CEAS Space Journal
https://doi.org/10.1007/s12567-025-00623-w

ORIGINAL PAPER

SMARTies: a software suite for flexible and fully automated control of multi-sensor telescope stations

Christoph Bergmann[1] · Johannes Herzog[1] · Benjamin Hofmann[1] · Marcel Prohaska[2] · Yonathan Ascanio Hecker[1] · Hauke Fiedler[1] · Thomas Schildknecht[2] · Lucia Kleint[2]

Received: 10 October 2024 / Revised: 9 April 2025 / Accepted: 29 April 2025

Abstract
We describe the SMARTnet Instrument Enhancing Software (SMARTies) package, which we developed from scratch for fully automated remote control of our telescope stations within the Small Aperture Robotic Telescope Network (SMARTnet). Since March 2024, we have been using SMARTies continually to operate our SMARTnet telescope station in Chile successfully. We include a detailed description of the system design and architecture including the SMARTies modules, which are written in pure `Python` and are kept rather abstract, as well as the bespoke device controllers, which facilitate the communication with the actual hardware devices used. SMARTies was designed to fulfil a number of different use cases, including satellite or space debris tracking, survey observations, and observations of astronomical objects, including light curve acquisitions, and is therefore useful for the Space Situational Awareness community and other observational astronomers alike. While a fully automated observing mode following a user-defined schedule was one of the driving factors in the development of SMARTies, it does allow for near real-time manipulations of the schedule and even completely manual operations of the telescope. Because of its object-oriented and modular approach, new SMARTies functionalities can easily be added and different hardware devices can easily be included by adding new device controllers. Therefore, we envision SMARTies to be an extremely useful asset for many telescope operators across the world.

Keywords Telescopes · Observational methods · Software · Space situational awareness

1 Introduction

As the number of both satellites and space debris objects in Earth's orbit is rapidly increasing [1], more and more effort is put into monitoring and tracking this population of resident space objects (RSOs). One orbital region of particular interest is the one containing geosynchronous Earth orbits (GEOs) including the geostationary ring and its surroundings. This region is hardly accessible by radar observations but is usually monitored with passive-optical observations using telescopes. Its object density has also risen steeply over the past few years, as is the case for virtually all other orbital regions [1].

✉ Christoph Bergmann
christoph.bergmann@dlr.de

1 German Space Operations Center, German Aerospace Center, Münchener Str. 20, 82234 Weßling, Germany

2 Astronomical Institute, University of Bern, Sidlerstr. 5, 3012 Bern, Switzerland

A number of satellite and space debris monitoring / tracking campaigns exist that are partially or mainly dedicated to the GEO region (e.g., [2–5]). In collaboration with the Astronomical Institute of the University of Bern (AIUB), the German Space Operations Center (GSOC) has set up the SMARTnet for detecting, tracking, and monitoring RSOs at high altitudes [6]. As part of SMARTnet, GSOC operates several passive-optical telescope stations, currently SMART-01-SUTH at Sutherland, South Africa, SMART-02-KENT at Mt Kent, Australia, and SMART-03-ELSA at El Sauce, Chile, and will add more stations in the coming years. These are all robotic telescope stations at different time zones, and as such they need a reliable control software for fully automated operations without the need for a night observer.

Furthermore, the control software needs to satisfy all the different use cases we defined for our telescope stations, as further detailed in Sect. 2.2. For this reason, many observatories around the world use custom-made in-house software to operate their respective telescopes and instruments.

Unfortunately, this software often remains unpublished and near impossible to obtain and adapt, mainly because it is very hardware-specific. However, while there are a number of software suites available (e.g., [7–10]), they cannot be fully adapted to our purposes and/or lack one or more of the required functionalities. For example, the functionality to include multiple telescopes and/or sensors in the observation schedule, the ability to perform ephemeris tracking, or a dedicated high-cadence light curve acquisition mode is often missing.

We thus decided to build a new software system called SMARTies (SMARTnet Instrument Enhancing Software) completely from scratch, which is guaranteed to meet all requirements and satisfy all use cases we identified for our purposes, which can be lean as it does not have to carry a lot of legacy code, and which is made as simple as possible to use. At the same time, we aimed to make SMARTies a useful asset for other users with a wide variety of desired applications, from space situational awareness (SSA) applications like satellite and space debris tracking with different tracking modes to general astronomical observations whether they are astrometric, photometric, or spectroscopic in nature. Due to its object-oriented and modular approach, SMARTies can easily be adapted to different mounts, telescopes, sensors, and other devices, as well as for applications that we have not yet foreseen.

We describe the design concept and structure of the SMARTies software in Sect. 2, together with an overview of the most important use cases, the network communication structure, the device controllers, and the error handling. The individual `Python` modules are explained in Sect. 3. In Sect. 4, we give some examples on how to use SMARTies. Finally, Sect. 5 gives a summary and a brief outlook on planned future activities.

2 Concept and structure

2.1 Software development

Over the past couple of decades, `Python` has become the de facto standard programming language in astronomy and related fields [11, 12], and many extremely useful libraries are available to the scientific community, e.g., `scipy`, `astropy`, `numpy`. This makes software development significantly more convenient and efficient, although one aspect of our design philosophy was to use third party packages as little as possible for the sake of code transparency. Moreover, given its prevalence and the existing knowledge in our team of developers, we decided to choose `Python` as the programming language for SMARTies and to adhere to established software engineering guidelines. These include implementing unit tests wherever possible and applying a stringent code review policy, so that every line of code has passed through at least two pairs of eyes. For version control, issue tracking, mutual code review, and automated unit tests, we have been using GitLab. We also put considerable effort into various forms of documentation, including numerous in-code comments, verbose docstrings, and various interface control documents (ICDs).

For maximum flexibility, we decided to make SMARTies independent of the operating system used. Moreover, the vast majority of all source code in the SMARTies `Python` modules is written in an object-oriented manner and makes use of class inheritance. This fits well to the decision for a very modular approach, which we adopted in order to make any changes in hardware as simple as possible.

As an example, if a telescope station operator has different cameras, possibly even from different manufacturers, with different functionalities, they may need a separate `Python` class for each camera type to account for these differences, e.g., one might have a CCD (charge-coupled device) sensor with a mechanical iris shutter, whereas another one might have a sCMOS (scientific complementary metal-oxide-semiconductor) sensor with an electronic rolling shutter, but all of them are child classes of the same parent camera class. This way, the overlapping functionality is already covered without the need to implement it twice. If two hardware devices offer the same general functionalities but only differ in, for example, their allowed temperature range, range of motion, or number of pixels, there is no need for a separate `Python` class and it is enough to provide different settings in the respective ini-files (see below).

SMARTies is designed to be configured by a small number of ini-files, which provide hardware-specific settings and also tell SMARTies which devices are available. For the sake of simplicity, we use the standard Windows ini-file format, i.e., separated into sections each containing key-value pairs, so that users can easily adapt these files manually. In order to run SMARTies in automatic mode, a user must provide a schedule file in `JSON` (JavaScript Object Notation, [13]) format containing all *tasks* that are to be performed in a given observing night (see Sects. 3.1 and 4). We use a simple script to ensure the validity of these schedule files before feeding them into SMARTies. While it is noted that there are efforts by CEN/CENELEC to establish a standardized scheduling file format, called Scheduling and Command message (SCM) [14], we decided to adopt the `JSON` format for the sake of simplicity, flexibility, human-readability, and its widespread use not only in the scientific community but also in software development and (software) engineering in general. Also, note that SMARTies will attempt to perform all *tasks* and will not assess the reasonability of the commanded actions. Indeed, this is the desired behaviour so as not to limit the users' options. It is also worth pointing out that several telescopes and/or sensors can be controlled by

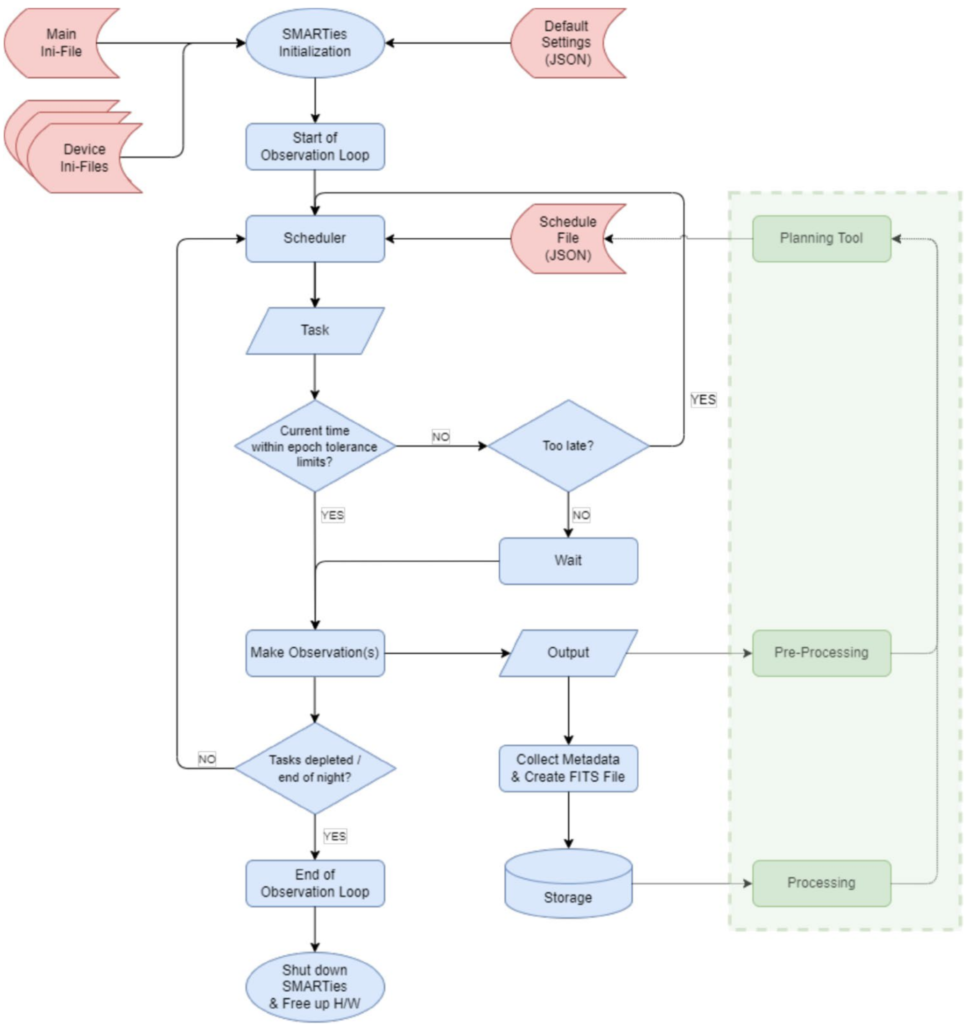

Fig. 1 Simplified flowchart showing the main aspects of running SMARTies

one instance of SMARTies by simply indicating the desired instrumentation for a given *task* in the schedule file.

Figure 1 shows a simplified high-level flowchart for SMARTies. After SMARTies is initialized using a number of input files, the main observation loop is started. The scheduler reads the schedule file and creates individual *tasks*. If a certain start epoch, possibly with tolerances, is requested in the *task*, SMARTies either performs the observations, waits until the requested epoch and then performs the observations, or, in case the requested time has already passed, moves on to the next *task*. If observations are performed, all corresponding metadata are collected and the data and metadata are written to a FITS (Flexible Image Transport System, [15]) file. The main observation loop continues until the end of the schedule file is reached, in which case SMARTies parks the telescope and shuts down properly. The three boxes in the shaded green area on the right side of the diagram indicate optional processing and pre-processing steps, which are not part of the SMARTies software. A user may perform such steps and, if, for example, the exposure time for a given object is found to be insufficient, may wish to adapt the schedule file.

In the absence of a hardware setup with which to test the source code, we found the concurrent development of the SMARTies python modules and the device controllers to be very challenging. Hence, in order to circumvent this problem to some extent during the early development stages of SMARTies, we worked with dummy device controllers, which simulate the network communication, and eventually allowed us to run mock end-to-end simulations of entire observing nights. This also helped us in ensuring that all output files (FITS files containing image data and metadata) are created correctly. After having put together a beta-version of SMARTies in mid-2023, we could eventually perform further intensive testing of the software with the actual hardware of SMART-03, which had been temporarily installed at Zimmerwald Observatory, Switzerland, by that time. This allowed us to test thoroughly all aspects of the SMARTies functionality. We began by testing all commands individually for each decoupled hardware device before eventually conducting observations in fully automated observing mode for several nights using schedule files. In particular, this included all tracking modes (azimuth-elevation (AZELE) tracking, sidereal tracking, angular-velocity tracking, and ephemeris tracking) and all different types of observation (bias frames, dark frames, flat fields, focus series, and science frames). We performed these scheduled observations with a multitude of combinations of the settings adjustable for a given *task* (see Sect. 4 for some examples). Please note that many of these settings, like exposure time, telescope pointing angles, or on-chip binning, can change from one exposure to the next within a given series. As might be expected, these tests under real-world conditions uncovered a number of issues and unexpected behaviour, especially in the network communication between the SMARTies modules, the intermediary software we call device controllers (see Sect. 2.3.2), and the hardware devices, and thus was an important step in quickly pushing SMARTies towards a stable version.

We reached a fully operational version 1.0 of SMARTies in September 2023. Since then, our focus has shifted from developing and debugging towards implementing additional features that extend beyond the SMARTies core functionalities. The most important additional feature, which we have added in v1.3.0 (July 2024), has been the ability to run SMARTies in light curve acquisition mode. We implemented this new mode in order to unlock the full potential of sCMOS cameras, which usually have much higher frame rates compared to CCDs, therefore enabling high-cadence image acquisitions needed for light curve analyses of fast rotating objects. All settings that may vary for individual exposures within a given series in regular observing mode are kept fixed in light curve acquisition mode in order to maximize the image acquisition rate.

While there will be further developments and minor bug fixes as part of a continual development process, the reliability of SMARTies has now been proven through many months of successful operations at SMART-03-ELSA, during which SMARTies has not caused a single failure.

2.2 Use cases

Before we began the programming phase, we defined a number of use cases for SMARTies and derived software requirements from them. The use cases we defined for SMARTies are naturally tailored to SSA purposes, i.e., anything related to satellite and space debris tracking and survey observations, as this is our main line of work. They can be divided into three categories: taking calibration frames, survey observations, and follow-up observations of known RSOs. Firstly, use cases related to calibration frames include taking bias frames, dark frames, flat fields, focus series, and images used for creating mount pointing models or mapping models. Secondly, the survey-observation use cases comprise scanning fields fixed in azimuth (AZI) and elevation (ELE) and fixed stellar fields, i.e., with the mount performing sidereal tracking. Finally, the use cases for follow-up observations include ephemeris tracking of satellites, space debris objects (including fragmentation events), Solar System objects, as well as tracking objects with fixed rates in the angles used by the mount (either right ascension (RA) and declination (DEC) or AZI and ELE). This includes a mode we call pseudo-ephemeris tracking, where the telescope follows any user-defined path on sky provided in the form of ephemerides, regardless of whether this corresponds to the motion of an object. On-sky observations may be performed in “standard” observation mode or in light curve mode, which is specifically designed to obtain high-cadence observations for photometry, including the use of different filters. However, please note that the applicability of SMARTies is by no means limited to these use cases and that use cases defined by other users may well be covered by the functionality of SMARTies already. In fact, we expect this to be the case for the majority of possible telescope operating modes an observer may wish to employ.

2.3 Communication

For each of the hardware-controlling modules described in Sect. 3.2, there is at least one corresponding device controller, which acts as an intermediary between the hardware-controlling module and the device firmware (see Sect. 2.3.2). The main reason for this additional abstraction layer is to make SMARTies independent of the hardware devices used. For example, some manufacturers require a specific operating system for their device drivers or software development kits (SDKs), so a device controller may run on a Windows computer while SMARTies runs on a Linux machine or vice versa. Note that these computers may be physically separated, perhaps even in a different building, as is the case for our SMARTnet telescope stations. SMARTies only needs to take care of the network communication to the device controller via TCP/IP (Transmission Control Protocol / Internet Protocol). When a hardware device is changed or replaced, a new device controller may become necessary as different hardware devices can have different functionalities and/or may use a different communication protocol (e.g., classic ASCOM [16, 17], ASCOM Alpaca [18], INDI[1], or a manufacturer’s own protocol). Hence, in the example given in Sect. 2.1, each camera may require a separate device controller. However, no changes to SMARTies would be required.

The interplay between the hardware-controlling modules, the corresponding device controllers, and the physical hardware devices is illustrated in Fig. 2. The TCP/IP communication

[1] https://indilib.org/. last accessed 09 Sep 2024.

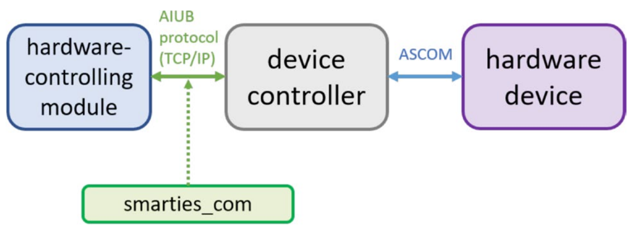

Fig. 2 Abstract depiction of the communication scheme between hardware-controlling modules, device controllers, and hardware devices

between the hardware-controlling module and the device controller is handled by the communication-handling module *smarties_com* using the AIUB protocol (see Sects. 2.3.1 and 3.3), and the communication between the device controller and the physical hardware device is handled by the device controller using the communication protocol dictated by the manufacturer (ASCOM in this example).

In summary, the modular approach and the abstraction of the communication between software and hardware minimize the effort required to adapt SMARTies to new hardware devices.

2.3.1 AIUB protocol

For network communications, we decided to use the AIUB protocol [19], which has been proven reliable over many years of telescope station operations. This decision saved us the time to develop a protocol of our own. The AIUB protocol defines a network-based exchange of requests and replies between a requester and an executor. Communication takes place in plain text and can be used for both synchronous and asynchronous applications. The basic structure of a record is as follows:

```
$SOR,<length>,<number of messages>,<list
of messages>,$EOR
```

where:

`$SOR` Start of record.

`<length>` The number of characters of the entire record.

`<number of messages>` The number of messages contained in the record.

`<list of messages>` The comma-separated list of all messages.

`$EOR` End of record.

The message itself is constructed accordingly:

```
%$MESSAGE_ID,<length>,<number of param-
eters>, <parameters>,$EOM
```

where:

`$MESSAGE_ID` The pre-defined identifier of the message.

`<length>` The number of characters of the entire message.

`<number of parameters>` The number of parameters contained in the message.

`<parameters>` The comma-separated list of all per message pre-defined parameters.

`$EOM` End of Message.

2.3.2 Device controllers

As mentioned above, for every hardware device used (e.g., mount, camera, focuser, weather station, filter wheel, derotator,...) there is a device controller, which acts as an intermediary between the hardware-controlling SMARTies modules described in Sect. 3.2 and the device firmware. A device controller relays commands received from the SMARTies module to the hardware device itself and passes on any responses, outcomes, and results of the commanded actions to the commanding hardware-controlling SMARTies module. Communication between the hardware-controlling SMARTies modules and the device controllers is achieved through the communication-handling *smarties_com* module by sending back and forth a byte stream via a TCP/IP connection. For this, we adopted the use of the AIUB protocol described above. Please note that all users can add their own bespoke device controllers as long as they follow the well-defined interface described in the SMARTies ICD.

Using the AIUB protocol, a generalized division of the tasks to be completed in the interaction between SMARTies and any device results can be summarized as shown in Fig. 3. Within SMARTies, the topmost box labeled is a hardware-controlling module like the *camera* module or the *mount* module. The other two boxes are part of the *smarties_com* module, which handles the AIUB protocol syntax and the TCP/IP connection.

The structure plan for an entire example telescope station is shown in Fig. 4. This setup is very similar to the one at SMART-03-ELSA, although we have removed the second camera and focuser for simplicity. Here, the core hardware devices include a mount, a camera, and a focuser. For each of them there is a hardware-controlling module and a device controller. Note that a dome controller is not necessary in this setup, as this is controlled by a separate station monitoring and control software. The *mount* module also communicates with the flipflop device. In addition, there is a weather station providing meteorological data and a time card acting as an event recorder for precise timing. In this example, the communication between the mount and the focuser with their respective device controllers works via ASCOM Alpaca (cyan boxes), with an additional Alpaca server in between, which is

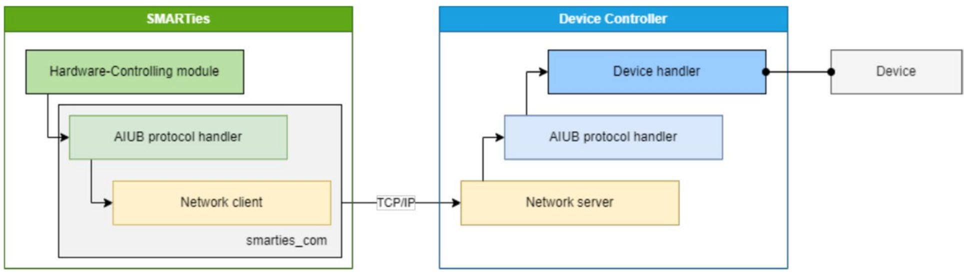

Fig. 3 Basic principle of the collaboration between SMARTies, a device controller, and the associated device

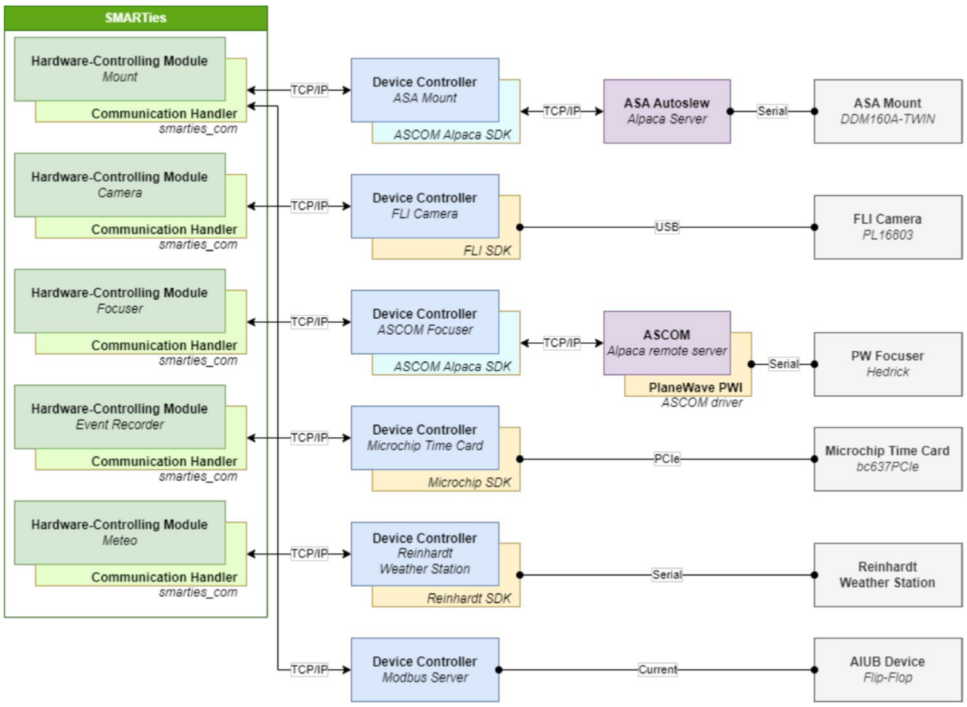

Fig. 4 Structure plan for the interaction of SMARTies, some device controllers, and the associated devices based on an example of an entire telescope station

provided by the manufacturer. In contrast, the manufacturers of the camera, weather station, and time card provide their own software communication protocol, indicated by the yellow boxes beneath the respective device controllers.

2.4 Error handling

In order to ensure smooth and stable operations of a telescope station, it is imperative to handle any potential errors that may arise with great care. Thus, all SMARTies modules strictly adhere to the same error handling. Every function or method call always returns an outcome, even if no other values are returned. This outcome can either be “no error”, or one or several well-defined error codes, which are also logged. These error codes are always relayed up the calling hierarchy, sometimes through several layers, until eventually they are passed back to the *main* module. It is only at this stage that a decision is made on how to deal with a given error, depending on their severity. SMARTies may simply jump to the next exposure, e.g., when camera settings could not be set or the FITS file could not be written, or it may jump to the next *task*, e.g., when no connection could be established to the mount. Experience has taught us that it may pay to keep on trying in these cases. Only as a last resort is SMARTies terminated, which happens, for instance, if no camera is available upon initializing SMARTies.

3 SMARTies python modules

Almost all SMARTies modules are written in an object-oriented way. This comes in particularly handy for the hardware-controlling modules, in which we define an abstract base class, from which hardware-specific child classes inherit their core functionalities. Any modifications from or additions to the core functionalities can therefore easily be realized within the child class.

The SMARTies `Python` modules can be divided into four categories: organizational (*main*, *scheduler*), hardware-controlling (*camera*, *mount*, *focuser*, *timing*, *meteo*), communication-handling (*smarties_com*), and auxiliary modules (*conversion*, *format_checker*, *file_handler*, *series_evaluation_tool*). Note that, depending on the setup of the telescope station, there may be additional hardware-controlling modules, for example, for a filter wheel / stage, a derotator, or a dome with a slit. They are not explained here in detail as the existing SMARTnet stations do not have any additional hardware (yet), but they must at least have getter and setter methods in principle, similar to the *focuser* module.

3.1 Organizational modules

3.1.1 Scheduler

The *scheduler* module reads the user-provided schedule file and breaks it down into *tasks*, which are yielded to the *main* module for execution. These *tasks* are series of N_{obs} individual exposures of a given exposure type (bias, dark, light), for which certain settings can be specified. As of SMARTies version 1.3.0, a *task* can also define a series of light curve observations consisting of N_{obs} individual exposures. The adjustable settings include general settings like the coordinate system and tracking mode as well as sensor-specific settings like chip temperature including tolerances, exposure time, focus position, binning, sub-frame windows, etc. There are default settings for all exposure types, which include the number of exposures per series, the exposure time, and camera settings like temperature, binning, or window size. They are overwritten for a given *task* by the user-defined settings if the specific keywords are present in the schedule file. Optionally, the user can also provide a specific start epoch including tolerances, the time between consecutive observations including tolerances, and exclusion windows. After each *task*, the scheduler also checks whether a new schedule file has been created by the user, thereby allowing for dynamic, near real-time scheduling changes should the need arise.

3.1.2 Main module

The main workflow is handled by the *main* module. After SMARTies has been initialized, i.e., all housekeeping tasks are completed and all hardware-specific class instances are created according to their respective ini-files, the scheduler reads the user-provided schedule file and feeds *tasks* to the *main* module, which then delegates smaller work packages required to fulfil these *tasks* to the other modules. These work packages include applying a number of hardware settings, performing coordinate transformations and starting tracking, handling exposure timing, acquiring and reading out an image, collecting all metadata, and saving all data and metadata to a file. *Tasks* are always conducted in the order in which they appear in the schedule file. However, it is possible to change the schedule in near real-time by providing a new schedule file. Once all *tasks* are completed, the telescope is parked and SMARTies is shut down.

3.2 Hardware-controlling modules

The hardware-controlling modules are in charge of controlling the hardware devices. When a hardware-controlling module sends a command to perform some action or to read the status of a device, the corresponding device controller relays the command to the actual hardware device and also relays the hardware response, e.g., a temperature, a tracking status, or an error message, back to the hardware-controlling module. They are included as class instances within the *main* module and are configured with a hardware-specific ini-file upon instantiation. Note that there may be multiple hardware devices of the same kind, e.g., two telescopes on a twin mount, and thus two cameras and two focusers, in which case one class instance is used for each hardware device. Communication with the device controllers is realized via a TCP/IP connection and is handled by a dedicated communication-handling module described in Sect. 3.3.

3.2.1 Camera module

The *camera* module is in charge of all camera-related activities. The main tasks of the *camera* module are setting and getting the camera temperature, selecting the region of the chip to be used for an exposure (full frame or sub-frame), changing the camera binning and/or read-out speed, and, of course, acquiring an image with a user-defined exposure time and reading out the camera buffer. We have implemented two separate camera classes for CCD and sCMOS cameras, respectively, which both inherit their core functionality from a common parent camera class.

3.2.2 Mount module

The *mount* module handles commands related to mount and hence telescope movement. It is mainly used for moving the mount to a user-defined position (given as either RA / DEC or AZI / ELE) and starting tracking. Several tracking modes can be employed, namely AZELE tracking (i.e., point and hold), sidereal tracking, angular-velocity tracking (with constant user-defined angular velocities in the two reference angles, e.g., RA / DEC for equatorial mounts), and ephemeris tracking, which is realized by providing an external file containing ephemerides. In addition, the *mount* module also retrieves the current mount status and relays meteorological data to the mount, which is needed to calculate refraction in case the mount provides such functionality. The meteorological data itself is provided by the *meteo* module, see below. Finally, it retrieves information about the revolution the mount is currently in from a flipflop sensor, if present.

3.2.3 Focuser module

The sole purpose of the *focuser* module is to control the focuser, i.e., to set and get focus positions. Note that the determination of the optimal focus position is not handled by the *focuser* module but rather by the *series_evaluation_tool* module.

3.2.4 Timing module

The *timing* module can serve as an event recorder for a specific timing solution, i.e., it can be used to retrieve the shutter opening and closing times to better than millisecond accuracy based on a global navigation satellite system (GNSS) signal. For that to work, GNSS-synced time cards must be installed in the server computer(s) connected to the camera(s). In order to measure these epochs, an electronic implementation of detecting and reporting signal changes must be realized in the camera(s). The *timing* module sends a command to prepare the time cards before each observation and retrieves the recorded epochs after the exposure has finished.

3.2.5 Meteo module

Strictly speaking, the *meteo* module is not really controlling a hardware device, as it only has a getter function used to retrieve meteorological data from a weather station, if available.

3.3 Communication-handling module

This module, called *smarties_com*, acts as the intermediary between the hardware-controlling modules and the corresponding device controllers. It handles the network communication via TCP/IP, translates the commands it receives from the hardware-controlling modules into the required syntax, and sends them to the respective device controllers. Upon receiving a reply from a device controller, it parses this reply and breaks it down to the level that the respective hardware-controlling module can digest before passing it on. Finally, it also facilitates the transfer of the actual image data as a byte stream from the camera controller to the *camera* module.

3.4 Auxiliary modules

The auxiliary modules do not communicate with any devices controllers but provide additional functionality and helper functions, which are used across multiple hardware-controlling modules and the *main* module.

3.4.1 Conversion modules

The *conversion* module is used to perform a variety of astronomical calculations. Most importantly, these include coordinate conversions between different reference systems (e.g., from J2000 to True of Date (TOD)), the conversion of a position on sky between different coordinate systems (e.g., from RA / DEC to AZI / ELE), the conversion of position vectors between Cartesian coordinates and geocentric/geodetic coordinates, and the calculation of diurnal and annual aberration corrections when converting between apparent and catalogue coordinates. It also contains a small library of other low-level auxiliary routines that perform various often needed calculations, such as an interpolation function or rotation matrices.

3.4.2 Format checker

The *format_checker* module is invoked by most higher-level routines in order to screen their input arguments for any violations of the expected format. It thereby increases code stability, facilitates proper error handling and logging, and reduces boiler-plate code. Additionally, it also serves to distinguish between scalar and vectorized inputs, so that performance can be enhanced by running vectorized operations where applicable.

3.4.3 File handler

The *file_handler* module is tasked with saving the image data together with all collected metadata to disk. It determines the appropriate filename and directory and stores the observations in FITS format [15]. In regular observing mode with CCD cameras, one FITS file contains one 2-dimensional image plus a single header. However, certain sCMOS cameras support multiple image channels with different gains, so a FITS file may contain multiple header data units (HDUs). Also, note that in light curve mode, all individual exposures constituting the light curve are stored as single HDUs within one large FITS file.

3.4.4 Series evaluation tool

This module, called *series_evaluation_tool*, is invoked by the *main* module every time a focus series is conducted in order to perform the necessary steps to determine the optimal focus position automatically. A focus series is a series of exposures of a field of stars with sidereal tracking enabled and with variable focus values. For each exposure, a 2-dimensional Gaussian is fitted to all suitable stars in the field of view and their widths are recorded. Subsequently, the optimal focus position is determined as the one minimizing these widths.

4 How to use SMARTies

SMARTies can be started from a command line and the name of the main ini-file must be given as an input argument. The main ini-file contains, amongst other things, the station coordinates and lists the filenames and directories of all input and output files. In particular, it references all hardware-specific ini-files, which define certain device properties and can be used to provide various settings, and the schedule file, which contains all *tasks* that have to be conducted in a given night.

In the following, we will give examples of some common *tasks* that can be performed with SMARTies, and how the respective *tasks* in the schedule file can be constructed.

4.1 Take calibration frames

```
{
    "00001": {
        "object": "Flatfield",
        "series": "0001",
        "epoch_start": "2024-08-29T21:30:00.000",
        "coord_system": "AZELE",
        "angle_1_deg": 90.0,
        "angle_2_deg": 15.0,
        "ref_frame": "TOD",
        "tracking_mode": "AZELE",
        "sensor": {
            "SMART-03-B-ELSA": {
                "settings": "Flat",
                "n_obs": "9",
                "exposure_time_sec": 0.5,
                "cool_temp": -25.0,
            }
        }
    },
    "00002": {
        .
        .
        .
    },
    .
    .
    .
}
```

Listing 1 Example schedule file containing a series of flatfield frames as the first task

Listing 1 shows an excerpt from a schedule file, indicating how the schedule file is comprised of several *tasks*. In this example, the first *task* commands SMARTies to take a series of flatfield frames starting at 21:30 on 29 Aug 2024 (UTC). For this *task*, the mount tracking is set to AZELE tracking at an azimuth angle of 90° and an elevation angle of 15°. The series consists of 9 individual exposures with an exposure time of 0.5 s, and the camera temperature is set to −25 °C.

4.2 Take a series of survey observations

```
{
    .
    .
    .
    "00042": {
        "object": "Survey",
        "series": "0013",
        "coord_system": "RADEC",
        "angle_1_deg": [120.0, 122.0, 124.0, 126.0, 128.0],
        "angle_2_deg": [-45.0, -44.0, -43.0, -42.0, -41.0],
        "ref_frame": "J2000",
        "tracking_mode": "AZELE",
        "sensor": {
            "SMART-03-B-ELSA": {
                "settings": "Default",
                "exposure_time_sec": [8.0, 10.0, 12.0, 14.0, 16.0],
                "delta_t_sec": 30.0,
                "x_binning": 2,
                "y_binning": 2,
            }
        }
    },
    .
    .
    .
}
```

Listing 2 Example schedule file containing a series of survey observations as the 42nd task

Another typical use case is to take a series of survey observations, as illustrated by Listing 2. While the settings used in this example might be questionable for a standard survey series, they highlight some of the flexibility that SMARTies offers its users. Here, the target coordinates are given as RA and DEC in the J2000 frame, AZELE tracking is performed rather than sidereal tracking. Here, the number of exposures is inferred from the length of the list of exposure times. Note that the target coordinates are also given as lists, so for each of the five individual exposures, the telescope will be pointing to a different location. Furthermore, all exposures will have a different exposure time as well, there will be 30 s between the starts of consecutive exposures, and the camera is set to a 2×2 binning mode.

4.3 Take a light curve series of an object

```
{
    .
    .
    .
    "00127": {
        "object": "123ABC",
        "series": "0001",
        "tracking_mode": "ephemeris",
        "ephemerides_file": "123ABC.eph",
        "sensor": {
            "SMART-03-A-ELSA": {
                "settings": "Lightcurve",
                "exposure_time_sec": 0.1,
                "duration": 120,
                "lower_left": [1848, 1848],
                "region": [400, 400]
            }
        }
    },
    .
    .
    .
}
```

Listing 3 Example schedule file containing a series of light curve observations as the 127th task

Listing 3 shows an example for a *task* commanding SMARTies to take a series of exposures in light curve mode. In this example, the mount performs ephemeris tracking according to a user-provided file containing the object's ephemerides. The exposure times for all exposures are set to 0.1 s and SMARTies will take as many consecutive exposures as possible for a duration of 120 s. Also, only a 400×400 pixel sub-frame will be used, which has coordinates $(x, y) = (1848, 1848)$ at its lower left corner. Using a sub-frame instead of the full frame speeds up the read-out process significantly and is therefore common practice for light curve observations of RSOs.

5 Summary and outlook

In order to fulfil all our needs and use cases for operating our telescope stations within SMARTnet, we have developed and implemented SMARTies. In particular, this software system allows for fully automated control of remote multi-sensor telescope stations while offering a high degree of flexibility.

After the completion of the initial code development phase and several months of rigorous testing and debugging under real-life conditions with the telescope station hardware installed at Zimmerwald Observatory, Switzerland, SMARTies has been in continual use successfully at our telescope station SMART-03-ELSA at El Sauce Observatory, Chile, since March 2024. Having performed many thousands of observations with SMARTies in different observing modes by now, we are very pleased with its performance and usability, allowing us both, to obtain standard observations in care-free automatic mode, as well as to perform special observations with a high degree of flexibility. Furthermore, the strict and fastidious error handling and logging has proven extremely valuable for code maintainability and improvement whenever non-critical errors have occurred. Yet, perhaps the most outstanding property might be its code stability, as SMARTies has not crashed once. One of the lessons learnt is the fact that despite using an external syntax checker for the schedule files, it is still possible, albeit unlikely, for SMARTies to get stuck if a user provides certain nonsensical settings in and/or combinations of the tasks in the schedule file. Users should thus be extra careful when creating their schedule files.

In the near future, we envision a number of enhancements to the functionality of SMARTies. In addition to measuring the on-sky positions of RSOs, we want to obtain photometric measurements to study their brightness variations. By analysing these brightness variations as a function of time, i.e., the objects' light curves, information about their rotational periods and axes can be revealed. This is also an important precursor activity for future active debris removal missions. For this purpose, we acquired an FLI Kepler KL4040 sCMOS camera, for which we have also developed a device controller, while our existing telescope stations are all equipped with FLI PL16803 CCD cameras. We plan to install the new sCMOS camera at

SMART-03-ELSA in the near future, so that we can obtain and analyse light curves of RSOs in the near future. Besides, any future telescope stations operated by GSOC will feature sCMOS cameras.

Furthermore, work has already commenced on designing and implementing a graphical user interface (GUI). While a GUI is not needed when SMARTies is run in fully automated mode, an interactive manual observing mode shall be possible, providing a high degree of convenience and flexibility.

In order to increase observing efficiency and minimize dead time, we are also working on an optimized workflow using asynchronous commands that do not block each other while being executed. For example, there is no need to wait for the camera to finish reading out an image, which can take minutes for large CCD cameras, before slewing the telescope to the next desired position.

Another future functionality of SMARTies will help to maximize the efficacy and thus unleash the full potential of multi-sensor telescope stations by operating all telescopes and/or sensors simultaneously, which may or may not be done in a synchronized way. For example, with two or more telescopes installed on the same mount, one sensor could conduct wide-field survey observations measurements, while another sensor could perform high-cadence measurements of a single target in the field of view for light curve analyses.

As mentioned above, the first telescope station to be operated using SMARTies is GSOC's third SMARTnet station SMART-03-ELSA, which was installed and commissioned in Chile in early 2024. In the future, we successively plan to switch over our other telescope stations in South Africa and Australia to running with SMARTies as well. Any future SMARTnet stations will also be using SMARTies.

Please note that this work describes the current version of SMARTies, and while the overall design and core functionalities will likely not change much, we expect that some implementation details or minor structural changes will occur in future versions.

Acknowledgements We would like to acknowledge the valuable support received from HIFIS Software Services. This research has made use of NASA's Astrophysics Data System. This work also made use of the following software packages: `Python` [20], `astropy` [11, 21, 22], `numpy` [23], and `scipy` [24, 25]. Software citation information aggregated using The Software Citation Station [26, 27]. Finally, we would like to thank the anonymous referees, whose insightful comments helped in improving this manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability Not applicable.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. ESA: ESA Space Environment Report 2024. https://www.esa.int/Space_Safety/Space_Debris/ESA_Space_Environment_Report_2024 (2024)
2. Boër, M., Klotz, A., Laugier, R., Richard, P., Dolado Perez, J.-C., Lapasset, L., Verzeni, A., Théron, S., Coward, D., Kennewell, J.A.: TAROT: A network for Space Surveillance and Tracking operations. In: 7th European Conference on Space Debris, p. 72 (2017)
3. Park, J.-H., Yim, H.-S., Choi, Y.-J., Jo, J.H., Moon, H.-K., Park, Y.-S., Bae, Y.-H., Park, S.-Y., Roh, D.-G., Cho, S., Choi, E.-J., Kim, M.-J., Choi, J.: OWL-Net: A global network of robotic telescopes for satellite observation. Adv. Space Res. **62**(1), 152–163 (2018). https://doi.org/10.1016/j.asr.2018.04.008
4. Blake, J.A., Chote, P., Pollacco, D., Feline, W., Privett, G., Ash, A., Eves, S., Greenwood, A., Harwood, N., Marsh, T.R., Veras, D., Watson, C.: DebrisWatch I: A survey of faint geosynchronous debris. Adv. Space Res. **67**(1), 360–370 (2021). https://doi.org/10.1016/j.asr.2020.08.008. arXiv:2008.12799 [astro-ph.EP]
5. Luo, H., Mao, Y.-D., Yu, Y., Tang, Z.-H.: FocusGEO observations of space debris at Geosynchronous Earth Orbit. Adv. Space Res. **64**(2), 465–474 (2019). https://doi.org/10.1016/j.asr.2019.04.006
6. Fiedler, H., Herzog, J., Ploner, M., Prohaska, M., Schildknecht, T., Weigel, M., Klabl, M.: SMARTnet – First Results of the Telescope Network. In: 7th European Conference on Space Debris, p. 97 (2017)
7. Husser, T.-O., Hessman, F.V., Martens, S., Masur, T., Royen, K., Schäfer, S.: pyobs - An Observatory Control System for Robotic Telescopes. Frontiers in Astronomy and Space Sciences **9**, 891486 (2022) https://doi.org/10.3389/fspas.2022.891486arXiv:2203.12642 [astro-ph.IM]
8. Zhang, C., Zhu, C.: CHES robotic observation software kit. Frontiers in Astronomy and Space Sciences **9**, 896570 (2022). https://doi.org/10.3389/fspas.2022.896570
9. Kouprianov, V., Molotov, I.: FORTE: ISON Robotic Telescope Control Software. In: 7th European Conference on Space Debris, p. 112 (2017)
10. Berg, S., N.I.N.A. contributors: N.I.N.A. - Nighttime Imaging 'N' Astronomy. https://nighttime-imaging.eu/docs/master/site/ (2024)
11. Astropy Collaboration, Price-Whelan, A.M., Lim, P.L., Earl, N., Starkman, N., Bradley, L., Shupe, D.L., Patil, A.A., Corrales, L., Brasseur, C.E., Nöthe, M., Donath, A., Tollerud, E., Morris, B.M., Ginsburg, A., Vaher, E., Weaver, B.A., Tocknell, J., Jamieson, W., van Kerkwijk, M.H., Robitaille, T.P., Merry, B., Bachetti, M., Günther, H.M., Aldcroft, T.L., Alvarado-Montes, J.A., Archibald, A.M., Bódi, A., Bapat, S., Barentsen, G., Bazán, J., Biswas, M., Boquien, M., Burke, D.J., Cara, D., Cara, M., Conroy, K.E., Conseil, S., Craig, M.W., Cross, R.M., Cruz, K.L., D'Eugenio, F., Dencheva, N., Devillepoix, H.A.R., Dietrich, J.P., Eigenbrot, A.D., Erben, T., Ferreira, L., Foreman-Mackey, D., Fox, R., Freij, N., Garg, S., Geda, R., Glattly, L., Gondhalekar, Y., Gordon, K.D., Grant, D., Greenfield, P., Groener,

A.M., Guest, S., Gurovich, S., Handberg, R., Hart, A., Hatfield-Dodds, Z., Homeier, D., Hosseinzadeh, G., Jenness, T., Jones, C.K., Joseph, P., Kalmbach, J.B., Karamehmetoglu, E., Kałuszyński, M., Kelley, M.S.P., Kern, N., Kerzendorf, W.E., Koch, E.W., Kulumani, S., Lee, A., Ly, C., Ma, Z., MacBride, C., Maljaars, J.M., Muna, D., Murphy, N.A., Norman, H., O'Steen, R., Oman, K.A., Pacifici, C., Pascual, S., Pascual-Granado, J., Patil, R.R., Perren, G.I., Pickering, T.E., Rastogi, T., Roulston, B.R., Ryan, D.F., Rykoff, E.S., Sabater, J., Sakurikar, P., Salgado, J., Sanghi, A., Saunders, N., Savchenko, V., Schwardt, L., Seifert-Eckert, M., Shih, A.Y., Jain, A.S., Shukla, G., Sick, J., Simpson, C., Singanamalla, S., Singer, L.P., Singhal, J., Sinha, M., Sipőcz, B.M., Spitler, L.R., Stansby, D., Streicher, O., Šumak, J., Swinbank, J.D., Taranu, D.S., Tewary, N., Tremblay, G.R., Val-Borro, M.d., Van Kooten, S.J., Vasović, Z., Verma, S., de Miranda Cardoso, J.V., Williams, P.K.G., Wilson, T.J., Winkel, B., Wood-Vasey, W.M., Xue, R., Yoachim, P., Zhang, C., Zonca, A., Astropy Project Contributors: The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package. ApJ **935** (2), 167 (2022) https://doi.org/10.3847/1538-4357/ac7c74arXiv:2206.14220 [astro-ph.IM]

12. Faes, D.M.: Use of Python programming language in astronomy and science. arXiv e-prints, 1807–04806 (2018) https://doi.org/10.48550/arXiv.1807.04806arXiv:1807.04806 [astro-ph.IM]
13. Pezoa, F., Reutter, J.L., Suarez, F., Ugarte, M., Vrgoč, D.: Foundations of json schema. In: Proceedings of the 25th International Conference on World Wide Web, pp. 263–273 (2016). International World Wide Web Conferences Steering Committee
14. Białkowski, A., Duźniak, P., Santana-Ros, T., Adamczyk, A., Taberski, G., Pieniowska, K., Baksalary, J., Renk, R., Matysiak, J., Kwiatkowski, T., Kamiński, K., Bartczak, P.: Scheduling and Commanding Message Standard usage in telescope tasking activities for NEO and SST. In: 1st NEO and Debris Detection Conference (ESA), p. 32 (2019)
15. Wells, D.C., Greisen, E.W., Harten, R.H.: FITS - a Flexible Image Transport System. A &AS **44**, 363 (1981)
16. George, D.B., Denny, R.: ASCOM - Progress In Technology And Applications. International Amateur-Professional Photoelectric Photometry Communications **84**, 16 (2001)
17. Denny, R.B.: Software Interoperation and Compatibility: ASCOM Update. Society for Astronomical Sciences Annual Symposium **21**, 39 (2002)
18. Denny, R.: ASCOM - Not Just for Windows Any More. In: Buchheim, R.K., Gill, R.M., Green, W., Martin, J.C., Menke, J., Stephens, R. (eds.) 38th Annual Conference of the Society for Astronomical Sciences (SAS-2019), p. 31 (2019)
19. Gurtner, W., Pop, E., Utzinger, J.: Automation and Remote Control of the Zimmerwald SLR Station. In: 11th International Workshop on Laser Ranging (1998)
20. Van Rossum, G., Drake, F.L.: Python 3 Reference Manual. CreateSpace, Scotts Valley, CA (2009)
21. Astropy Collaboration, Robitaille, T.P., Tollerud, E.J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A.M., Kerzendorf, W.E., Conley, A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M.M., Nair, P.H., Unther, H.M., Deil, C., Woillez, J., Conseil, S., Kramer, R., Turner, J.E.H., Singer, L., Fox, R., Weaver, B.A., Zabalza, V., Edwards, Z.I., Azalee Bostroem, K., Burke, D.J., Casey, A.R., Crawford, S.M., Dencheva, N., Ely, J., Jenness, T., Labrie, K., Lim, P.L., Pierfederici, F., Pontzen, A., Ptak, A., Refsdal, B., Servillat, M., Streicher, O.: Astropy: A community Python package for astronomy. A &A **558**, 33 (2013) https://doi.org/10.1051/0004-6361/201322068arXiv:1307.6212 [astro-ph.IM]
22. Astropy Collaboration, Price-Whelan, A.M., Sipőcz, B.M., Günther, H.M., Lim, P.L., Crawford, S.M., Conseil, S., Shupe, D.L., Craig, M.W., Dencheva, N., Ginsburg, A., VanderPlas, J.T., Bradley, L.D., Pérez-Suárez, D., de Val-Borro, M., Aldcroft, T.L., Cruz, K.L., Robitaille, T.P., Tollerud, E.J., Ardelean, C., Babej, T., Bach, Y.P., Bachetti, M., Bakanov, A.V., Bamford, S.P., Barentsen, G., Barmby, P., Baumbach, A., Berry, K.L., Biscani, F., Boquien, M., Bostroem, K.A., Bouma, L.G., Brammer, G.B., Bray, E.M., Breytenbach, H., Buddelmeijer, H., Burke, D.J., Calderone, G., Cano Rodríguez, J.L., Cara, M., Cardoso, J.V.M., Cheedella, S., Copin, Y., Corrales, L., Crichton, D., D'Avella, D., Deil, C., Depagne, É., Dietrich, J.P., Donath, A., Droettboom, M., Earl, N., Erben, T., Fabbro, S., Ferreira, L.A., Finethy, T., Fox, R.T., Garrison, L.H., Gibbons, S.L.J., Goldstein, D.A., Gommers, R., Greco, J.P., Greenfield, P., Groener, A.M., Grollier, F., Hagen, A., Hirst, P., Homeier, D., Horton, A.J., Hosseinzadeh, G., Hu, L., Hunkeler, J.S., Ivezić, Ž., Jain, A., Jenness, T., Kanarek, G., Kendrew, S., Kern, N.S., Kerzendorf, W.E., Khvalko, A., King, J., Kirkby, D., Kulkarni, A.M., Kumar, A., Lee, A., Lenz, D., Littlefair, S.P., Ma, Z., Macleod, D.M., Mastropietro, M., McCully, C., Montagnac, S., Morris, B.M., Mueller, M., Mumford, S.J., Muna, D., Murphy, N.A., Nelson, S., Nguyen, G.H., Ninan, J.P., Nöthe, M., Ogaz, S., Oh, S., Parejko, J.K., Parley, N., Pascual, S., Patil, R., Patil, A.A., Plunkett, A.L., Prochaska, J.X., Rastogi, T., Reddy Janga, V., Sabater, J., Sakurikar, P., Seifert, M., Sherbert, L.E., Sherwood-Taylor, H., Shih, A.Y., Sick, J., Silbiger, M.T., Singanamalla, S., Singer, L.P., Sladen, P.H., Sooley, K.A., Sornarajah, S., Streicher, O., Teuben, P., Thomas, S.W., Tremblay, G.R., Turner, J.E.H., Terrón, V., van Kerkwijk, M.H., de la Vega, A., Watkins, L.L., Weaver, B.A., Whitmore, J.B., Woillez, J., Zabalza, V., Astropy Contributors: The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. AJ **156** (3), 123 (2018) https://doi.org/10.3847/1538-3881/aabc4farXiv:1801.02634 [astro-ph.IM]
23. Harris, C.R., Millman, K.J., Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. Nature **585**(7825), 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2
24. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods **17**, 261–272 (2020) https://doi.org/10.1038/s41592-019-0686-2
25. Gommers, R., Virtanen, P., Burovski, E., Haberland, M., Weckesser, W., Oliphant, T.E., Reddy, T., Cournapeau, D., alexbrc, Nelson, A., Peterson, P., Roy, P., Wilson, J., Polat, I., endolith, Mayorov, N., Walt, S., Brett, M., Laxalde, D., Eric Larson, E., Millman, J., Sakai, A., Lars, peterbell10, Mulbregt, P., Carey, C., eric-jones, McKibben, N., Kern, R., Kai: Scipy/scipy: SciPy 1.11.0. https://doi.org/10.5281/zenodo.8079889
26. Wagg, T., Broekgaarden, F.S.: Streamlining and standardizing software citations with The Software Citation Station. arXiv e-prints, 2406–04405 (2024) arXiv:2406.04405 [astro-ph.IM]
27. Wagg, T., Broekgaarden, F., Gültekin, K.: TomWagg/software-citation-station: V1.2. https://doi.org/10.5281/zenodo.13225824