1900F; Fig. 1) is mounted on an OGS-1900 equatorial fork created by Optical Guidance Systems and located on the roof of GMU's Research Hall, at an elevation of ∼154 m and within a 25-foot diameter Ash Dome. The mount has Renishaw hour angle and declination motor encoders and astronomy common object model

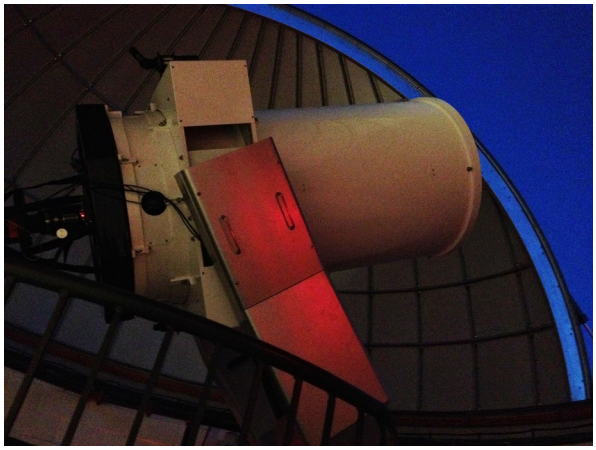

Fig. 1 The GMU Ritchey–Chrétien telescope pointing on a target through the dome shutter, near twilight. This equatorially mounted telescope has a charge-coupled device (CCD) camera and eyepiece connected at the base right below the 0.8-m primary mirror. The tertiary mirror has a maximum of four positions that it can rotate through to attach different devices to the telescope.

(ASCOM)-compliant drivers, with a Software Bisque controller operated through TheSkyX. Due to an unknown cause, the telescope occasionally (about once per night) experiences "jumps" in RA of a few arcminutes, which our RA absolute motor encoder then corrects with a small time-lag on the order of one minute. We also occasionally experience oscillations in the declination axis, which we believe is exacerbated by an imbalance of the telescope and potentially building vibrations.

The telescope's primary mirror is 32" in diameter, with a focal ratio of $f/7.2$ and a focal length of 5760 mm, and its secondary mirror is 12" in diameter. Installed on the secondary mirror is an optical guidance system robotic focuser that allows its distance from the primary mirror to be finely tuned and adjusted to micrometer-level precision. The relative stepper motor has no physical "home" position, so we must be careful to avoid hitting the mechanism's hardware limit switches at either end, which requires a physical hardware reset using a banana connector. These limit switches were put in place in 2018 to prevent the secondary mirror from running off the end of its range, which it previously ran the risk of doing, requiring significant manual repair. In the past, students avoided manually focusing the telescope for fear of moving it too far—a problem that was eliminated by adding the limit switches. The focuser can be controlled manually through the RoboFocus software interface by Technical Innovations or through MaxIm DL. The 45 deg Perseus tertiary mirror, just under the primary, can be rotated to redirect the telescope light to any one of four instrument ports, including an eyepiece and CCD camera. The location of the mirror can be toggled via the Perseus control software.

Our CCD camera is a high-quantum efficiency (60%) Santa Barbara Instrument Group (SBIG) STX-16803 with dimensions of 4096×4096 16-bit pixels and a plate scale of 0.34"/pixel, leading to an FOV of $23.2' \times 23.2'$. It has a nominal read noise of $9e^-$ and dark signal of $3e^-$ px^{-1} s^{-1} when cooled. It saturates at $\gtrsim$40,000 counts. The CCD cooler is a two-stage thermoelectric system, with a variable fan, that can reach temperature deltas of up to 50°C. The camera is mounted on the telescope so that the image x/y axes are approximately aligned with the negative RA and Dec axes along the sky, respectively, to within a couple of degrees rotation. A filter wheel is also installed on the CCD camera; this allows us to use a set of Johnson-Cousins filters: clear, U, B, V, R, I, and $H\alpha$. The camera and filter wheel are both manually controlled via MaxIm DL. Within the dome is an incandescent light bulb, used as a flat lamp for taking dome flats, with power connected to an Arduino control switch accessible by the observatory control computer. There is also a dehumidifier that is used to keep the inside of the dome dry. Outside the dome is an all-sky camera used for remotely determining weather conditions and cloud coverage, and on the roof of Research Hall, there is also a local weather station that collects data on precipitation, wind, humidity, pressure, etc. and updates a public server hosted in the observatory control room. A collection of webcams is also installed to

provide live feeds of inside and outside the dome, as well as inside the control room. We particularly note that we do not currently have a second CCD camera, which would be useful for parallel guiding and/or focusing operations. As such, all focusing and guiding operations performed by the code must utilize data from our SBIG STX-16,803 only.

3 Development and Testing

3.1 *Code Structure and Threading Overview*

We structured the automation software for our campus telescope on the principles of modular object-oriented programming and multithreading, to run in Python version 3.8 on a Windows 10 operating system. Windows 10 was chosen primarily for its legacy support, to avoid the future obsolescence of the operating system, and for its compatibility with common commercial telescope hardware drivers and software programs. This has also facilitated more accessibility and ease of use for students, who may be unfamiliar with a Unix-based system. We avoid challenges with Windows' automatic updates interrupting observing and software functionality by disabling them. For long-term memory and swap space management, we created a scheduled task to reboot the computer every Sunday morning. We also have the control system mirrored on an independent computer, which, prior to the pandemic, we used to apply and test operating system updates before installing them on the main system. We also have another scheduled task to backup and sync all observatory data daily to a remote computing cluster through ssh (via Git for Windows[26] and MinGW), to prevent data loss from OS or hard drive failures.

A flowchart of the modular organization and sequencing of the project is depicted in Fig. 2. The structure of the project can be organized into five broad categories. The first is the modules that are assigned independent threads that asynchronously control individual "base" hardware components or higher-level processes that rely on multiple hardware components. The second is the core system that processes input from the command line and configuration parameters and executes observing sequences that we refer to as "observation tickets." The third is the inputs and configuration parameters themselves, which we standardize on json formatted input files; the inputs include specific information required for the code to perform an observation on a specific target and the command line interface (CLI) input parameters. The fourth category is controls for

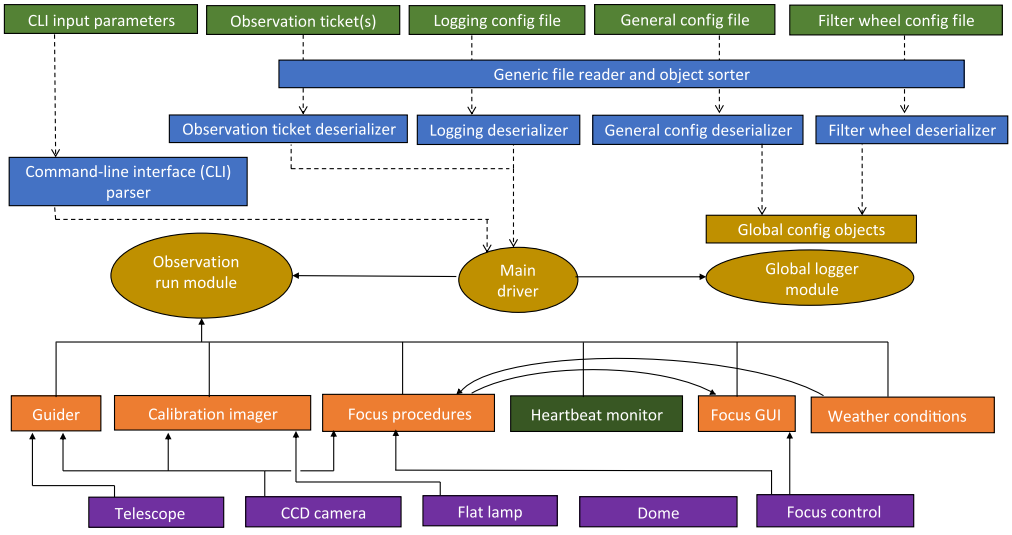

Fig. 2 A flowchart of the automation software depicting the flow of data and structure of the code. Dashed arrows indicate data flow through file and software object reading; solid arrows indicate the structure of operational inputs and dependencies. The heartbeat monitor depends on all other threads to run. Block colors indicate different categories of objects: purple objects are direct hardware control interfaces and threads, orange objects are higher-level processing threads that rely on multiple hardware components, light green objects are inputs, blue objects are file readers and data parsers, and caramel-colored objects are core system processes.

Table 2 A list of all 12 threads that are initiated to run simultaneously during an observing sequence and the purpose of each thread. For hardware objects, the type of connection interface is notated in parenthesis. The "type" column shows the type of structure that the thread's run method follows – a queue system, constant monitoring, or GUI.

Thread name	Purpose	Type
Main	Main observing sequence and thread dispatcher	Main
Camera	CCD camera hardware control (MaxIm DL API)	Queue
Telescope	Telescope mount hardware control (ASCOM API)	Queue
Dome	Dome hardware control (ASCOM API)	Queue
Flat lamp	Flat lamp power control (serial)	Queue
Focus control	Focuser hardware control (serial)	Queue
Focus procedures	Focus routines and intelligence	Queue
Focus GUI	GUI for manual focus override	GUI
Guider	Active guiding routines	Queue
Calibration imager	Calibration imaging routines	Queue
Weather monitor	Continuous background weather checks	Monitor
Heartbeat monitor	Continuous background operation checks	Monitor

parsing and processing the configurations and inputs into software objects. Finally, the fifth (not depicted in Fig. 2) is the general utilities used throughout the project such as astronomy-specific tools for time, coordinate conversion, and image handling. In Table 2, we present an overview of all 12 threads that run simultaneously during nominal operations for an observing sequence. Apart from the main thread, there are five threads dedicated specifically to hardware control, three for higher-level processing routines, two monitors, and one graphic user interface (GUI).

The core structure of our asynchronous design utilizes Python's built-in threading module. Each of the threads listed in Table 2 (except the main thread) is composed of a software object subclassed from the Thread class and dispatched via the main thread. We override the __init__ and run methods to fit each particular thread. For the hardware control, focusing, guiding, and calibration threads, we construct the run method (which starts the thread as a separate instance from the main thread) to be a queue system that is continuously waiting for commands to be requested, checking once per second. Other threads can then place commands on that thread's queue with a separate onThread method, which allows actions to be requested in parallel. In the unique cases of the weather and heartbeat monitors, we instead construct the run methods to be constantly monitoring for the appropriate conditions, without a queue system. The focus GUI also does not have a queue system.

We handle cross-thread communication primarily with Event objects, which can be set or cleared and which other threads can check directly or wait until the event is set with the wait method (which is used mainly to wait for hardware responses, i.e., slewing the telescope). Lock objects are also sometimes used to prevent multiple threads from accessing the same resources at once. The main usage of locks is in relation to plotting with matplotlib because its backend does not support multithreaded plotting. For actual computations (i.e., for the FWHM while focusing, displacement of stars for guiding, converting coordinates to Alt/Az), we avoid running the computations themselves on more than one thread to not encounter race conditions.

3.2 *Initiation*

A typical observation sequence for a TESS follow-up candidate (target of interest or TOI) will last for an entire night (from sunset to sunrise), even if the transit is only for a short portion of that time, to collect as many pre- and post-transit baseline measurements as possible. Exposures are typically done in the R or V filters because they are the most efficient in our typical observing

conditions, and exposure times range from 10 to 120 s depending on the magnitude of the target. The exposure times are determined to maximize the signal-to-noise ratio (SNR) of the target without going over our CCD's saturation or linearity limits. We require a target SNR > 100 because the transit of a Jovian-sized planet orbiting a Sun-like star produces a brightness change of $\sim$1%, and our typical photometric precision systematic noise floor is $\sim$0.1%; anything less than an SNR of 100 will not have sufficient photometric precision to recover the detection of the TOI. Transits also require a reasonable cadence between images to properly resolve a transit in the time domain, so we typically do not go over 120 s even if this is not enough to reach an SNR of >100 because we can make use of temporal binning if the transit duration is sufficiently long to achieve the required sensitivity. Next, if a TESS TOI observation sequence requires the use of multiple filters to check for chromaticity in the observed transit depth, we can utilize one of our other *U*, *B*, or *I* filters. CCD readout times are $\sim$5 s. Additionally, there are typically $\sim$5 min of overhead at the beginning/end of observing (or during temporary shutdowns due to weather) to slew the telescope/dome, open the shutter, and wait for the CCD cooler to settle to -30°C—all actions that occur simultaneously on separate threads.

We create observation tickets with a Python GUI (which runs separately from the main observing software) that generates json files for a target on a given night. These files can be created manually, but the GUI streamlines the process for those who are not familiar with json file formatting. The input specifications that are required for the code to run are the target name (for image saving and FITS header purposes), RA and Dec (J2000), observation start and end times, filter(s), exposure time(s), and number of exposures. The number of exposures is usually specified as an arbitrarily large number to allow observations to simply continue until the end time has passed. We effectively set aside blocks of time for each target to be observed on a given night; however, the option is still present to take a specific number of exposures if this is necessary. There are also options to enable or disable guiding or cycling through filters with each image. Using the urllib python library, information for target scheduling with content is retrieved from a Google sheet maintained by humans (name, filter, exposure time, start time, and end time); the GUI automatically fetches the necessary target information for the night and automatically gathers RA and Dec from the ExoFOP database.[27] There are basic checks in place to ensure that the input quantities make sense (i.e., exposure time >0 s, start time $<$ end time, number of images >0), but a user may accidentally input the wrong observing time or coordinates although they are still valid. In such a case, there are additional checks in place to make sure the RA/Dec are within physical slew limits (above 15-deg altitude and within ± 8.5 h HA), and observations will not begin during the day when the sun is up. The HA slew limit was implemented because our RA motor encoder tape has a limited length, and slewing further than ± 8.5 h carries a risk of it running out and delaminating the edges. If two targets have overlapping observation times, the one with the earlier start time always takes precedence, and the other objects are not be considered until the end time of the first one passes. We also currently do not check for idle time during a night, so users must be careful that observation time slots fill up the entire night when creating tickets—the observing queue is rigidly set once the program has been started.

The role of a human observer in this process is to create an observation ticket (or set of tickets) using the GUI and subsequently initiate the code, with the ticket(s)' file paths as arguments, using the CLI created with the argparse Python module. This is most often done through the Git for Windows bash terminal due to its emulation of familiar bash commands from Unix-based systems. The main package is installable using pip in the main code directory, and after installation, the CLI can be run from any directory. All of the dependencies are listed in the requirements.txt file and are installed automatically with pip. The CLI has additional options that may be passed in while starting the code. Among these are options to disable the automated focusing, disable automation calibration imaging, or disable the automated shutdown sequence at the end of observing. One may also pass in custom filepaths for configuration files rather than using the default paths.

The CLI arguments are parsed by the argument parser, which then passes the observation instructions to the main driver. The main driver then reads the observation ticket json file(s) and the configuration json files for general configurations, the filter wheel, and the logging module. These configurations are then converted to software objects by the file readers and deserializers. The main driver then instantiates the observation run module to begin the observation routine and

the logging module for status messages and debugging. The observation run module itself instantiates all of the base and higher-level hardware objects and starts their respective threads. Most higher-level hardware structures require at least one of these objects as input to function. For example, the guider module requires the CCD camera to read and wait for images and the telescope module to send pulse guide commands. An arbitrary number of observation tickets may be passed into the code simultaneously, either by passing in each file path separately or by providing a path to a parent folder that houses all of the requested observation tickets. This potentially allows for the queueing of weeks or months of observations in advance, although this would require the scheduling of these observations to be performed in advance as well. From here, the observation routine begins. In Fig. 3, we present a flowchart detailing

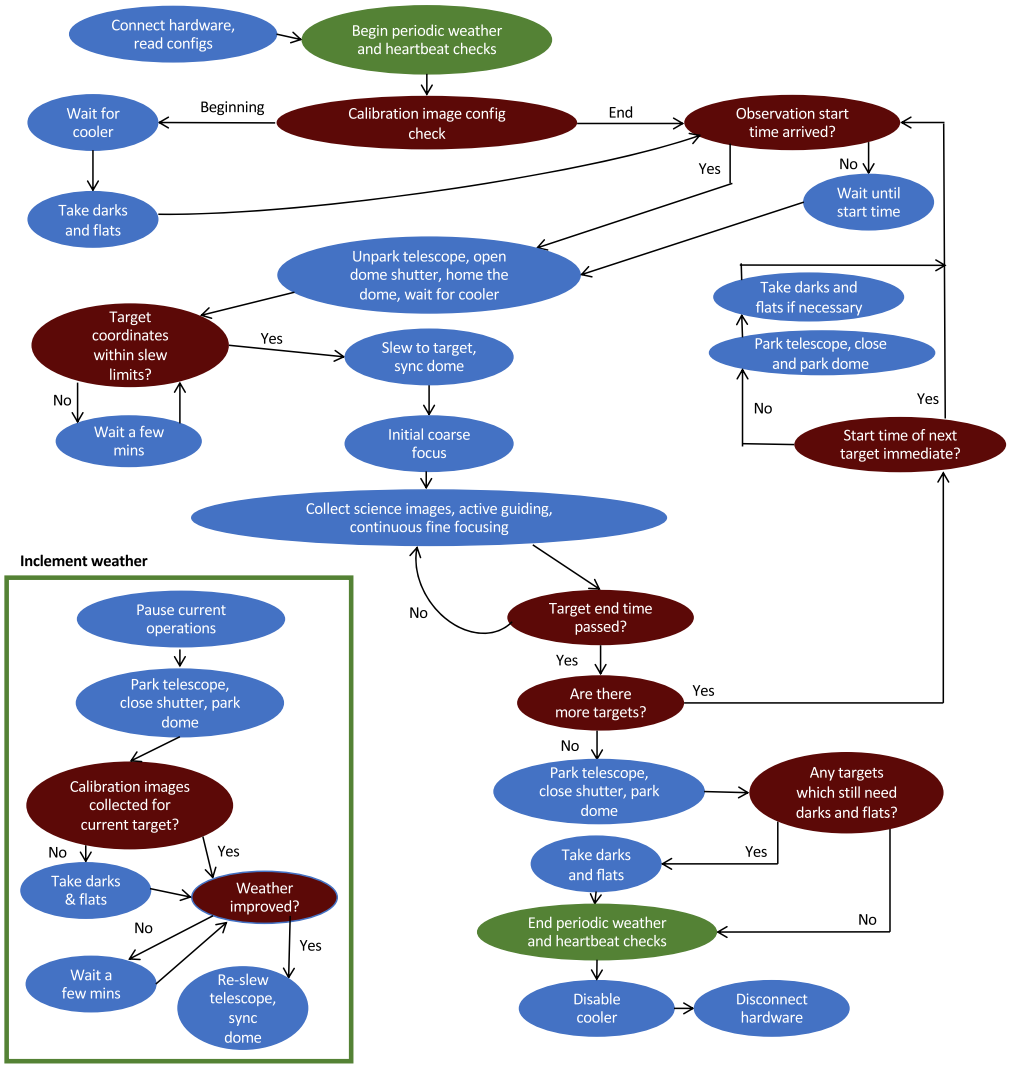

Fig. 3 A flowchart of the main observing sequence performed by the code. Decisions are marked in dark red, and actions are marked in blue. All actions listed within the same bubble are performed in parallel. Green signifies the periodic weather and heartbeat checks that occur constantly in the background. Once the weather checks are initiated at the start of the night, the sequence may, at any point, jump into the box labeled "inclement weather" if conditions deteriorate. If conditions subsequently improve, then the sequence may leave the box and resume its previous operations, continuing to observe the target within the current designated ticket or moving on to the next target if the time window has passed while in the "inclement weather" box. Otherwise, if conditions remain poor for the remainder of the allocated time for all targets, the sequence will jump to the end, disabling the cooler, disconnecting from all hardware, and terminating. Note that we consider sunrise to be "inclement weather," so an observing sequence that has an end time too late (or a start time too early) may jump to the "inclement weather" box during the daytime.

the main observing loop, highlighting important actions and decision points. If possible, the human observer stays until the initial focusing routine finishes (see Sec. 3.5), so they can manually adjust the focus if it fails using the GUI, but this is often not necessary.

Importantly, the main thread also gathers FITS header information for the CCD exposures as each exposure completes. The MaxIm DL dispatch object itself automatically handles basic header information that is directly related to the image and CCD, such as exposure time, filter, binning, frame type, image pixel dimensions, CCD temperature, etc. The additional information that we acquire for each image includes the observation site latitude, longitude, and altitude; the Julian date (JD) at the start of observations; the JD and Barycentric dynamical time (BJD_{TDB}) at the exposure midpoint; the RA and Dec of the target object both in J2000 and apparent equinox coordinates; and the altitude, azimuth, zenith distance, hour angle, and airmass of the target object, also at the exposure midpoint. These quantities are all gathered through our global utility function modules, which perform the coordinate and time conversions in real time. The BJD_{TDB} calculation relies on an astroquery request to the SIMBAD database for information on the target's proper motion, parallax, and radial velocity. The software first attempts a simple search with the target name; however, this often fails, as many TESS TOIs are not registered as aliases in SIMBAD. Thus, if this initial search fails, we use the RA and Dec coordinates to probe a radius of 5′ around the target coordinates and choose the closest object found. If for any reason the necessary quantities still cannot be recovered, the BJD_{TDB} calculation simply returns a null value and must be computed later in data reduction.

At this point, the code continues running until all observation tickets are completed, even if this takes multiple days, as the code closes up and waits during the daytime. After the final observation ticket is completed, the final shutdown sequence initiates, and all threads, hardware, and software connections are terminated. In the next subsections, we go over each module of the project in more detail.

3.3 *Base Hardware Control Modules*

The base hardware of our automation structure, upon which higher-level structures such as focusing and guiding are built, includes the CCD camera, telescope, dome, focuser, and flat-field lamp. Each hardware device is mirrored in the project with its own Python class. Each hardware device class has its own attributes and methods, as well as its own CPU thread to allow for actions to be performed concurrently and asynchronously across hardware devices. We chose to have all of our hardware devices inherit from a generic base Hardware class that implements the basic structure of our design, to reduce repetition within the code, and to make it less cumbersome to add additional hardware control modules in the future. This means that actions common to all hardware devices, such as handling asynchronous multithreading and communication between classes as described in Sec. 3.1, are a part of the base Hardware class (which itself is a child of thread). On the other hand, actions for specific hardware are left to the children Camera, Telescope, Dome, Focuser, and FlatLamp classes to implement. As an example of a hardware-specific method, our camera class has the expose method that takes a camera exposure with the specifications that are given for the method, in this case the exposure time, filter, save path, and frame type for the shutter state (light for open shutter or dark for closed shutter). Hardware communication is handled in a device-dependent manner. The telescope and dome are controlled via client-side COM port dispatches to their ASCOM device application programming interfaces (APIs). Our CCD camera is dispatched via the MaxIm DL API. This was done because our camera model, the SBIG STX-16803, does not directly support an ASCOM device API, and we adopted the similar approach taken in early versions of the MINERVA-North camera control,[22] which has since transitioned to implementing its own ASCOM-compliant server for its cameras. The cooler and filter wheel are also controlled through the MaxIm DL camera API. At the start of observations, the code nominally attempts to reach a cooler set-point of −30°C, but with our air cooling system, this may not always be possible, especially on hot summer nights. After the cooling rate of the detector slows down to <3°C/min without reaching the nominal set-point of −30°C, the software begins iteratively adjusting the temperature set-point every minute. The amount that the set-point is adjusted is based on the difference between the set-point and the actual temperature of the detector,

with the adjustments being ~half the difference. This is repeated until the set-point and actual temperature match within 0.2 deg. This routine is performed once at the beginning of the night and may be repeated if the software runs for multiple days. During the day when the Sun is up or when the software terminates, the cooler set-point is returned to an idle value of +5°C to minimize strain on the system.

The automated dark and flat-field images necessitated a method for remotely switching the flat lamp on and off, so we created a flat lamp module using a direct COM port serial connection to our Arduino device with the PySerial library; the hardware itself is controlled via an Arduino that controls a power supply switch to the lamp. We then implemented a similar serial structure for the focuser control module, which controls the z-axis position of the telescope's secondary mirror, to avoid using the native but unstable RoboFocus[28] API, which we found would frequently crash unexpectedly on Windows 10.

3.4 *Data I/O and Logging*

In addition to the base hardware control modules, the essential modules of the code include data input/output (I/O), logging, and utilities. The inputs are handled in terms of configuration files in the json format, command-line arguments, and observation "tickets" (also jsons) with instructions for observing a specific target. The I/O module handles reading the observation ticket and configuration specifications from json files and converting them into python objects. The logging module keeps track of status messages and displays them on the command-line while also saving them to a log file. We implement a rotating file handler system for the log files, in which the log files have a maximum size of 10 MB before a new one is created. Recent log files are kept saved but are deleted once they become older than the previous 10. Log files typically build up at a rate of 4 to 5 MB per night of observing, meaning we typically keep data from the past ~20 to 25 nights. Because we are currently active in responding to software bugs, we find that this provides a good balance between hard drive space concerns and debugging. However, we may find in the future that we wish to preserve log data for longer periods. In this case, all data storage and formatting options for the logging files are stored in the logging configuration json file and may be readily adjusted. Because the logging output typically depends on a slower cadence while waiting for hardware responses (i.e., long CCD exposures, weather updates), the concern of infinitely recursive errors building up rapidly and filling up all of the logging space is minimized. The higher-level structures of the code, which we discuss next, rely on the interplay of the base hardware components, core system functions, and modules.

3.5 *Focusing*

The goal of our focusing module is to achieve and maintain an optimal focus relatively quickly, and we take a two-fold approach to focusing the telescope. Inherently variable atmospheric conditions sometimes rapidly affect focus quality, and our software and hardware impose additional constraints on our measurements. Additionally, we must be cautious to not allow the stepper motor to hit the physical limit switches at either end of its range, requiring a manual reset, as discussed in Sec. 2.

Consequently, to achieve a practical and mostly reliable automation of the telescope focus, we first codify the assumption that the secondary mirror position is close to the optimal focus position when the software is started at the beginning of an observing sequence; if the focus is significantly in error, and stars cannot be detected in the image, then a manual focus is required. Many autofocusing routines have been developed for the purposes of automated observations, including binary and fibonacci searches.[29] However, many of these routines attempt to optimize searching a wide range of possible focus positions, and thus they take many images to find a best fit focus. We instead choose a simple grid search technique in combination with a parabolic fit, which limits the number of test exposures that are required in exchange for a smaller search area. We choose to focus on the science target's field of view to eliminate telescope flexure differences that could otherwise introduce an offset. This also eliminates additional slew times and the need to readjust pointing between targets and focus stars, as differences in detector pixel sensitivities can introduce light curve systematics. During the initial telescope focus at the beginning of

an observing sequence, a single exposure is taken at the current focus position. The locations and relative brightnesses of stars in the initial image are identified with the find_peaks function in the photutils Python package with a center-of-mass centroiding algorithm. For each star that is found, we calculate an SNR

$$\mathrm{SNR} \approx \frac{N - S}{\sqrt{N + S}}, \tag{1}$$

where N is the target counts and S is the sky background counts. This ignores readout and instrumental noise from the CCD, but for our purposes, it is a good enough approximation. We then calculate the full-width at half-maximum (FWHM) of each star's Gaussian PSF for the image and combine them using a weighted average of each star's FWHM value, with the SNRs used as weights.

We then take a series of additional images at positions from -50 to $+50$ steps from the initial position, in increments of 10 steps. Focus image exposure times are set to one-third the duration of the science exposures, which we find still typically have high enough SNRs to perform adequate focus calculations, but this is an adjustable parameter. We do not correct these images for noise, as typically the appropriate dark and flat images are not collected until the end of an observation, and the improvement in focus from these corrections would be marginal. During this focusing procedure, the code automatically saves diagnostic plots of the focus apertures used on each image, and the radial profiles of the four highest SNR-stars in each image. An example of each of these plots from the night of July 7, 2021 UT is shown in Figs. 4 and 5. The stars in this image show FWHMs of ∼15 pixels, or 5.1", which is to be expected—during the focusing sequence, testing out-of-focus positions can produce FWHMs upwards of 20 pixels. After the focus procedure is complete, we can get as low as $\lesssim$10 pixels or 3.4". The background level of ∼1500 counts is also typical, due to a combination of our CCD's bias voltage (corresponding to ∼1100 to 1200 counts) and light pollution in Fairfax. If no stars can be located in an image, the software attempts to retake the image. If stars cannot be located in three consecutive images, the focusing routine is terminated.

We expect from diffraction-limited optics that the FWHM of images should increase linearly in both directions (creating a "V" shape) from the optimum focus (minimum FHWM). However, for telescopes > 0.2 m, atmospheric turbulence, blurring, and optical imperfections cause the

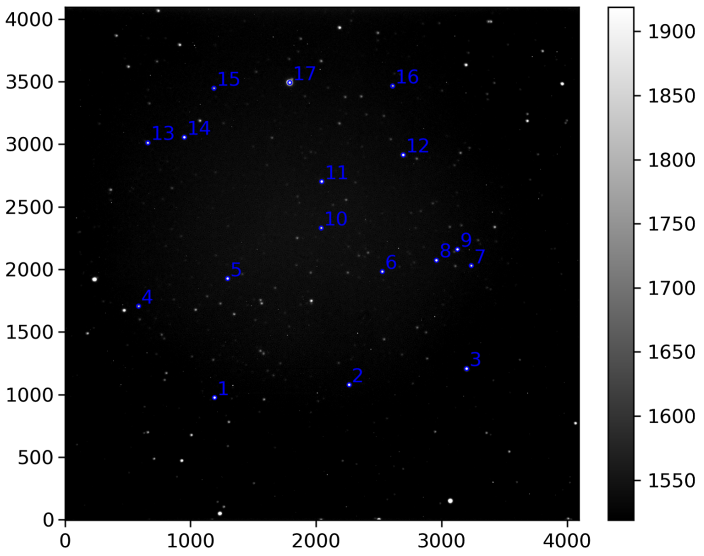

Fig. 4 A plot of the focus apertures in blue used by the code superimposed over the CCD image on which they are being used. The *x* and *y* pixel numbers are displayed in the axes, and a color bar shows the grayscale intensity stretch of the counts from the CCD image. Stars near the edges of the image, as well as any stars that are not bright enough compared with the background, are ignored as focus star candidates. Some stars that are visible were not chosen by the find_peaks algorithm because their peaks were too dim compared with the sky background or too spread out.

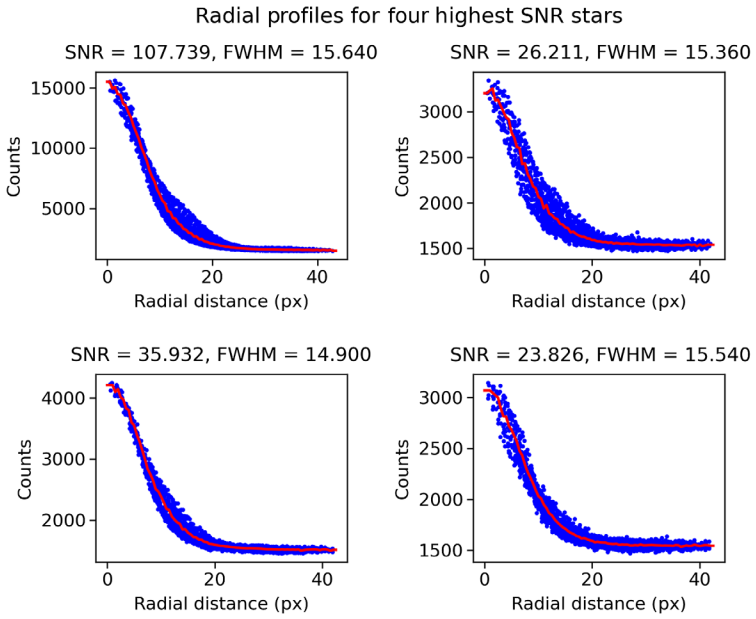

Fig. 5 Radial profiles of the four highest SNR stars from the image and apertures in Fig. 4. The *x*-axes are radial distance away from the center-of-mass of the star, in pixels, and the *y*-axes are the number of counts of these pixels (blue dots). The blue points are the values of each pixel around the center-of-mass of the star, and the red curves are the azimuthally averaged profiles of these pixel values, which we use to calculate the FWHM. The SNRs and FWHM in pixels are indicated.

minimum FWHM to be larger than the diffraction limit and flatten out the characteristic "V" shape into an approximately parabolic shape (at least for small displacements from the minimum). For this reason, we choose to fit measurements of FWHM as a function of focus position to a parabola using a scipy.optimize nonlinear least-squares curve fitting technique (curve_fit). Once an acceptable fit is found, the focuser moves to the fit local minimum. We reject any parabolic fits with a negative quadratic term to not unintentionally drive the focus mechanism out of range. Otherwise, if the parabolic fits predict a minimum beyond the range of the focus positions explored, the focus position is returned to the original position before any temperature-based adjustments are made by the software, and this result is logged for future human intervention on a subsequent night to return the focus position to an optimal position. We choose this approach, rather than setting the focus to the edge of the focus range explored and rather than broadening the focus range search, to prevent a hardware limit switch from being triggered. The optimal focus position does not change drastically from night to night, so by resetting to the previous night's optimal position, we typically achieve at least passable focus. For the analysis of light curves and exoplanet transits, in particular, an optimal focus is not necessary to obtain usable data, and in fact, defocusing may be advantageous for preventing a target from saturating.[30–32] This entire procedure takes from 5 to 15 min depending on the set exposure time of the observation, with ∼5 s of overhead between each image, but because we only have a single camera, this entire duration is used up on focusing and not collecting science data.

Second, after the initial focus at the start of an observing sequence, we wish to maintain that focus as long as possible, without interrupting science data collection. Our science case, consisting of long-duration uninterrupted light curves to detect transiting planets, requires collecting continuous data. However, our coarse focus procedure models the FWHM as a function of stepper motor position and would require moving the focuser to test positions that will be significantly out of focus. Running this procedure again during science observations would introduce large light curve systematics resulting from a changing FWHM. Alternatively, pausing science observations to rerun this procedure, if illtimed, could potentially miss key times such as the exoplanet's ingress or egress. Consequently, the focus is later adjusted over the course of the night based on a deterministic linear temperature-dependence model that we obtained experimentally from ongoing observations: as the telescope cools and the primary mirror shrinks,

we move the focus inward. The current linear model is expressed as $d(T) = d_i + \xi(T - T_i)$, where d_i is the initial best-fit focus position at some initial temperature T_i, T is the current atmospheric temperature, ξ is the thermal focal drift that we obtain experimentally, and d is the current focus position. The current model assumes a thermal focal drift of $\xi = 2$ steps/°F, which appears to qualitatively hold the FWHM steady over the course of an entire night. We defer quantifying our long-term focus performance to future work. We also do not apply any corrections for changes in the gravity vector as the telescope altitude changes. Finally, we also developed a custom GUI to allow for manual adjustments to the focus position in the case that the automation code fails. This was required because the serial COM port does not allow for multiple simultaneous connections from both our automation software and RoboFocus, and we chose not to implement a pass-through COM port to relay serial communications to and from RoboFocus.

3.6 *Calibration Imaging*

The calibration module takes the desired flat-field and dark images either before observing starts or after it finishes, depending on the configuration settings (set in the general configuration json file). The code is designed to take a set of flat-field images for each filter in which science observations are being collected and a set of dark images to match each exposure time of the science images and flat-field images. Flats are taken first by pointing at the closed dome with the flat-field lamp turned on. First, a test exposure is taken with an exposure time of a few seconds, with the exposure time being set to an initial guess on a per-filter basis and based upon typical flat-field exposure times. Then, the median counts are measured in the initial test image for a given filter. If the initial test image has saturated median image counts, then the exposure time is halved, and the test exposure is repeated until the minimum exposure time is reached. If the flat-field image is still saturated in the minimum exposure time, the flat-field exposure time is left at the minimum exposure time; we assume that this scenario is never encountered in practice because it would be impractical to obtain any flat-field exposures. Once nonsaturated median image counts are obtained in the test exposure for a given filter, the flat-field exposure time is scaled to obtain 15,000 median counts in each flat-field image; these exposure times are stored in a variable for later use with the dark images. The number of flat-field images collected per set per filter is also a configuration parameter that may be adjusted. Under nominal operations, the flat-field exposure times for each filter do not change on a nightly basis, varying by 1 to 2 s for a 15-s exposure. By recalculating it each night, however, we prevent the need for making adjustments in the future due to luminance drift over longer periods of time from various effects—dust accumulating on the primary mirror, replacing the flat lamp bulb, drift in the telescope and dome park positions, etc. The detector temperature may also introduce smaller variations from night to night.

For dark images, we choose to collect a set of darks for each flat-field and science image exposure time. We choose to collect images in this way, as opposed to using bias images and exposure time scaling of the counts in dark images with a fixed exposure time. The difference in collection times between these two methods is minimal, and this method is more convenient for our data reduction pipeline. We do not have to separate bias counts from dark current and are not reliant on dark exposure time scaling, which may be inaccurate if there are nonlinear effects present in the dark current due to nonideal detectors. Small variations in the dark current due to differences in the detector temperature, as discussed, may also lead to small inaccuracies if new images are not collected nightly. This collection method is not novel or unique and has been used, for example, in IRTF's iSHELL spectrometer.[33] To maximize potential observing time, we do not attempt to take calibrations for each target separately, but we take calibration images for an entire night of observing at once. We typically do not enable calibration imaging before observing because the GMU telescope is often used manually for astronomy classes and other observations prior to research observing, and thus calibration images are usually obtained at the end of the night.

During nights in which the weather is inclement, either temporarily or for the remainder of the night, and if the calibration observations are set to be taken at the end of the night, our software takes advantage of the downtime and collects the necessary calibration images for the current target and any previously completed targets before attempting a possible reopen. The software does not collect calibration images for future targets that have not begun observations yet. The weather module has a minimum 30-min timer before reopening after a weather

shutdown (see Sec. 3.8), so this is an ideal time to collect such calibration images as the upper limit on the clock time to collect calibration images for a single target is about 30 min (assuming 120-s science exposures). However, this does mean that, when our Observatory closes due to weather, it will remain closed for either a minimum of 30 min or the duration of the calibration image collection, which can exceed 30 min on rare occasions when more than one target needs calibration images, before it checks the weather again and determines whether or not the dome can reopen.

3.7 *Guiding*

The guiding module tracks the movement of stars between images and adjusts the telescope's position to keep their alignment relatively stable. Because we only have a single detector, guiding procedures must utilize the science observations themselves. As mentioned in Sec. 2, our CCD camera is currently mounted such that the image x- and y-axes align with the negative right ascension (RA; α) and declination (Dec; δ) axes, with a 180-deg field rotation. This is shown in Fig. 6 using AstroImageJ (AIJ)[34] for an arbitrary field of view. So, our automated software assumes that this angular offset between the sky and our CCD axes is a constant, and we give it a configuration parameter, currently set at 180 deg, in which counterclockwise angles are positive in the reference frame where RA increases to the left and Dec increases upwards. For axis orientations that are mirrored (i.e., for no field rotation, both positive image axes would align with the positive celestial axis, whereas the other positive image axis would align with the negative corresponding celestial axis), we have another configuration parameter as a simple Boolean to flip the y-axis orientation.

Stars are first located in the image with the find_peaks function as in Sec. 3.5, and an optimal guiding star is chosen from these peaks by finding the brightest star that (1) is not saturated; (2) is not within 500 pixels of the image borders, to prevent the possibility of it drifting out of the frame; and (3) is not within 100 pixels of any other detected stars, to prevent detecting nearby companions instead of the guide star and inferring movements when there are none. If no stars are found in the image that fit these criteria, the guiding procedure is unable to get a stable measurement of position in the image, so it waits for the next image to try again. This may happen due to low sky transparency or a sparse field of view, but we find that typically conditions are good enough to find at least a few. However, if multiple are found, we only choose the

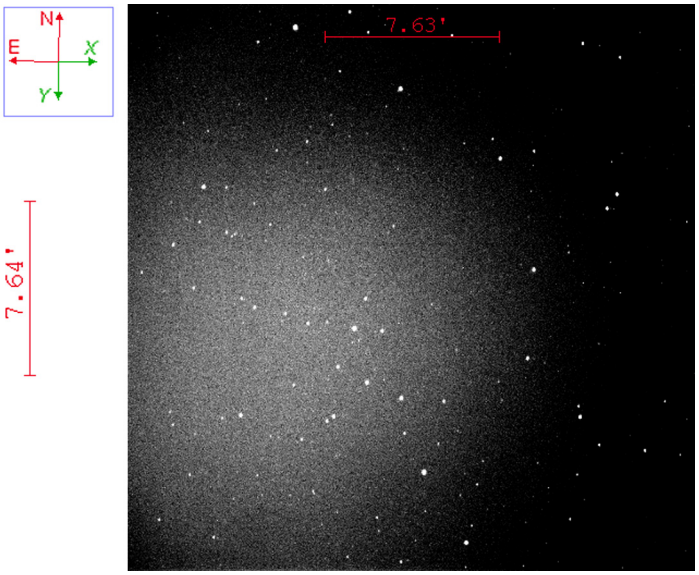

Fig. 6 A raw field image from our telescope showing the relative orientations of the image x and y axes with the North (+Dec) and East (+RA) axes along the sky in the blue square. Angular scales are shown as red bars. The image is scaled to gray-scale to maximize contrast, so all stars show as white circles in the image; sky emission has some curvature due to some vignetting from our tertiary mirror, resulting in lower transmission and sky counts toward the right of the image.

brightest one. After locating a guiding star from the first science image, we do not reapply the find_peaks algorithm to the entire field of view for every image thereafter. Instead, to save computation time and allow for more responsive guiding, we extract a subframe around the guide star and only utilize image data from this subframe. Choosing only a single star simplifies and further reduces the computation time required in this process without sacrificing much accuracy. This allows us to execute guiding moves $\lesssim$1 s after each image is saved. Because these moves are small, we begin the subsequent exposure simultaneously to reduce overhead, which does not have a noticeable effect on the resultant image. After choosing a guide star, the offset between this star's current position and the desired position (set by the first observation) is calculated, which is then transformed into the telescope move necessary to return the star to its original position. The telescope move distances and directions in each axis are calculated using the standard counterclockwise rotation matrix with the field rotation angle γ

$$\mathbf{r}' = \begin{bmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} = \begin{bmatrix} \Delta x \cos\gamma - \Delta y \sin\gamma \\ \Delta x \sin\gamma + \Delta y \cos\gamma \end{bmatrix}, \tag{2}$$

$$\mathbf{j} = p \begin{bmatrix} \eta_\alpha(\Delta x \cos\gamma - \Delta y \sin\gamma) \\ \eta_\delta(\Delta x \sin\gamma + \Delta y \cos\gamma) \end{bmatrix}, \tag{3}$$

where x and y are the object positions in the newest frame (in pixels) and x_0 and y_0 are the original positions (also in pixels) from the first frame, so the differences are Δx and Δy. Our image plate scale is p, η_α and η_δ are dampening coefficients, and $\mathbf{r}'$ is the undampened position vector in RA/Dec coordinates, in units of pixels. To get $\mathbf{j}$, the final telescope jog distances in the right ascension and declination axes (in arcseconds), we multiply $\mathbf{r}'$ by the plate scale and the damping coefficients. Because our current guider field rotation is $\gamma = 180$ deg to within $\sim$1 deg, the sin γ factors are 0, and we are left with simply coefficients of -1 from the cos γ factors. This way, a positive $\mathbf{j}$ component corresponds to a positive displacement in RA/Dec, which we can correct for with a telescope move in the same direction. If the "flip y" Boolean configuration parameter is true, Δy is simply negated.

The dampening coefficients η_α and η_δ, independent for the RA and Dec motors, are applied to the telescope move amplitudes to avoid over-correcting the telescope position, caused by the variability inherent in the atmospheric distortions to the PSF and centroid location from observation to observation. By testing a series of images with both coefficients set to 1, we observed how far and in which directions the targets tended to drift over time, and we found that the guiding was under-correcting in right ascension and over-correcting in declination. We then adjusted the coefficients and repeated this process until we landed on results that produced stable guiding over the course of an entire night of observing. We found that we achieve optimal guiding performance with $\eta_\delta = 0.75$ and $\eta_\alpha = 1.25$. If the software determines that a total offset of $|\mathbf{j}| \geq 15''$ is necessary, it skips the telescope move and resets the position on which it is guiding. This is necessary with our telescope to avoid overcorrecting during its RA jumps (Sec. 2). The jump ruins the current science exposure but is usually resolved by the time the next exposure starts.

Atmospheric conditions and the aforementioned RA jumps may cause the guide star to exit its subframe, in which case the guiding routine will fail to find an appropriate guiding star. If it cannot find a star for three consecutive images, we remove the subframe and analyze the entire image again to find a new guiding star.

3.8 *Weather Monitoring*

Perhaps the most important task to be automated with telescope observing is a real-time assessment of weather conditions and whether or not the weather permits the collection of on-sky data. Factors such as high humidity ($\geq$ 85%), high winds (gusts $\geq$ 25 mph), clouds (cloud coverage $\geq$ 75%), rain (any precipitation within $\sim$20 miles), fog, snow, daylight, and even ash and large smoke particles from fires both nearby and distant (such as the recent 2021 Canadian fires in British Columbia) can shut down an observatory. We have implemented multiple checks for weather, including data on humidity, precipitation, and wind speed from our local Davis weather station[35] shown in Fig. 7, but also precipitation, clouds, humidity, and weather radar from web-based weather websites, namely Refs. 36 and 37.

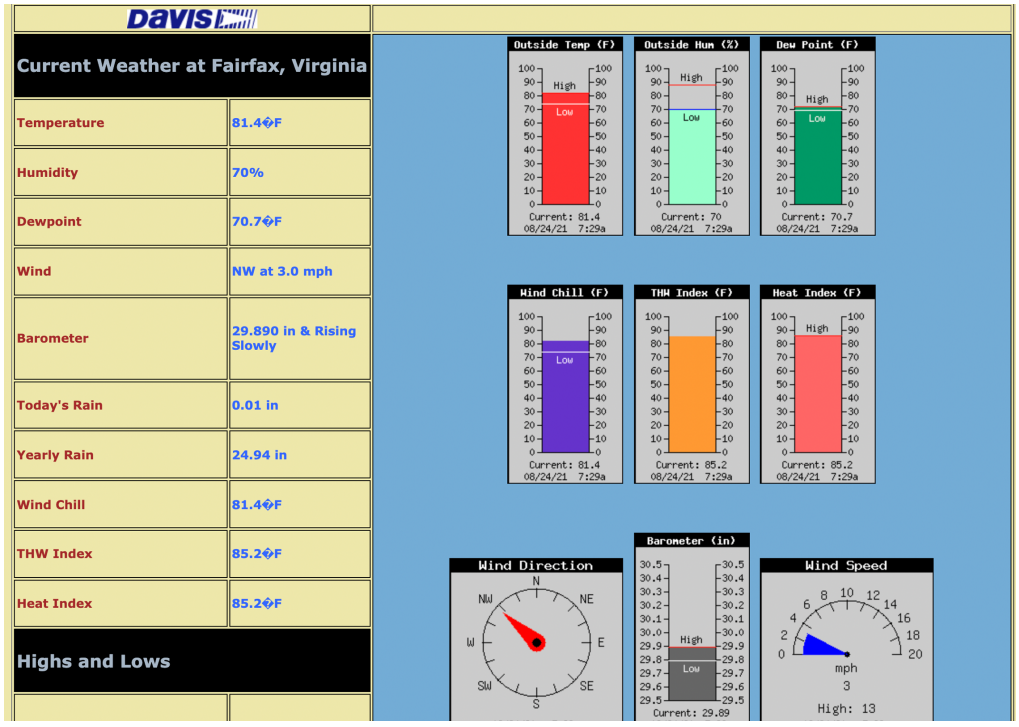

Fig. 7 An example of of our local Davis weather station web page. The wireless and solar-powered weather station is located atop the control room rooftop structure and thus provides conditions in the immediate vicinity of the observatory. The data are currently collected by a computer running Windows 95 and uploaded via FTP to the weather station web page, which is updated once every 10 min. Shown on the left are various measurements as indicated, and shown are the right are some of the same measurements shown graphically as single-bar histograms, with highs and lows from the past 24-h indicated as horizontal lines. Wind speed and wind direction are shown as an odometer-like display at the bottom.

The weather module checks current conditions every 6 min asynchronously to ensure that it is still safe to have the telescope open. This 6-min cadence is an adjustable configuration parameter and was chosen to not alias with the weather station update frequency, which updates every 5 min (thus, any update time shorter than 5 min would result in wasted/repeated computations). The code considers a failure to connect to the Internet as a reason to close; in this case, it is impossible to determine the current weather conditions. If conditions are such that the dome should be closed, the camera exposures are halted, the dome closes, the telescope parks, and then the software waits 30 min before checking the weather conditions again to see if conditions permit reopening; the 30 min period is another adjustable parameter. This built-in delay prevents the software from repeatedly opening and closing the dome when weather conditions are marginal, given the natural time-scale of local weather variability and the time it takes for the dome to open and the telescope to begin re-observing. In the time that the observatory is shut down due to weather, the software may begin taking calibration images as mentioned in Sec. 3.6. If the current ticket's observation end time passes while shut down due to weather, the software moves on to the next target or terminates if that was the final target in the observing queue, as explained in Fig. 3.

3.9 *Thread Monitoring and Error Handling*

Due to the multithreaded structure of our design, a "heartbeat" monitor to keep track of the status of each thread becomes crucial for maintaining stability in edge cases that may cause crashes in one or more threads. We currently have 12 threads running simultaneously during an observation (see Table 2): the heartbeat monitor periodically checks each of these threads with the exception of itself and the main thread, to ensure that none have crashed. If the heartbeat monitor detects a crash event, it will record the event in the logs and attempt to restart the thread. This is our main

and only method for handling errors in unexpected cases. If the heartbeat monitor itself crashes, we currently have no safeguard to reinitialize it, but the code can technically continue to operate without it (at least until another thread crashes). We also have no software-based safeguards if the main thread crashes. Our Ash dome does have a built-in safety feature that causes it to close if it does not receive a heartbeat signal from the ASCOM dome controller for 5 min, but our telescope does not have a similar feature and may be left continuously tracking. This is a current weakness and a necessary area of development for the code to be improved in the future. However, because the main thread acts primarily as a skeleton for the observing sequence, and delegates a vast majority of important operations to separate threads, the main thread itself is at a lower risk of crashing compared with the others.

In the cases of foreseeable errors, we avoid most crashes on hardware threads through the use of try/catch blocks. The main thread can then check if a hardware operation is completed successfully, and if not, can attempt it again or give up. This approach is taken for telescope and dome slewing/parking. However, a failure in the observing sequence may not always present itself as an error. For example, if the telescope begins to slew outside of physical limits. We already check coordinates before executing a telescope move, but sometimes (if the telescope was not properly parked or became twisted from a previous manual observation) the slew path might attempt to take the telescope into the ground on its way to the destination. We handle this by asynchronously checking the telescope's coordinates 10 times/s while slewing and aborting immediately if they leave the physical limits. If this happens, we first try parking and re-slewing, but if it occurs twice in a row, we give up on observations for the night, as the problem must be diagnosed by a human. We implement a similar procedure if the telescope leaves physical limits while passively tracking. On rare occasions, we have also encountered crashes with MaxIm DL, closing our camera connection, so we have implemented a check that restarts the software and reinitializes a camera connection before resuming exposures. We do not track the failure rate of the code rigorously, but over the past year, we estimate that failures to collect useful data (for any reason) occur on $\lesssim 5\%$ of nights.

3.10 *Testing*

Once enough of the base hardware and core structure of the code was in place to allow for end-to-end test runs, such as the weather, telescope, dome and camera modules, and core systems, all code development was thereafter conducted in tandem with testing on every clear night available. This allowed for both refinement of the core structure and the accelerated implementation of the new focusing, guiding, and calibration features simultaneously. For testing nights, we would typically create an observation ticket for a low-priority TESS target to minimize the potential loss of data in the case of failures due to the software. The software would then be initialized as detailed in Sec. 3.2, and the observers would stay to monitor for errors and unexpected behaviors. If an error was encountered, it would be handled according to its severity. For inconsequential errors that would not severely affect data quality, the behavior was simply noted and a fix would be implemented on a subsequent day. For more severe errors, same-night fixes and restarts would be attempted whenever possible; this testing approach in particular accelerated the development and testing cycles of the software. Eventually, the higher-level observing controls such as guiding and focusing were implemented sequentially as modules, once they had each been approved for testing on sky after code reviews. The first two of these modules were guiding and calibration imaging, both of which required iterations of bug fixes after the initial on-sky tests. We initially had the guider angle γ set to 0 deg instead of 180 deg, which caused guiding corrections to be in the opposite direction and increased the RMS position of the stars rather than stabilizing them. Additionally, if the telescope pointing model is off and the science target star is not in the field of view to begin with, the guider will not be aware and will begin guiding on the wrong star field for the entire observation, which has occurred. In this case, a manual fix of the telescope pointing model is required. CCD images would also overwrite old ones if the code was restarted on the same night. This was fixed by adding a check for current images in the directory and incrementing the image number by the largest existing number plus 1.

This was followed by focusing, which has gone through more substantial changes within its core structure and design in addition to the bug-fixing. We initially attempted a more human-like

focus routine that would decide which direction to move the focuser after each image until it reached a point at which both directions led to a worse focus. We found in practice that the grid search technique was better because it is guaranteed to converge or terminate after a fixed amount of time. Our initial FWHM calculations were also inaccurate for severely defocused donut or torus-shaped PSFs because we performed an azimuthal average around the geometric center. After switching to a center-of-mass, this problem was eliminated. The most recent new addition was the heartbeat monitor, which again required bug fixes after on-sky testing. An ongoing problem for the heartbeat monitor is that it is not always successful in recovering from the unexpected failure or termination of an individual module thread because just restarting the thread may not address the underlying issue of why the thread crashed.

A large problem that we have faced in all modules of the code throughout its development is threads that get stuck waiting for an event trigger that never happens, due to an oversight in the portion of the code that is supposed to set the trigger. This is a problem unique to multithreaded systems, and it can cause unexpected behaviors that are difficult to track down to a root cause. It is possible to place maximum wait times with the wait method, but this may cause more severe problems if things across threads are executed before they should be. As an example—we initially forgot to place an event trigger after the continuous temperature-based focusing for a target finishes (when the target's observation time has ended). This caused the program to become stuck at the beginning of the next target's observations, as it began waiting for this event to be triggered before beginning the initial coarse focus routine, causing the entire observing sequence to stall for the night.

4 Results

We have observed TESS candidate exoplanets with our automated control software for 171 partially or fully clear nights and counting, not including testing nights.

4.1 *WASP-93b*

WASP-93 b is a near Jovian-mass ($M_p = 1.47 \pm 0.29\, M_J$) and Jovian-sized ($R_p = 1.597 \pm 0.077\, R_J$) planet orbiting an F4 star.[38] Its large size in comparison with its stellar host, $(R_p/R_*)^2 = 0.01097 \pm 0.00013$, signifies a large transit depth that makes it an ideal candidate for assessing the upper limit of quality of our automated results. The GMU telescope observed a WASP-93 b transit event on UT November 10, 2020. Observations were performed in the R band ($\lambda_0 = 6940$ Å and $\Delta\lambda = 2070$ Å). Observations began about 90 min prior to the transit to achieve a sufficient out-of-transit relative flux baseline. Exposure times were 120 s. The automated focusing was able to achieve $3.4 \pm 0.2''$ FWHM seeing over the course of the entire night, which is representative of good seeing conditions in Fairfax, Virginia, United States. A total of 153 exposures were taken, with the active guiding keeping the target star relatively stable within a few pixels of its original position. An ingress and partial egress was recovered, but cloudy conditions forced a pause in observations during the latter portion of egress. Dark and flat-field images were then automatically collected to be used for data reduction during the cloudy conditions at the conclusion of the on-sky observations.

The data were reduced using an AIJ[34] pipeline in which flat-field images in the R filter are dark subtracted with a median-combined dark frame image of the same exposure time and then median combined and normalized into a master flat. Then the science images are dark subtracted with a separate median-combined dark image matching their exposure time, and the master flat is divided out. The images are then plate-solved using the Astrometry.net plate-solving server, and we perform aperture photometry with a radius of $4.42''$, inner sky annulus $7.82''$, and outer sky annulus $11.90''$, for the target star and ~10 comparison stars in addition to all stars within $2.5'$ catalogued by Gaia.[39] The final light curve, along with a model fit and residuals, is shown in Fig. 8. The depth, period, inclination, and $(R_p/R_*)^2$ found by the best fit model are all consistent with the information found by Ref. 32, with the exception of the transit duration; our best fit $(a/R_*) = 7.5$ disagrees with their Markov–Chain Monte Carlo (MCMC) results, which put it closer to 5.9, likely due to our incomplete egress.

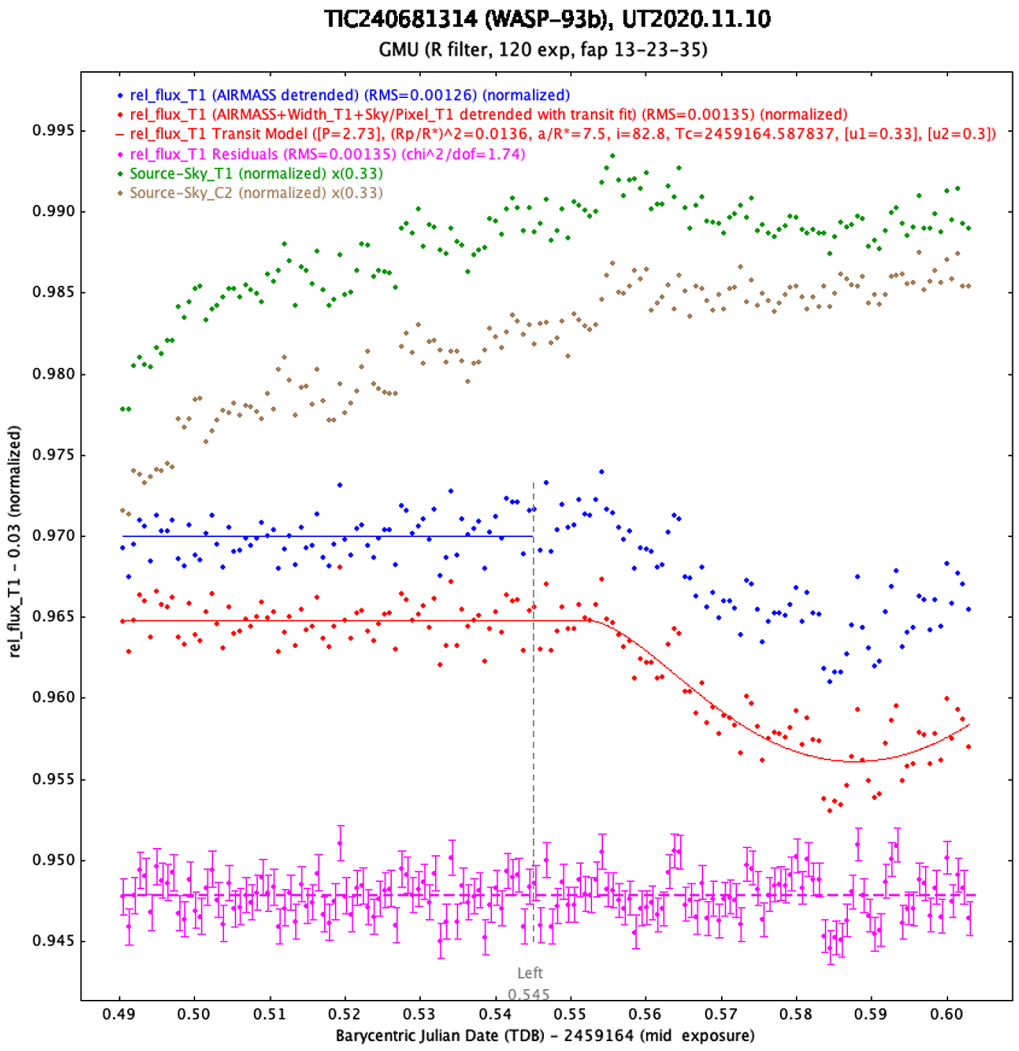

Fig. 8 An example automated observation light curve of WASP-93b, produced with AIJ.[34] Note that the vertical axis is normalized to unity, and then an arbitrary shift is applied for each flux time-series (e.g., a shift of –0.035 is applied for the airmass-, PSF width-, and annular sky/px sky brightness-detrended light curve (red dots). The blue dots are a light curve that is detrended with airmass only. The red light curve is detrended with a joint fit to a transit model (red line) with a reduced $\chi^2 = 1.74$. "T1" refers to the primary target, WASP-93b. The magenta data are the residuals of the transit model. Both the blue and red dots are corrected using a collection of comparison stars, whereas the green and brown dots show the raw, undetrended (but still normalized) flux of WASP-93b and a companion star. These raw light curves are also scaled arbitrarily by a factor of 0.33 to increase the visibility of the plot, but they show how the airmass, FWHM, and sky transparency affected the raw flux of the target over the course of the observation.

4.2 *TIC 321669174.01, aka TOI 2081.01*

TESS input catalog (TIC) 321669174.01 or TESS Object of Interest (TOI) 2081.01 is a planetary candidate (PC) around an M-dwarf star with a $T_{\text{eff}} = 3742 \pm 157$ K, $R_* = 0.52 \pm 0.02$ $R_\odot$, and $M_* = 0.51 \pm 0.02$ $M_\odot$.[8] The transit has an expected depth of 1.2 ± 0.1 ppt and a period $P = 10.5050 \pm 0.0002$ days, so in addition to assessing the quality of data achieved with a midtransit shutdown, this analysis demonstrates the sensitivity of our ground-based observations and analysis due to the relatively shallow transit depth. The GMU telescope observed the transit of TOI 2081.01 on UT May 13, 2021. Observations were again done in the R filter, and baseline observations began about 2 h before the predicted transit ingress to achieve the maximum

possible out-of-transit baseline. Exposure times were 80 s, with the automated focusing achieving a focus FWHM of $3.86''$ at the beginning of the night. At the time of these observations, our telescope's right-ascension motor encoder tape for absolute positioning had de-laminated, so our telescope's relative pointing accuracy was initially in error by $\sim 10'$, within our CCD field of view. We sent a manual jog command to center the target within the frame after the first observation. The guiding module performed well and kept the star stable throughout the night. However, due to a midobservation pause during the night due to weather and having to reslew the telescope, the position of the target changed, and we did not manually correct the error in the telescope pointing. Exposures continued hours after the transit until sunrise to achieve a symmetric baseline, allowing us to collect a total of 250 images. Darks and flat-field frames were collected during the temporary midobservation shutdown.

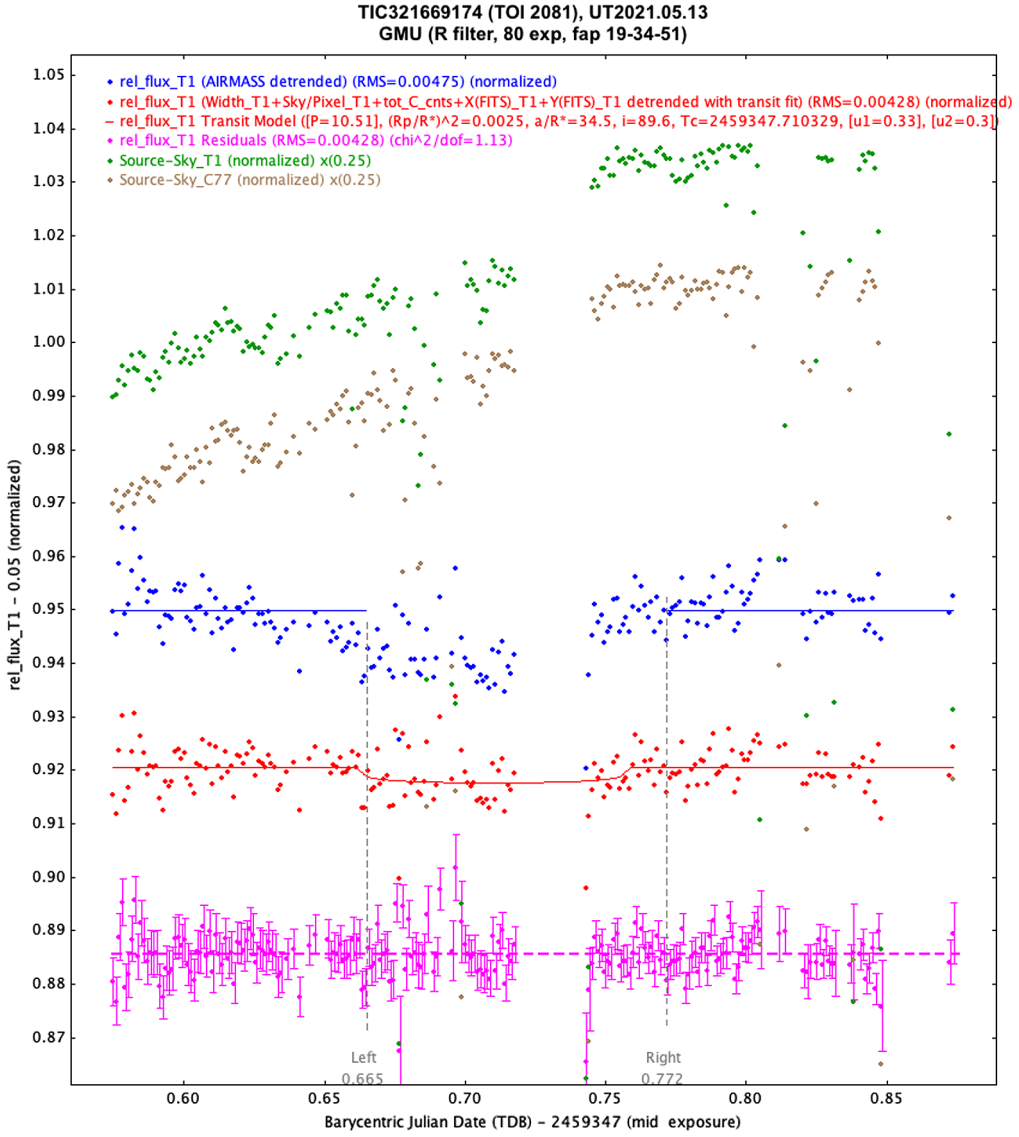

Fig. 9 An example automated observation of TOI 2081.01. As shown in Fig. 8, the figure is generated with AIJ, with light curves normalized and arbitrary shifts applied. The raw absolute photometry light curve prior to companion star correction and detrending is plotted as green and brown dots; blue data is detrended with airmass; and red data is detrended with width, sky per pixel, total counts, and x and y positions in the image. The red line is the best-fit transit model with a reduced $\tilde{\chi}^2 = 1.13$, and pink data are the residuals.

The data for this target was reduced and the light curve was extracted using the same pipeline as previously detailed for WASP-93 b, using AIJ and Astronomy.net. This time, an aperture size of 6.46″ was used with an 11.56″ inner sky annulus and 17.34″ outer sky annulus. The light curve, model fit, and residuals are depicted in Fig. 9. As shown, we find an $(R_p/R_*)^2 = 0.0025$, $(a/R_*) = 34.5$, inclination $i = 89.6$ deg, and time of conjunction $T_c = 2459347.710329$. The depth of 2.4 ppt is twice as deep as the expected 1.2 ppt, indicating a possible false positive, but AIJ does not estimate model parameter uncertainties/posteriors. A nearby eclipsing binary (NEB) check was performed on all nearby Gaia stars using AIJ, but none were found to be obvious candidates for a false-positive signal.

To evaluate the statistical significance of our results, we ran an MCMC simulation modeling our ground-based light curve, with the EXOFASTv2[40,41] software. We impose priors on log M_*, R_*, $T_{\rm eff}$, log g, distance, T_C, log P, R_p/R_*, cos i, and limb-darkening coefficients up to quadratic order as inferred from the TESS light curve and the stellar host characterization in the TIC,[42] with one exception: we adopt a loose prior on R_p/R_* to assess whether or not our data provides a tighter posterior distribution indicating that the data constrains the transit depth and thus is indicative of a statistically significant detection. These priors are summed in Table 3. The transit fit produced by the simulation is shown in Fig. 10, and the cornerplot is in Fig. 11. We see a result that agrees well with our initial fitting done in AIJ, and we observe well-behaved and nicely converged posterior distributions that signify our solution is unique and stable in the model parameter space. We even find that the median transit depth is deeper than the best fit depth from AIJ, as our depth posterior is $3.3^{+1.3}_{-1.1}$ ppt. This is a statistically significant detection of the transit depth of 3σ and is consistent to within 2σ of the expected transit depth from TESS. In comparison with our prior on R_p/R_*, which corresponds to a prior on depth of 2.5 ± 4.3 ppt, this is a substantial improvement in statistical significance. We also derive a consistent transit duration of $2.55^{+0.13}_{-0.38}$ h, and this is a clear indication that our observations represent a robust detection of the candidate transiting exoplanet. Thus, we conclude that our telescope automation program was able to recover a transit on the order of a few parts-per-thousand even while closing up due to a weather event in the middle of the transit observation.

Table 3 The prior probability distributions used to generate the MCMC from EXOFASTv2. u_1 and u_2 correspond to the linear and quadratic limb-darkening coefficients, respectively. The first two columns after the label represent the mean and standard deviation of a Gaussian distribution centered at the prior belief, and the final two columns represent hard limiting boundaries on the value of the posterior. A "–" signifies that no lower/upper bound was specified, so it can be formally thought of as $\pm\infty$.

Prior	Mean value	Standard deviation	Lower bound	Upper bound
log M_*	–0.290	0.017	–1	0
R_* ($R_\odot$)	0.516	0.015	0.1	1
$T_{\rm eff}$ (K)	3742	157	3000	4500
log g	4.7238	0.0086	—	—
T_C (days)	2459347.710	0.017	2459347	2459348
log P	1.0213961	8.1×10^{-6}	—	—
Distance (pc)	62.345	0.083	—	—
R_p/R_*	0.050	0.043	0	1
cos i	0.007	0.017	0	1
u_1	0.3	0.1	—	—
u_2	0.3	0.1	—	—

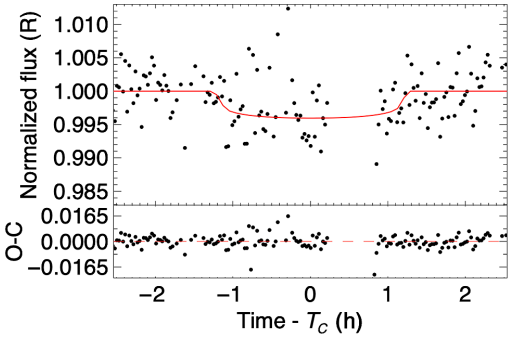

Fig. 10 The median MCMC transit model of TOI 2081 from the posterior distributions. For the upper graph, the vertical axis is the flux of the target star, which has been normalized so that the baseline out-of-transit value is unity. The horizontal axis is time, in hours, since the time of conjunction (T_C), or midpoint time, of the transit. The lower graph shows the residuals (observed–calculated) of our transit model shown in red.

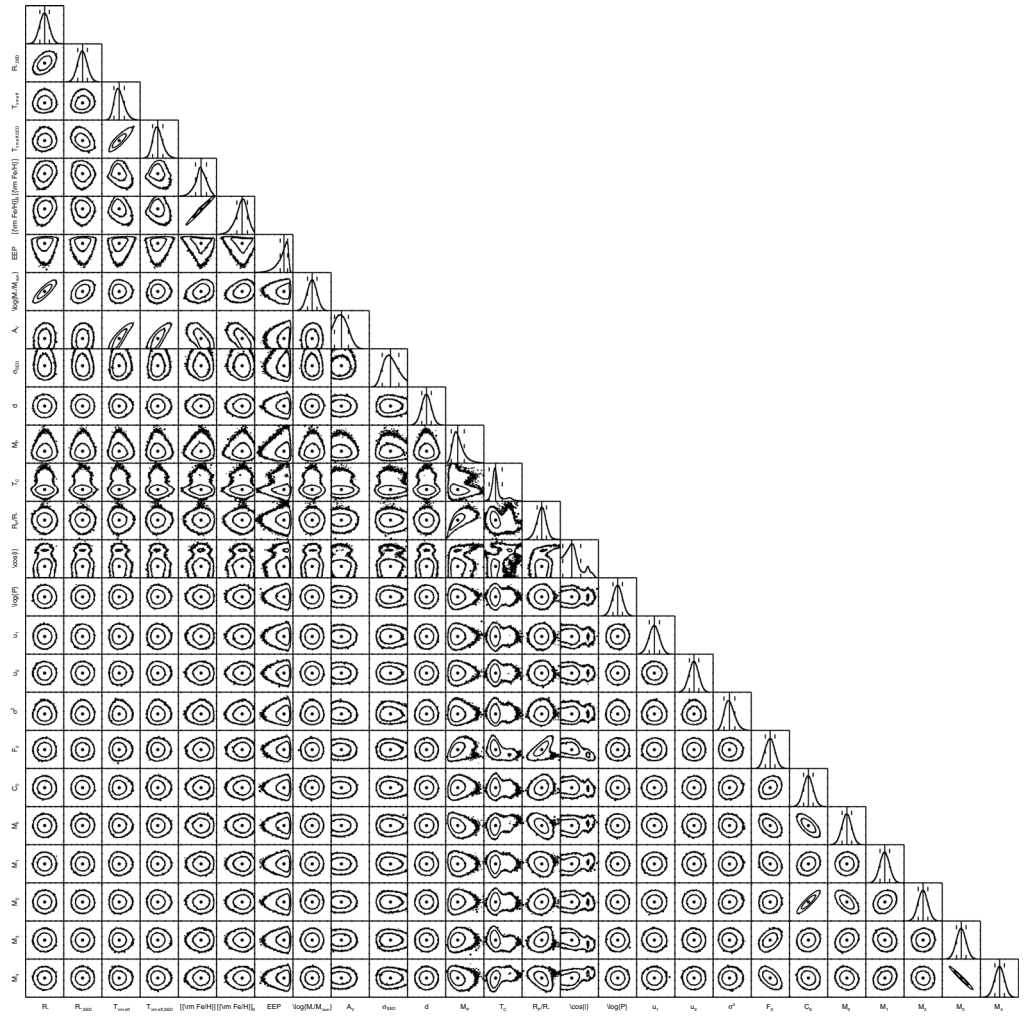

Fig. 11 The cornerplot showing the model parameter space explored by walkers during the MCMC simulation. The rightmost plots on each row along the diagonal show the one-dimensional posterior distributions of the model parameters labeled on the bottom axis, which are mostly Gaussian. All other plots show two-dimensional (2D) covariance plots between each pair of model parameters, such that uncorrelated parameters exhibit circular covariance contours (e.g., for a 2D Gaussian), whereas highly correlated parameters deviate from a 2D Gaussian, such as the highly-correlated metallicity and stellar effective temperature pairs of model parameters.

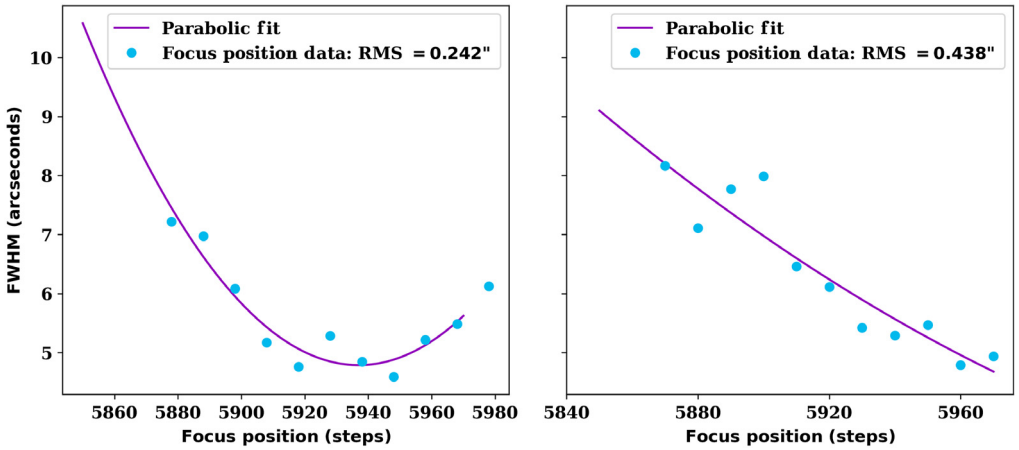

Fig. 12 Two plots of the software's FWHM measurements (in arcseconds) are shown in blue, and the associated parabolic model fits are shown in purple, as a function of the secondary mirror position (in steps, arbitrary zero point). The RMS of the difference between the data and model is shown in the legend. The two plots share the same vertical scale. On the left, the data was gathered on a single night for a single star. This data shows a night when the automation code performed relatively well and was able to find a minimum focus position that correlated with the data. On the right, the data was again gathered on a single night for a single star, but not the same night/star as the data on the left. This data shows a night when the code performed less optimally, as the ideal focus position was beyond the range of the focus steps explored; in other words, this particular night violated the assumption built into our code for quick focusing – that we start near an optimal focus.

4.3 *Focusing*

As seen in Fig. 12(a), on a clear night with relatively stable weather conditions, the code is able to step through 11 unique focus positions, fit a positive-concavity parabola, and move to a minimum focus, which on this night was $\sim 5''$. The central data point in the figure represents the initial focus position that the mirror was at before the software started, and the minimum of the parabola shows where it moves after the automated focusing procedure has completed. In this case, the minimum was relatively close to the initial position, which contributed to the success of the parabolic fit. During nights with less ideal and more variable seeing conditions or nights for which the initial focus position from the previous observing session is a large distance (in steps) away from the current optimal focus position or has been manually moved, our focus algorithm can yield inconsistent focus position minima. An example of this is shown in Fig. 12(b), where the initial focus point is a bit too far away from the optimal position, and the code is unable to obtain a full characterization of the parabolic shape of the FWHM as a function of focus stepper motor position. In these cases, when possible, a human observer uses the focus control GUI to correct the focus manually.

4.4 *Guiding*

Figure 13 shows a comparison between a night when the automated guider was utilized against a night when observations were performed manually and no guiding procedure was in place. The telescope drift in Fig. 13(b) from an inadequate telescope pointing model is readily apparent, and the large jump in the x- and y-positions is likely due to our telescope's unexplained "RA jumps" and Dec oscillations (Sec. 2). It is readily apparent from Fig. 13(a) that we are able to keep star positions steady and actively guide on the science images with an RMS as low as 3.23 pixels, or $1.2''$. This is a vast improvement to the nonguiding RMS as high as $64.4''$, a reduction of 98.17% in the y pixel direction (declination axis) and 98.02% in the x direction (right ascension axis). This is also beneficial to our photometric precision, data extraction, and modeling efforts post-observation, as having a more stable pixel position eliminates noise from differences in pixel sensitivities, bad pixels, and the gradual motion of the telescope. In general, our guide

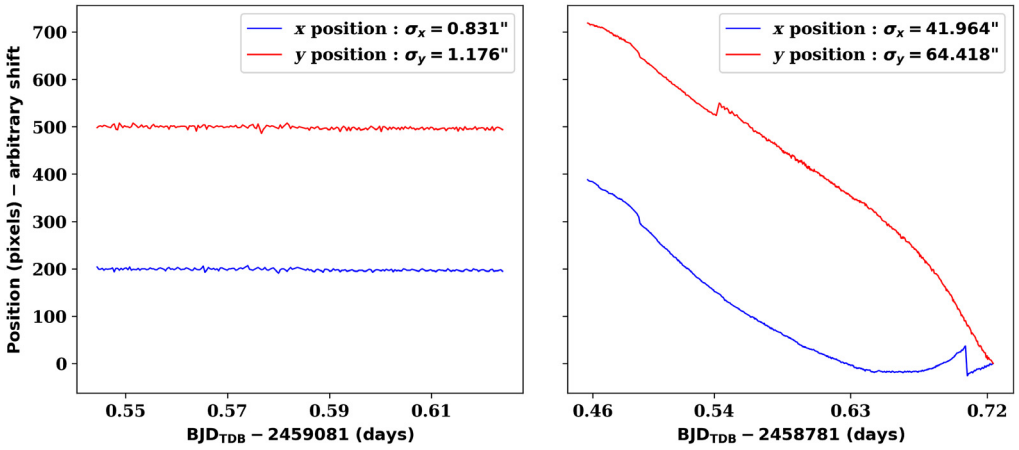

Fig. 13 Two plots of the software's guide star position data (in pixels) in each image are shown as a function of time (in Barycentric dynamical time). The *x* positions are shown in blue, and the *y* positions are in red. The standard deviation of the *x* and *y* positions is also shown in the legend. The two plots share the same vertical scale. On the left, the data were gathered on a single night for a single star. This was a night in which the automation code with guiding was utilized. On the right, again the data were gathered on a single night for a single star. This was a night in which the automation code with guiding was not utilized.

works this well in all but the most cloudy conditions. Because we do not have a second camera to take short exposures optimized for guiding, we must rely on the cadence of the target's exposure times for guiding. This means that fainter science targets will inevitably have a larger RMS deviation in guiding, as the guiding can only be adjusted as quickly as new science images are generated; thus we also regularly update our telescope pointing model manually to minimize the telescope drift during exposures.

5 Discussion and Conclusions

We have efficiently implemented an asynchronous and object-oriented framework for the GMU Observatory where each piece of hardware and higher-level function is associated with a Python class. By building upon these base hardware classes and constructing higher-level modules for important observing tasks, our software is able to monitor weather, acquire targets, focus, guide, and collect data and calibrations before shutting down, with minimal human intervention or surveillance required. Concurrent on-sky testing of software in parallel with the software development greatly accelerated the development of the automation control. The multithreaded structure greatly improved the efficiency and safety of our observations—overhead times for setting up or shutting down the observatory (∼5 min) are at least twice as fast as a single-threaded system could manage. Typical CCD readout times are ∼5 s and unavoidable for any observing system, and we have an additional ∼5 s of overhead between focus images, which is on par with a single-threaded system, but far superior to manual observations, which may require many seconds between images to determine the proper next focus move. We also eliminated ∼2 to 3 s of overhead between science exposures when guiding movements are being calculated and executed by starting the next exposure immediately, which is not possible for a single-threaded system. For a typical night with ∼200 exposures, this adds up to a nonnegligible 6 min of overhead saved. Checking the weather also cuts into observing time on a single-threaded system. Our combined collection of local weather station data, plus radar data for precipitation and cloud coverage, takes ∼2 s, saving ∼1 min per 3 h of observing. Continuous focus adjustments similarly interrupt observing on a single-threaded system, but in this case, the amount is negligible, <1 s per adjustment. Combined, on a typical night, we expect to have an extra ∼15 to 20 min of observing time compared with a single-threaded system.

The success of the automated focusing procedure relied on our two simplifying and reasonable assumptions: (1) we start the night near an optimal focus and (2) the focus position drift through the remainder of the night is linearly correlated with the ambient dome temperature and

less dependent on other factors such as the gravity vector. Using the science exposures, the guiding module has consistently stabilized pointing to an RMS $\sim 1''$ in all but the most cloudy conditions. The human observer creates the observation tickets for each target and initiates the observation sequence via the command line.

The results from WASP-93 b have demonstrated the functionality of our automation software and its ability to collect data with a quality on par with our manual observations. Further, our active guiding module enables reduced light curve systematics from pixel sensitivity variations by eliminating passive tracking telescope pointing drift; however, we did not directly quantify this improvement herein. The analysis of TOI 2081.01 has shown the software's versatility in suboptimal variable weather conditions to detect transits on the order of 1 ppt in depth. The automated focusing works well given the above assumptions, and our temperature-based linear focus drift model is continually adjusted throughout the night and does not require the interruption of science observations, avoiding additional gaps in the light curve for additional telescope focusing, either manual or automated. We do not identify any software-driven focus changes that significantly degrade the seeing FWHM over the course of a night, and in fact, some software-driven focus changes due to changes in temperature improve the FWHM seeing as in Fig. 16.

Every night, there are many follow-up targets from TESS and other exoplanet surveys that are observable by a meter-class telescope in the Northern Hemisphere. Since implementing the automation software, we have rarely missed an opportunity for observations on nights with clear or intermittently clear weather, and we are able to observe longer baselines on targets without pointing restrictions. We are also more easily able to work around telescope time allocated for classes and tours by requiring minimal manual setup. To date, we have achieved our goal of a more efficient observatory with over 171 automated observing nights and counting.

5.1 *Future Work*

We preface this section by emphasizing that this project's development has been primarily student-led, and although the present data quality and operation of the automation software have reached a reliable level of scientific utility, there are still areas for future improvements that we have prioritized. First, although we have executed a few multinight observation ticket sequences, more testing and development are needed. The thread monitor is not always successful in recovering from the unexpected failure or termination of an individual module thread. Log files build up relatively quickly and could be reduced in size to allow the storage and archiving of logs for much longer periods of time. Shorter focus exposure time scaling (as low as one-tenth) may also be explored to increase the efficiency of the initial coarse focus routine. Usage of "test" flat and dark images from previous nights could also allow us to partially correct for noise in focus images and achieve marginally improved focus measurements. Additionally, more interthread communication could allow us to pause calibration images if their collection time surpasses 30 min and the weather has improved (and resume them later), which could marginally increase on-sky time. Communication times between threads also limit us from getting an accurate measurement of the RA/Dec rate of change with a fine enough level of temporal precision to predict when an RA or Dec jump has occurred, but this is something that may be improved with further refinement, which would allow us to abort an exposure and save time starting the next one.

Second, the inherently modular design of our automation software allows for the easy addition of new features as we continue to expand the operations of the automation software. Currently, there are a few more hardware devices in the GMU Observatory that lack automation control: the tertiary telescope mirror, the near-infrared camera that was recently installed, and the dehumidifier located within the dome. Incorporating modules for these devices could allow us to perform automated observations in the near-infrared as well as all of the optical filters that we currently utilize or even both simultaneously with the use of the tertiary mirror control.

Third, the addition of a task scheduler module could automatically and intelligently choose and prioritize the best transit observations on a given night based on the target's observability and priority, a task that is currently performed by humans. Next, because the current software is command-line driven with a Python GUI front end, in the future we could pursue the creation of

an online server through a web development framework such as node.js to control the automation software. Fourth, an automatic compression module to fpack our FITS images[43] as they are being saved, followed by automatic data reduction and plate-solving, would cut down on data transfer times and storage needs for analysis and archiving. Finally, although there has been work done on documenting the general features and usage of the code for automated observing, the technical documentation within the code itself could be improved to allow for more transparency and collaborative code development.

6 Appendix

Here, we present a collection of three additional and representative light curves of TOIs 1333.01, 1724.01, and 3573.01 that were observed using our automation software (Figs. 14–17).

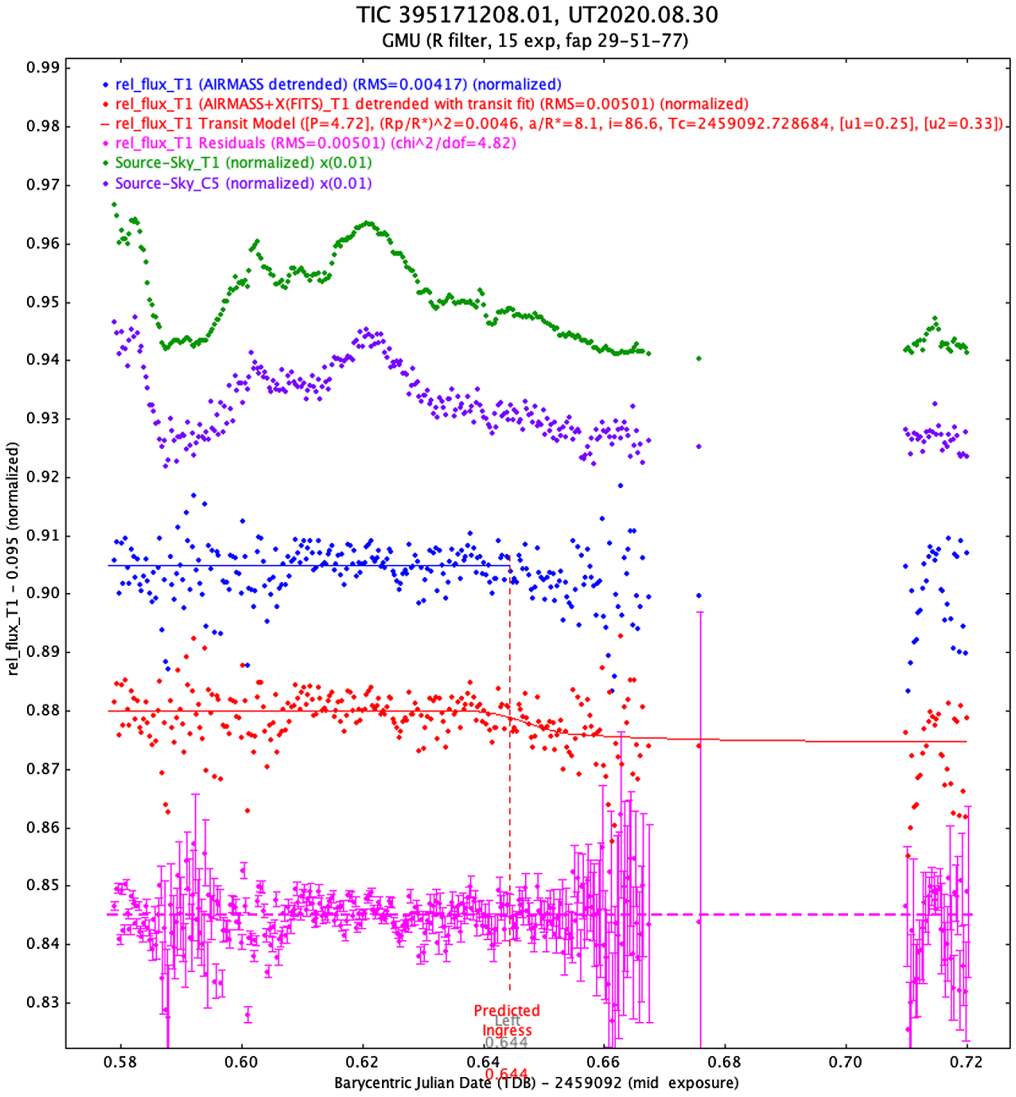

Fig. 14 This plot is produced with AIJ and follows the same conventions as in Figs. 8 and 9. We plot the light curve from an automated observation of TOI 1333.01 on UT August 30, 2020. Much of the data is missing due to very poor weather conditions and atmospheric transparency, as can be seen in the raw T1 and C5 flux time-series in green and purple, scaled by a factor of 0.01. However, the automated weather module was able to gather the ingress and some midtransit data. We find a best-fit $(R_p/R_*)^2 = 0.0046$, with a $\chi^2_{\text{red}} = 4.82$, which compares to the TESS mission value of 0.0055.

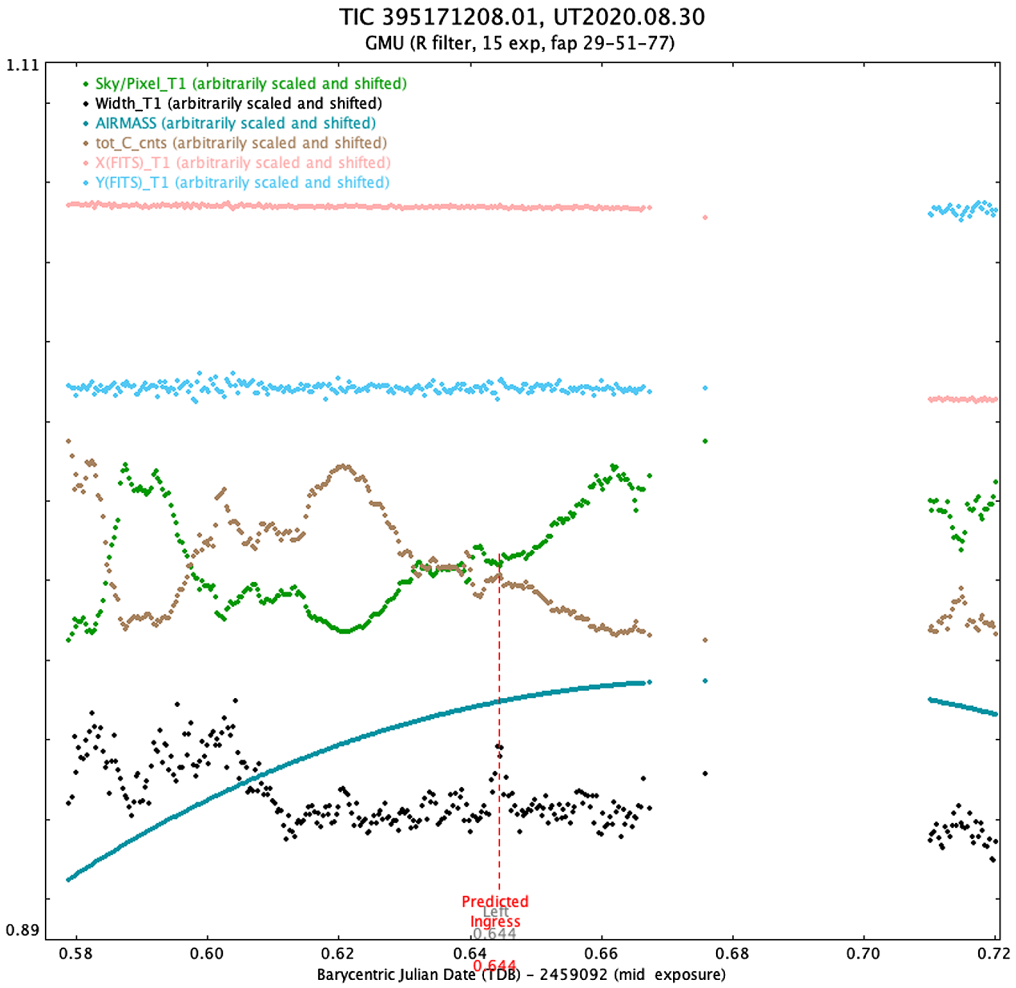

Fig. 15 This plot is produced with AIJ and is associated with the same dataset as Fig. 14. Here, we present a time-series of various weather and seeing-related quantities, arbitrarily scaled and shifted, to see their general shapes and trends. The series includes airmass, sky per pixel, total counts (summed across all-stars with apertures, which is useful for tracking atmospheric transparency when conditions are not photometric), PSF FWHM width, and X/Y image position of the target in pixels. Of particular note, after the initial focus, the telescope focus position is set using a deterministic linear model based upon the ambient dome temperature as described in Sec. 3.5; although the FWHM time-series is shifted and scaled, no significant low-temporal frequency FWHM drift is observed over the course of the night. By contrast, the sky brightness (sky/pixel) and transparency (tot_C_counts) show significant drift from the start to the end of the night, which is mirrored in the raw flux trends in Fig. 14.

Acknowledgments

MAR and PPP acknowledge support from NASA (Exoplanet Research Program Award #80NSSC20K0251, TESS Cycle 3 Guest Investigator Program Award #80NSSC21K0349, JPL Research and Technology Development, and Keck Observatory Data Analysis), the NSF (Astronomy and Astrophysics Grants #1716202 and 2006517), and the Mt Cuba Astronomical Foundation. Additional observatory automation efforts that we would like to acknowledge are PVS,[44] CAMAL,[45] and APF[46,47] and commercially available automation control software such as ACP[48] and MaxImDL.[49] The authors have no relevant conflicts of interest to disclose.

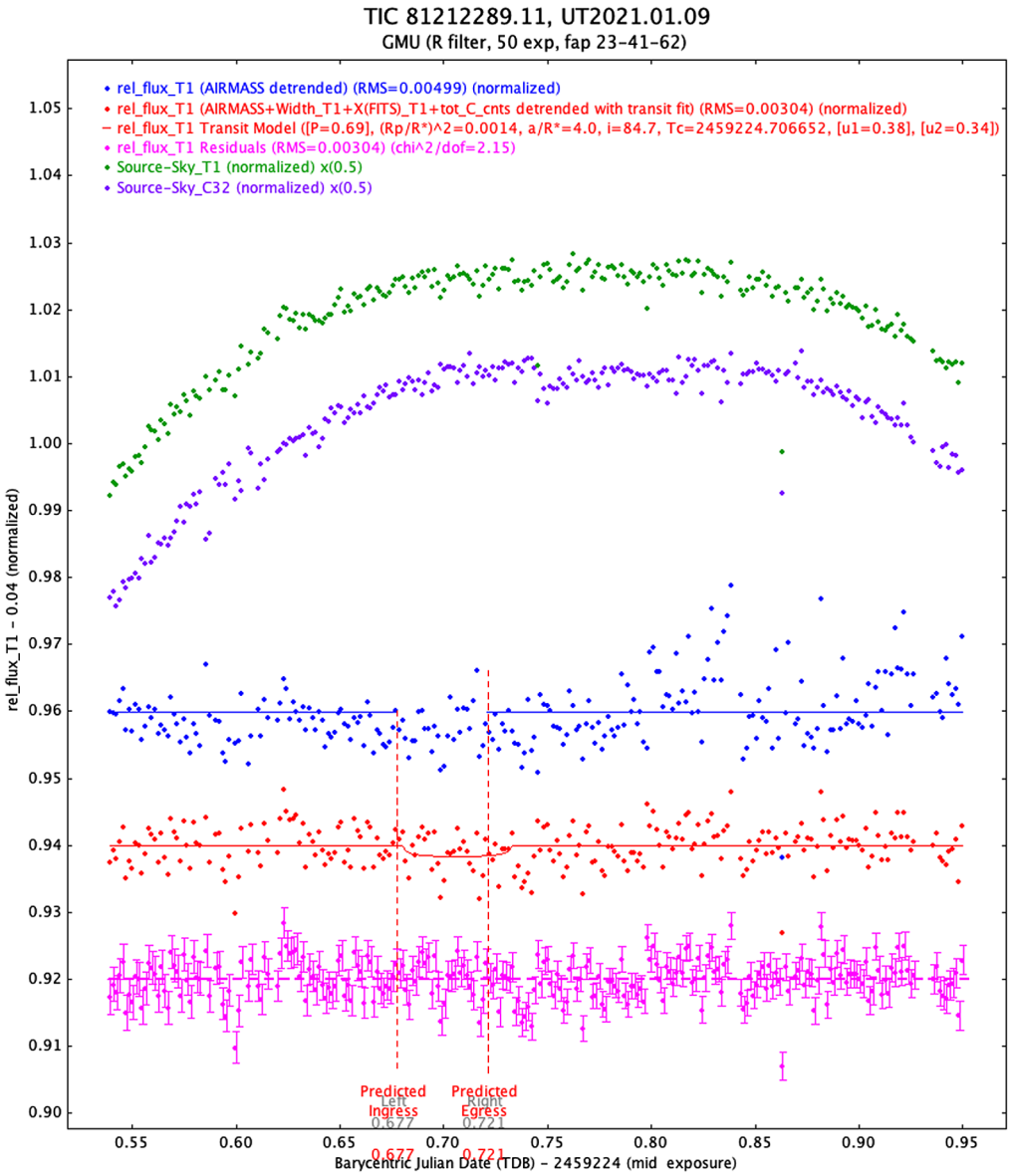

Fig. 16 This plot is produced with AIJ and follows the same conventions as in Figs. 8 and 9. We plot the light curve from an automated observation of TOI 1724.01 on UT on January 9, 2021. The complete dataset shows no transit signal, and an NEB check confirms a nearby target that produced the transit-like signal in TESS. This demonstrates the utility of automated observations in identifying false-positive candidates. We exclude a transit event matching the expected transit depth of 2.7 ppt from TESS during the predicted ingress and egress times, marked with the red dashed lines, reflecting the realization that a nearby eclipsing binary star produced the transit-like signal. A systematic at $BJD_{TBD} \sim 2459224.54$ appears to look like a substantially early transit egress, well outside the predicted time-of-transit uncertainty from TESS, but it is instead from an abrupt focus change at the start of the night, which could have been due to rapid cooling of the ambient temperature resulting in an over-correction in our focus model; however, the focus improved for the remainder of the night.

Code, Data, and Materials Availability

The OmegaLambda code repository is available on GitHub at https://github.com/Kakon24/OmegaLambda

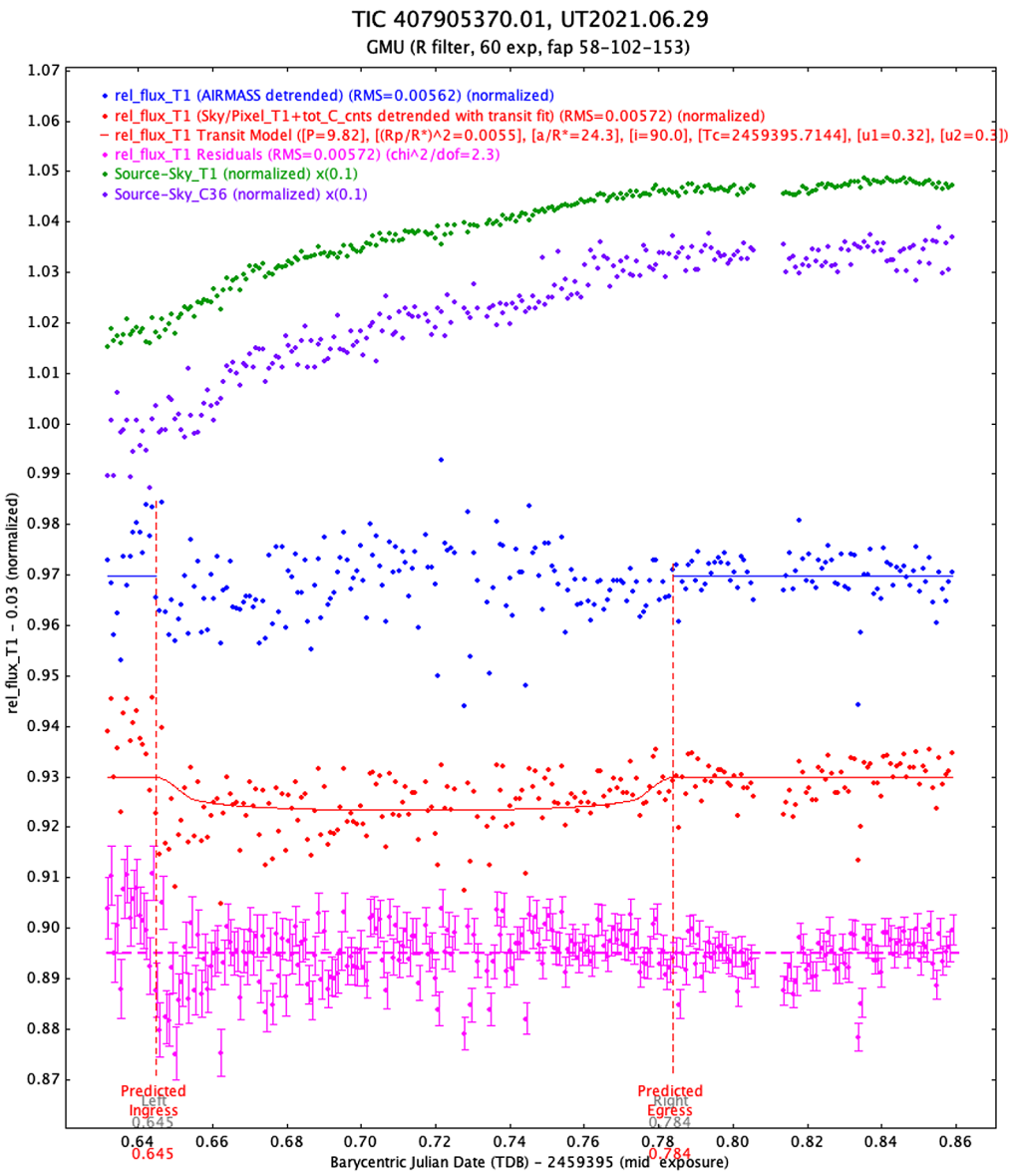

Fig. 17 This plot is produced with AIJ and follows the same conventions as in Figs. 8 and 9. We plot the light curve from an automated observation of TOI 3573.01 on UT on June 29, 2021. The full transit was collected, and we obtain best-fit values for $(R_p/R_*)^2 = 0.0055$ and $a/R_* = 24.3$, with a $\chi^2_{\text{red}} = 2.3$. The corresponding TESS candidate values are $(R_p/R_*)^2 = 0.0065$ and $a/R_* = 12.95$. On this particularly night, the seeing FWHM steadily improves through the night as well as the photometric precision, and this is likely due to the atmospheric conditions rather than our temperature-based focus position model.

References

1. W. J. Borucki et al., "Kepler planet-detection mission: introduction and first results," *Science* **327**(5968), 977–980 (2010).
2. G. R. Ricker et al., "Transiting exoplanet survey satellite," *J. Astron. Telesc. Instrum. Syst.* **1**(1), 014003 (2014).
3. D. Nesvorný and A. Morbidelli, "Mass and orbit determination from transit timing variations of exoplanets," *Astrophys. J.* **688**, 636–646 (2008).
4. D. Huber et al., "A hot Saturn orbiting an oscillating late subgiant discovered by TESS," *Astron. J.* **157**, 245 (2019).

5. A. Jordán et al., "TOI-677b: a warm Jupiter (P = 11.2 days) on an eccentric orbit transiting a late F-type star," *Astron. J.* **159**, 145 (2020).
6. L. D. Nielsen et al., "A Jovian planet in an eccentric 11.5 day orbit around HD 1397 discovered by TESS," *Astron. Astrophys.* **623**, A100 (2019).
7. E. A. Gilbert et al., "The first habitable-zone Earth-sized planet from TESS. I. Validation of the TOI-700 system," *Astron. J.* **160**, 116 (2020).
8. R. L. Akeson et al., "The NASA exoplanet archive: data and tools for exoplanet research," *Publ. Astron. Soc. Pac.* **125**, 989–999 (2013).
9. K. A. Collins et al., "The KELT follow-up network and transit false-positive catalog: pre-vetted false positives for TESS," *Astron. J.* **156**, 234 (2018).
10. T. D. Morton and J. A. Johnson, "On the low false positive probabilities of Kepler planet candidates," *Astrophys. J.* **738**, 170 (2011).
11. T. Barclay, J. Pepper, and E. V. Quintana, "A revised exoplanet yield from the Transiting Exoplanet Survey Satellite (TESS)," *Astrophys. J. Suppl. Ser.* **239**, 2 (2018).
12. P. W. Sullivan et al., "The Transiting Exoplanet Survey Satellite: simulations of planet detections and astrophysical false positives," *Astrophys. J.* **809**, 77 (2015).
13. P. Nutzman and D. Charbonneau, "Design considerations for a ground-based transit search for habitable planets orbiting M dwarfs," *Publ. Astron. Soc. Pac.* **120**, 317–327 (2008).
14. J. Pepper et al., "The Kilodegree Extremely Little Telescope (KELT): a small robotic telescope for large-area synoptic surveys," *Publ. Astron. Soc. Pac.* **119**, 923–935 (2007).
15. G. Bakos et al., "Wide-field millimagnitude photometry with the HAT: a tool for extrasolar planet detection," *Publ. Astron. Soc. Pac.* **116**, 266–277 (2004).
16. R. Alonso et al., "TrES-1: the transiting planet of a bright K0 V star," *Astrophys. J.* **613**, L153–L156 (2004).
17. D. Pollacco et al., "The WASP project and the SuperWASP cameras," *Publ. Astron. Soc. Pac.* **118**, 1407–1418 (2006).
18. J. Eastman et al., "DEMONEX: the DEdicated MONitor of EXotransits," *Proc. SPIE* **7733**, 77333J (2010).
19. C. Baranec et al., "Robo-AO: autonomous and replicable laser-adaptive-optics and science system," *Proc. SPIE* **8447**, 844704 (2012).
20. S. B. Cenko et al., "The automated Palomar 60 inch telescope," *Publ. Astron. Soc. Pac.* **118**, 1396–1406 (2006).
21. M. Ghachoui et al., "Using the OWL@OUKA telescope to follow-up the TESS planet candidates: first results," *Proc. SPIE* **11447**, 114479Z (2020).
22. J. J. Swift et al., "Miniature exoplanet radial velocity array I: design, commissioning, and early photometric results," *J. Astron. Telesc. Instrum. Syst.* **1**, 027002 (2015).
23. B. Addison et al., "Minerva-Australis. I. Design, commissioning, and first photometric results," *Publ. Astron. Soc. Pac.* **131**, 115003 (2019).
24. S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: a structure for efficient numerical computation," *Comput. Sci. Eng.* **13**, 22–30 (2011).
25. S. K. Lam, A. Pitrou, and S. Seibert, "Numba: a LLVM-based python JIT compiler," in *Proc. Second Workshop LLVM Compiler Infrastruct. in HPC, LLVM '15*, Association for Computing Machinery, New York (2015).
26. J. Schindelin et al., "Git for windows," https://gitforwindows.org/ (2022).
27. NASA Exoplanet Science Institute, "Exoplanet Follow-up Observing – - TESS," https://exofop.ipac.caltech.edu/ (2022).
28. Technical Innovations, "RoboFocus," http://www.robofocus.com/ (2013).
29. I. Helmy et al., "Autofocusing optimal search algorithm for a telescope system," *J. Astron. Instrument.* **10**(3) (2021).
30. M. Zhao et al., "Detection of Ks band thermal emission from WASP-3b," *Astrophys. J. Lett.* **748**, L8 (2012).
31. J. A. Johnson et al., "A smaller radius for the transiting Exoplanet WASP-10b," *Astrophys. J. Lett.* **692**, L100–L104 (2009).
32. G. Stefansson et al., "Toward space-like photometric precision from the ground with beam-shaping diffusers," *Astrophys. J.* **848**, 9 (2017).

33. J. Rayner et al., "iSHELL: a construction, assembly and testing," *Proc. SPIE* **9908**, 990884 (2016).
34. K. Collins and J. Kielkopf, "AstroImageJ: ImageJ for astronomy," Astrophysics Source Code Library (2013).
35. George Mason University, "David weather station," http://weather.cos.gmu.edu/ (2022).
36. The Weather Company, "Weather," http://weather.com (2022).
37. Space Science and Engineering Center, "Geostationary satellite imagery," http://ssec.wisc.edu/data/geo/ (2022).
38. K. L. Hay et al., "WASP-92b, WASP-93b and WASP-118b: three new transiting close-in giant planets," *Mon. Not. R. Astron. Soc.* **463**, 3276–3289 (2016).
39. Gaia Collaboration et al., "The Gaia mission," *Astron. Astrophys.* **595**, A1 (2016).
40. J. Eastman, B. S. Gaudi, and E. Agol, "EXOFAST: a fast exoplanetary fitting suite in IDL," *Publ. Astron. Soc. Pac.* **125**, 83–112 (2013).
41. J. D. Eastman et al., "EXOFASTv2: a public, generalized, publication-quality exoplanet modeling code," (2019).
42. K. G. Stassun et al., "The revised TESS Input Catalog and candidate target list," *Astron. J.* **158**, 138 (2019).
43. W. D. Pence, R. Seaman, and R. L. White, "Lossless astronomical image compression and the effects of noise," *Publ. Astron. Soc. Pac.* **121**, 414 (2009).
44. C. Blake, "The Princeton variability survey," *Publ. Astron. Soc. Pac.* **115**, 104–112 (2003).
45. A. D. Baker, C. H. Blake, and D. H. Sliski, "Monitoring telluric absorption with CAMAL," *Publ. Astron. Soc. Pac.* **129**, 085002 (2017).
46. K. Lanclos et al., "Key software architecture decisions for the automated planet finder," *Proc. SPIE* **9913**, 1533–1548 (2016).
47. S. S. Vogt et al., "APF—the lick observatory automated planet finder," *Publ. Astron. Soc. Pac.* **126**, 359–379 (2014).
48. DC-3 Dreams, SP, "ACP expert," https://acpx.dc3.com/ (2019).
49. Diffraction Limited, "MaxIm DL," https://diffractionlimited.com/product/maxim-dl/ (2022).

Michael Reefe is a student who recently completed his undergraduate degree in physics (with a concentration in astrophysics) at George Mason University (GMU). In addition to his work automating the GMU telescope and performing TESS follow-up analyses, he has also studied coronal emission lines and their relationships with active galactic nuclei and their host galaxies. He will continue his academic interests in astrophysics in graduate school at the Massachusetts Institute of Technology.

Biographies of the other authors are not available.