Towards virtual access of adaptive optics telemetry data

Tiago Gomes, Carlos Correia, Lisa Bardou, Olivier Beltramo-Martin, Thierry Fusco, Caroline Kulcsár, Timothy Morris, Nuno Morujão, Benoît Neichel, James Osborn, et al.

Towards virtual access of adaptive optics telemetry data

Tiago Gomes[a,f], Carlos Correia[b,f], Lisa Bardou[e], Olivier Beltramo-Martin[c], Thierry Fusco[c,d], Caroline Kulcsár[g], Timothy Morris[e], Nuno Morujão[h,f], Benoît Neichel[c], James Osborn[e], and Paulo Garcia[a,f]

[a]Faculdade de Engenharia da Universidade do Porto, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal
[b]Space ODT - Optical Deblurring Technologies, Porto, Portugal
[c]Aix Marseille Univ, CNRS, CNES, LAM, Marseille, France
[d]DOTA, ONERA, Université Paris Saclay, F-91123 Palaiseau, France
[e]Durham University (United Kingdom)
[f]Center for Astrophysics and Gravitation, Instituto Superior Técnico, Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal
[g]Université Paris-Saclay, Institut d'Optique Graduate School, CNRS, Laboratoire Charles Fabry, 91127 Palaiseau, France
[h]Departamento de Física e Astronomia, Faculdade de Ciências da Universidade do Porto, Rua do Campo Alegre s/n, 4169-007 Porto, Portugal

ABSTRACT

Large amounts of Adaptive-Optics (AO) control loop data and telemetry are currently inaccessible to end-users. Broadening access to those data has the potential to change the AO landscape on many fronts, addressing several use-cases such as derivation of the system's PSF, turbulence characterisation and optimisation of system control. We address one of the biggest obstacles to sharing these data: the lack of standardisation, which hinders access. We propose an object-oriented Python package for AO telemetry, whose data model abstracts the user from an underlining archive-ready data exchange standard based on the Flexible Image Transport System (FITS). Its design supports data from a wide range of existing and future AO systems, either in raw format or abstracted from actual instrument details. We exemplify its usage with data from active AO systems on 10m-class observatories, of which two are currently supported (AOF and Keck), with plans for more.

Keywords: Adaptive Optics, Telemetry, Data Exchange Standard, Data Model, PSF reconstruction, FITS, Object Oriented, Python

1. INTRODUCTION

Despite the increasing adoption of Adaptive Optics (AO) systems in astronomy, accessing and interpreting their telemetry data remains a real challenge. We use the term "telemetry" to represent AO internal signals such as wavefront sensor measurements, deformable mirror commands and several other reconstruction and control matrices and parameters. AO telemetry data has the potential to be used in many applications of high scientific interest, e.g., instrumentation research, deriving the state of the atmospheric turbulence during observations,[1–6] deriving the point spread function,[7–12] performing system performance estimation or runtime optimization.[13–15] Contradictorily, this type of data has been historically seen as "engineering data" and thus third-party access is disregarded by most data-producing systems.[16] While most of these systems already record large amounts of telemetry for internal use (including one-off instrument commissioning and regular calibration), these data are usually saved in private archives that are out of reach to the end-user, or it is simply not kept in the long-term. Even in cases where third-parties can access telemetry datasets, their documentation tends to be poor or non-existent. This complicates the task of interpreting and analysing them, given that the data points being shared and how they are structured usually varies significantly between systems (and sometimes even between datasets generated by the same system). In sum, the landscape is dominated by a complete lack of uniformity in the data, which ends up being one of the biggest obstacles to data access.

Given the extensive use-cases for AO telemetry data, broadening its access could change the AO and astronomy landscape on many fronts. Accordingly, many systems are starting to show interest in saving and sharing such data. One of the earliest examples of public archiving of AO telemetry was carried out by the CANARY project,[17] who made its open loop data for the sodium laser guide star experiment[18] available on the ESO Science Archive,[19] with a data format that is extensively detailed in an accompanying manual.[20] The Gemini Observatory is planning on publishing one of the first large scale archives of telemetry data[16] (mostly composed on wavefront slopes and DM commands), through the Gemini Observatory Archive.[21] However, publishing AO telemetry data currently requires significant effort from the responsible teams, as they are forced to design their own purpose-made data format, and make it available along with proper documentation for their prospective users. While such formats represent a significant step forward in data accessibility, given that they are tailored specifically for one instrument/observatory, they typically make no attempts at generalising support for multiple systems. In fact, to our knowledge, there has been no major attempt in the AO community to establish a consensual data format for AO telemetry to date.

We believe that access to AO telemetry data can only become ubiquitous by having the bulk of the AO community agree on a common way of sharing their data, that is, agreeing on a data exchange standard. A data exchange standard essentially defines a set of attributes/fields, their meanings, the way they relate to other data and the way they are stored in a shareable format. An ideal data exchange standard for AO telemetry would be able to package all relevant data in an unambiguous manner that is consistent across as many systems as possible, allowing data to be easily shared between data-producing systems and their end-users. It should also be structured in a generalised way that abstracts the user from observatory or instrument-specific details, without losing access to important information. While some of its fields should be mandatory in order to ensure the uniformity of data access, a significant portion should be optional, granting its flexibility to adapt to the irregularities of the telemetry being gathered by the different real-life instruments. Finally, it should be designed in a way that allows it to be continuously expanded to meet user needs and advancements in the AO field.

With all this in mind, we are developing AOT (Adaptive Optics Telemetry), a data exchange standard for AO telemetry data. This standard draws some inspiration from OIFITS,[22–24] which was created as a result of similar challenges that the interferometry community has faced. AOT is built on top of the Flexible Image Transport System (FITS),[25] meaning that AOT files fully comply with the latest FITS standard,[26] on top of which we define AOT-specific keywords and extensions. Specifically, we group data from different parts of the system with a set of 10 FITS binary tables, followed by as many image extensions as necessary to specify multi-dimensional data related to the system. While FITS is not an ideal standard for storing object-oriented data, building on top of FITS ensures significant compatibility with many of the existing tools, since any tool that fully supports the latest FITS standard will also be able to read AOT files, without requiring any modifications. A full discussion and exact specification of the first version of the AOT standard will be published separately at a later date. In this paper, we will be focused on the supporting mechanisms and tools that we have built for AOT.

The rate of adoption of a data exchange standard is its biggest metric of success, as low adoption would turn it into just another incompatible way of storing data, losing its intended purpose. Therefore, we are committed to alleviate anything that may prove to be a barrier to adoption. Specifically, we have developed *aotpy* (Adaptive Optics Telemetry for Python), an object-oriented Python[27] package that supports this data exchange standard. The goal of this package is to facilitate reading, editing and writing of AOT files, by abstracting the user from the actual file handling and its structure. It does so by implementing a data model through which the user can interact with all relevant data. We also provide preliminary "Translator" methods, currently for two AO systems on 10m-class observatories. These methods essentially act as interfaces between non-standard data currently being produced by these systems and our standard data model, allowing the user to work with supported legacy data via *aotpy* and in turn create standard AOT files for it. This feature is important as it bridges the gap for legacy systems that are unlikely to adopt a newer standard.

This project is being developed in the context of OPTICON–RadioNet Pilot, an European collaboration through which we aim to promote broad discussion and consensus within the AO community. We plan on making available on the ESO Science Archive a set of AOT files for existing ESO systems; preliminary talks have been established with responsible teams, to ensure that no showstoppers exist.

In this paper we will be going through the implementation of the *aotpy* package, code demonstrations, examples of its use-cases and the work that we have currently planned for the future.

2. PYTHON PACKAGE

aotpy is an object-oriented Python package that defines a set of classes and variables which can hold all data that may be shared through an AOT file, as well as interfaces for writing and reading such files. In the future, this package will also provide tools to explore data in AOT files interactively.

While currently this package only provides support for FITS files, it was designed with the intention of enabling an easy expansion to other file types, as long as these can still be represented by the *aotpy* data model.

2.1 Data Model and Classes

The classes in *aotpy* serve as abstractions of physical objects, their environment or the way they interact and behave in a AO system. Similar standpoints have been adopted in AO simulation tools such as OOMAO.[28] They are implemented as Python Data Classes and provide access to a set of related AO data, which aims to be generalised enough to uniformly support distinct AO systems (classical with single natural or laser guide stars, multi-wavefront sensor systems, tomographic, etc.). For this purpose, common object-oriented techniques such as inheritance and polymorphism are employed.

An overview of the data model implemented by *aotpy* is provided in Fig. 1. Each AOT file can be fully represented by a single instance of *AOSystem*, which contains telemetry from a single observation, as well as other data that may contextualise and support the analysis and interpretation of this telemetry.

A general description of all classes in *aotpy* is provided in the following sections. Note that most classes have a *name* variable, which is a unique user-defined identifier (that is, no other objects of the same type may have the exact same name). This is so that, even in scenarios where direct references to objects are not possible (for example, within a FITS file), these objects may still be uniquely referenced via their names.

2.1.1 AOSystem

This is the central class in *aotpy*. Its purpose is to provide general information about the observation, as well as references to all the different objects that detail the system (specifically, it references *Source*, *AtmosphericParameters*, *MainTelescope*, *ScoringCamera*, *WavefrontSensor*, *RTC* and *WavefrontCorrector* objects). A diagram defining this class can be found in Fig. 2.

2.1.2 Image

This class provides a generalised interface for multi-dimensional arrays of values. The array itself is stored as a NumPy[29] *ndarray*, but some metadata (such as units of measurement) can also be associated with an *Image* object.

Other classes in *aotpy* contain references to *Image* objects whenever some data cannot be represented as a single value or as a one-dimensional list of values. The same *Image* object may be referenced by multiple objects in the system, avoiding data duplication.

A representation of this class can be seen in Fig. 3.

2.1.3 AtmosphericParameters

Atmospheric data that may be relevant to the observation in question, mainly in regards to astronomical seeing and turbulence profiles, can be stored in *AtmosphericParameters* instances. The source of such data, as well as the moment when it was recorded, is coupled with the data itself. This is shown in Fig. 4.

2.1.4 Source

In *aotpy*, *Source* objects hold data related to the positioning of a light source. They shall be instantiated as either a *NaturalGuideStar* or a *LaserGuideStar*. The latter may contain further data related to the laser launch telescope used to create the source, and the sodium layer being targeted by it, as seen in Fig. 5.

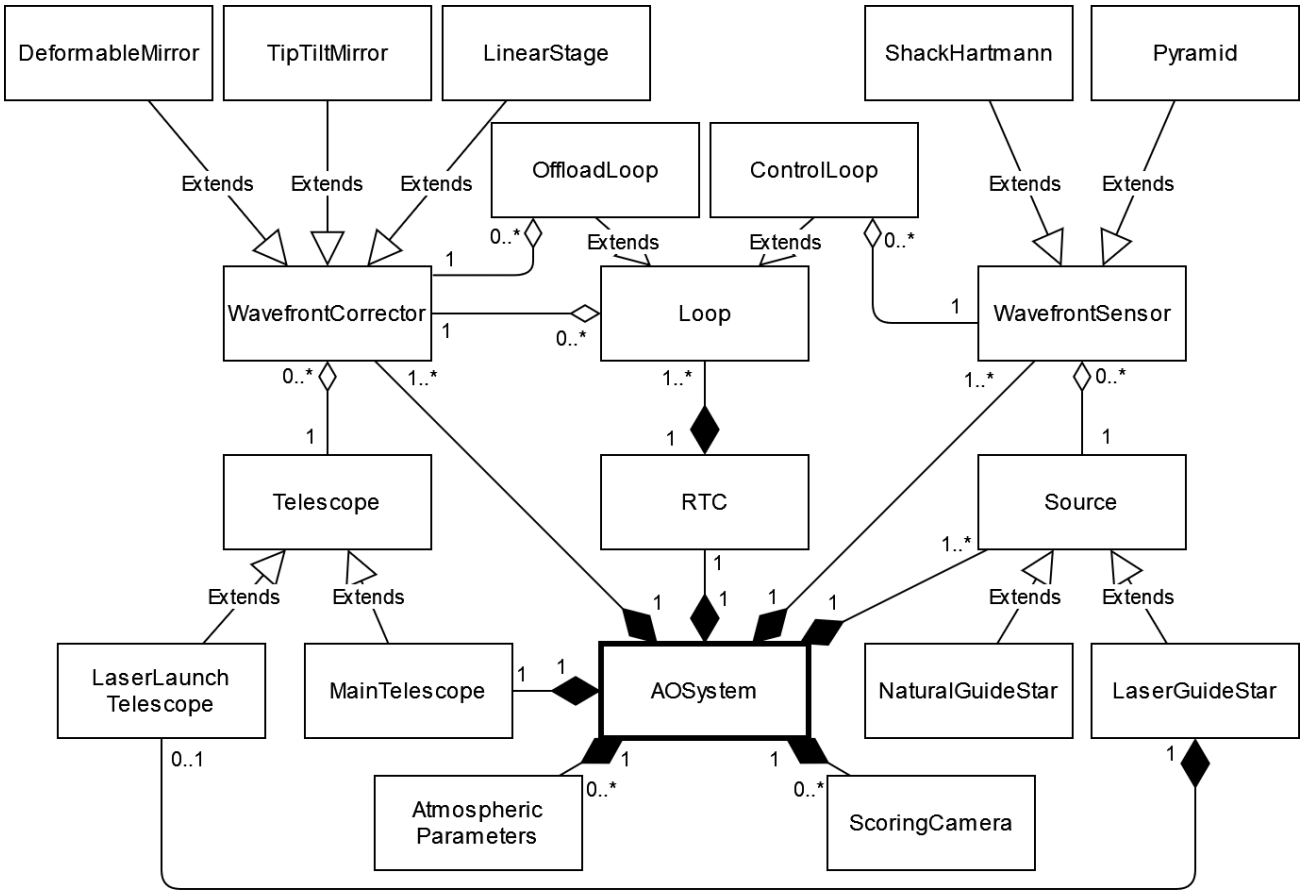

Figure 1. A language-agnostic UML Class Diagram that provides an overview of the data model and classes in *aotpy*.

2.1.5 Telescope

Telescope objects contain data about a telescope's physical characteristics and configuration at the time of observation. These objects should be instantiated as *MainTelescope*, if they contain data regarding the telescope at which the observation itself is performed, or *LaserLaunchTelescope* if their data relates to the telescopes used to create artificial guide stars.

Each *AOSystem* must only contain one *MainTelescope* object (but as many *LaserLaunchTelescope* objects as necessary). The contents of these classes are represented in Fig. 6.

2.1.6 Detector

Detector objects are an integral part of objects like *ScoringCamera* and *WavefrontSensor*. A *Detector* object may hold a wide set of data related to the physical characteristics and configurations of a detector, as well as the actual pixels recorded by it. A detailed representation of its contents is provided in Fig. 7.

2.1.7 OpticalRelay

An instance of this class contains data related to the physical configuration of the optical assembly, which may be present in *ScoringCamera* and *WavefrontSensor* objects. This is a rather simple class, which is represented in Fig. 7.

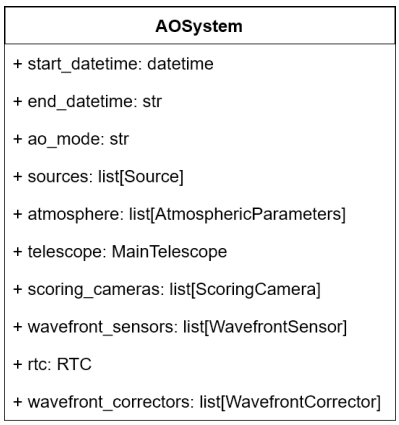

Figure 2. UML Class Diagram for the *AOSystem* class.

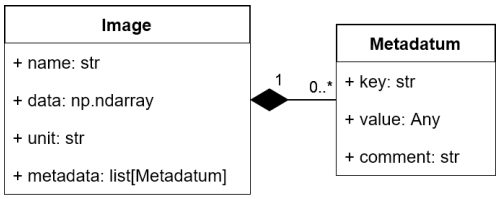

Figure 3. UML Class Diagram for the *Image* and *Metadatum* classes.

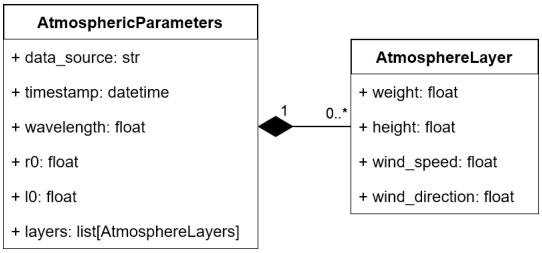

Figure 4. UML Class Diagram for the *AtmosphericParameters* and *AtmosphereLayer* classes.

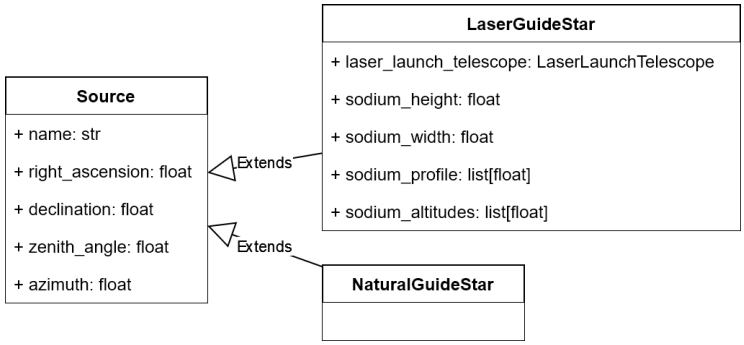

Figure 5. UML Class Diagram for the *Source* class and its subclasses *LaserGuideStar* and *NaturalGuideStar*.

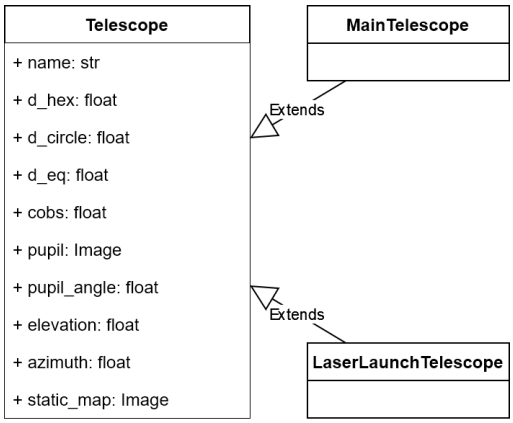

Figure 6. UML Class Diagram for the *Telescope* class and its subclasses *MainTelescope* and *LaserLaunchTelescope*.

2.1.8 OpticalAberration

This class provides variables that allow a user to describe optical aberrations. In *aotpy* these may be associated to *ScoringCamera* and *WavefrontSensor* objects (as represented in Fig. 7) but also to *WavefrontCorrector* objects.

2.1.9 ScoringCamera

A *ScoringCamera* object may contain some data about the characteristics of the camera, as well as references to related *Detector*, *OpticalRelay* and *OpticalAberration* objects. However, it is important to note that the science data itself is not included here, or anywhere else in the package or the AOT standard, given that the focus is in handling telemetry data (usually, science data is already accessible separately). Fig. 7 details this class as well as its relation with other classes.

2.1.10 WavefrontSensor

A *WavefrontSensor* object may contain a wide set of data related to its subapertures, the detected wavefront (such as gradients and intensities) and the algorithms and gain values involved. It can also provide references to related *Detector*, *OpticalRelay* and *OpticalAberration* objects. *WavefrontSensor* objects must always provide a reference to the *Source* that they are sensing.

Objects shall not be instantiated as *WavefrontSensor*, but specifically as one of the two subclasses provided: *ShackHartmann* or *Pyramid*. These subclasses support extra data that are specific to that type of Wavefront Sensor. Support for further types may be provided in the future.

A detailed representation of the contents of this class, its subclasses and interactions with other classes can be seen in Fig. 7.

2.1.11 WavefrontCorrector

An instance of this class characterises a set of actuators that are commanded jointly to correct wavefronts. *WavefrontCorrector* objects must always provide a reference to the *Telescope* that they are installed on (either a *MainTelescope* or a *LaserLaunchTelescope*) and may also provide a reference to related *OpticalAberration* objects. Objects shall not be instantiated as *WavefrontCorrector*, but specifically as one of the three subclasses provided: *DeformableMirror*, *TipTiltMirror* or *LinearStage*. The difference in these types of Wavefront Correctors lies mostly on the number of actuators involved (a Linear Stage has a single actuator, for translation, a Tip-Tilt Mirror has two actuators, for tip and tilt, and a Deformable Mirror may have any number of actuators larger than 2). Other types of correctors may be supported in the future. A diagram for this class is provided in Fig. 8.

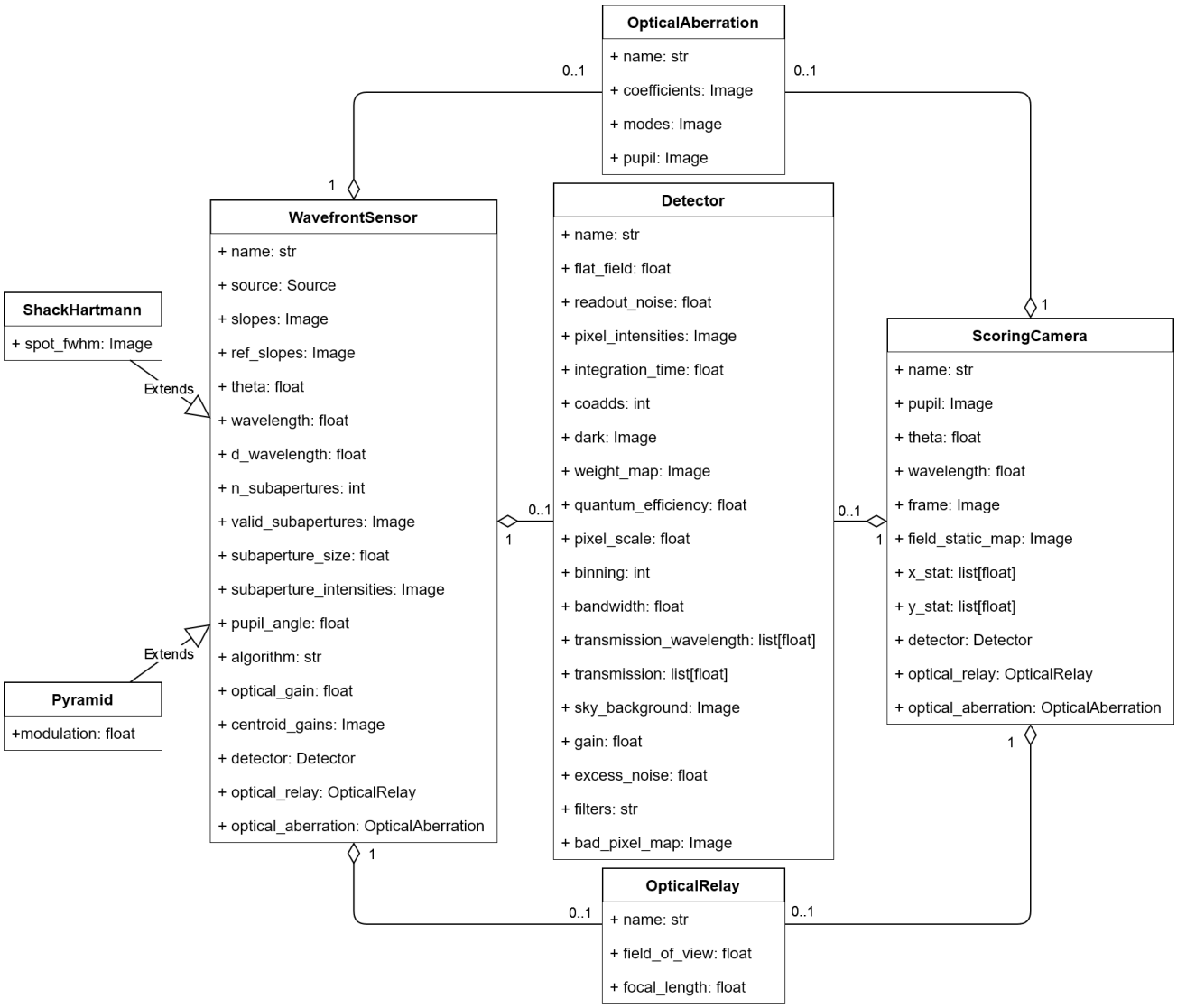

Figure 7. UML Class Diagram for all the classes related to optical sensors. This includes *Detector*, *OpticalRelay*, *OpticalAberration*, *ScoringCamera*, *WavefrontSensor* and its subclasses *ShackHartmann* and *Pyramid*

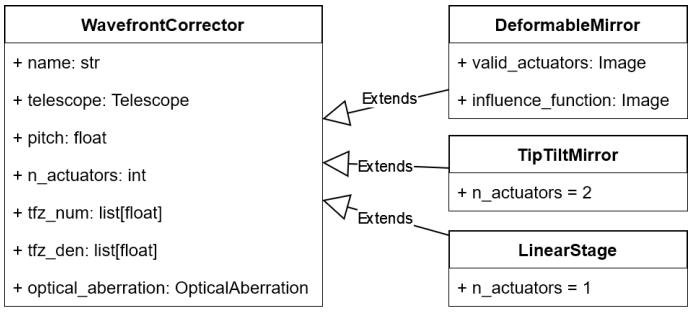

Figure 8. UML Class Diagram for the *WavefrontCorrector* class and its subclasses *DeformableMirror*, *TipTiltMirror* and *LinearStage*.

2.1.12 RTC

The *RTC* (Real-Time Computer) is a conceptual object that holds lookup tables (LUTs) and non-common path aberrations (NCPAs) that may be used in AO computation, as well as a set of *Loop* objects.

In the context of AO, a loop may be generally understood as the process through which the RTC periodically gathers data from a certain input and calculates a resulting output data, which is usually a set of commands sent to some sort of *WavefrontCorrector*. In the case of *ControlLoop* objects, wavefront data (as sensed by a *WavefrontSensor*) are taken as the input, from which the RTC calculates the necessary adjustments to a *WavefrontCorrector*, based on interaction and control matrices. As for *OffloadLoop* objects, the input is instead the set of commands received by one *WavefrontCorrector*, which then results on an output which is offloaded to another *WavefrontCorrector*, based on a offload matrix. *Loop* objects shall be instantiated as their specific subclass, in order to properly setup the objects they coordinate.

In the real-world, an AO loop may command multiple wavefront correctors based on a single wavefront sensor, or even multiple wavefront sensors may be used to command a single wavefront corrector. While *aotpy* can only instantiate one-to-one relationships, this limitation can easily be worked around by instantiating multiple *Loop* objects that represent the same real-world conceptual loop*, with minimal overhead (given that the duplicated data is mostly composed of references).

It is important to note that, since *ControlLoop* objects provide references to one *WavefrontSensor* object and one *WavefrontCorrector*, which themselves provide references to one *Source* object and one *Telescope* object respectively, we can access the entire chain of objects that are related to a single *ControlLoop* object via references. This property can be easily observed in the overview of the data model in Fig. 1, as all of these objects are linked together by aggregation. For a more specific view into the *RTC*, *Loop* and its subclasses, Fig. 9 is provided.

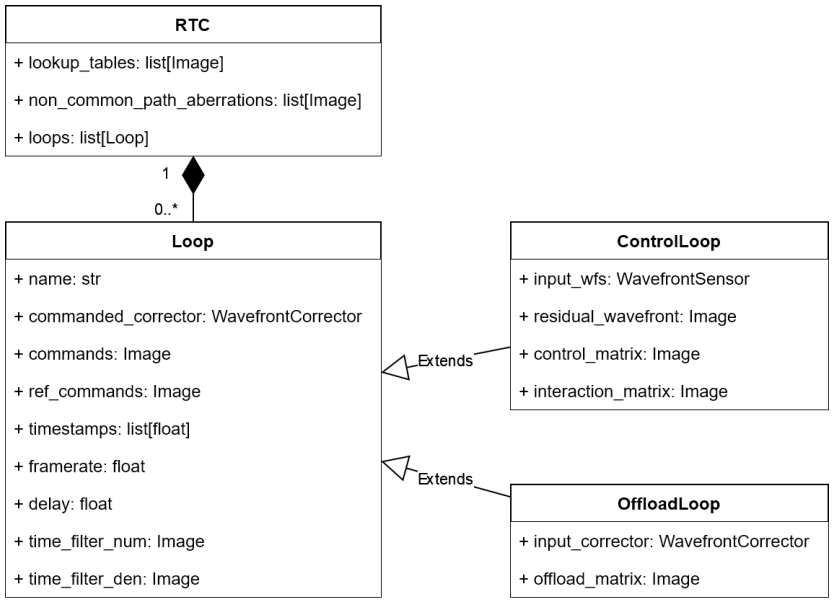

Figure 9. UML Class Diagram for the *RTC* class as well as *Loop* and its subclasses *ControlLoop* and *OffloadLoop*.

*For example, if a real-world AO loop uses a single wavefront sensor to command multiple wavefront correctors, in *aotpy* we can create one *ControlLoop* object for each wavefront corrector being commanded. These objects will all have the same *WavefrontSensor* object as input, but their control matrices and commands will differ based on their respective *WavefrontCorrector* object (output).

2.2 Readers and Writers

In *aotpy*, the task of a Reader is to receive one AOT file as input and return a single *AOSystem* object. Conversely, a Writer is able to write an *AOSystem* object into an AOT file. These tools completely abstract users from the file handling process, allowing them to analyse, edit and store AO telemetry without any prior knowledge of the actual file structure and specification. It is important to note that the reading and writing process is independent from the actual source of the data or its contents, as it simply handles a standardised *AOSystem* object. This means that, for AOT files, we can handle data produced by any system with a single Reader/Writer pair, for a given file type. A diagram representing the overall functionality of the Readers and Writers can be seen in Fig. 10.

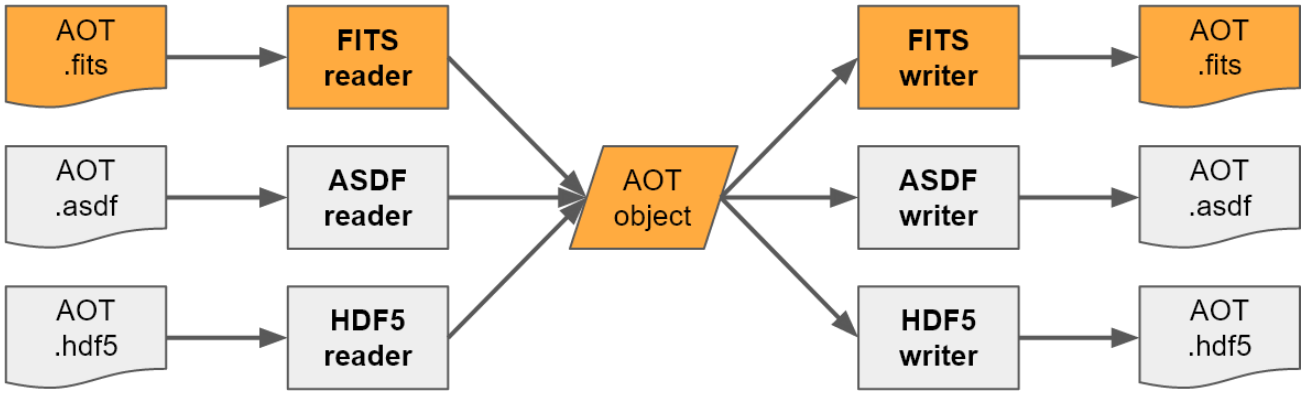

Figure 10. Diagram representing the functionality of the Readers and Writers in *aotpy*. The FITS Reader and Writer are currently implemented, while the others are examples of file types that could be supported. The "AOT object" here is a *AOSystem* object, which can be freely edited before being written again.

Currently, only a FITS Reader and a FITS Writer are provided, but other file types (such as ASDF[30] or HDF5[31]) could be supported by creating corresponding Reader/Writer pairs, without requiring any change to the data model or classes provided by *aotpy* (though this is not currently planned). The FITS Reader/Writer implementation makes heavy use of Astropy[32]'s FITS File Handling package. For all of the main classes, the FITS writer essentially aggregates objects of that type and stores their data in one table per type, with the exception of *Image* objects, which are stored separately as FITS image extensions.

2.3 Translators

While ideally all data-producing systems would adopt the AOT standard, in practice the current state of AO telemetry data is mostly not standardised (as previously described in Sec. 1). This means that most data currently available cannot to be read by the Readers provided by this package. To ease the adoption of AOT, *aotpy* aims to provide Translators for a set of relevant data-producing systems. For a supported data-producing system, a Translator is able to read a given set of files (produced by that system) that are related to a single observation, and then use that data to populate the corresponding fields of an *AOSystem* object. This object can then be handled as any other *AOSystem* object, particularly it can then be written to an AOT file, allowing for future reading of that data without requiring the user to go through the translation process again. A conceptual diagram of this feature can be seen in Fig. 11.

Translators are highly system specific, as they require in-depth knowledge of how a given system stores its data. As such, their development requires significant collaboration with system designers/engineers familiar with that system. Therefore, *aotpy* is currently only able to provide preliminary Translator support for two systems: the Adaptive Optics Facility (AOF) at ESO's VLT[33] and the AO system at the W. M. Keck Observatory.[34] Preliminary support for other systems in ESO's VLT (specifically, NAOMI[35,36] and CIAO[37]) is currently under development.

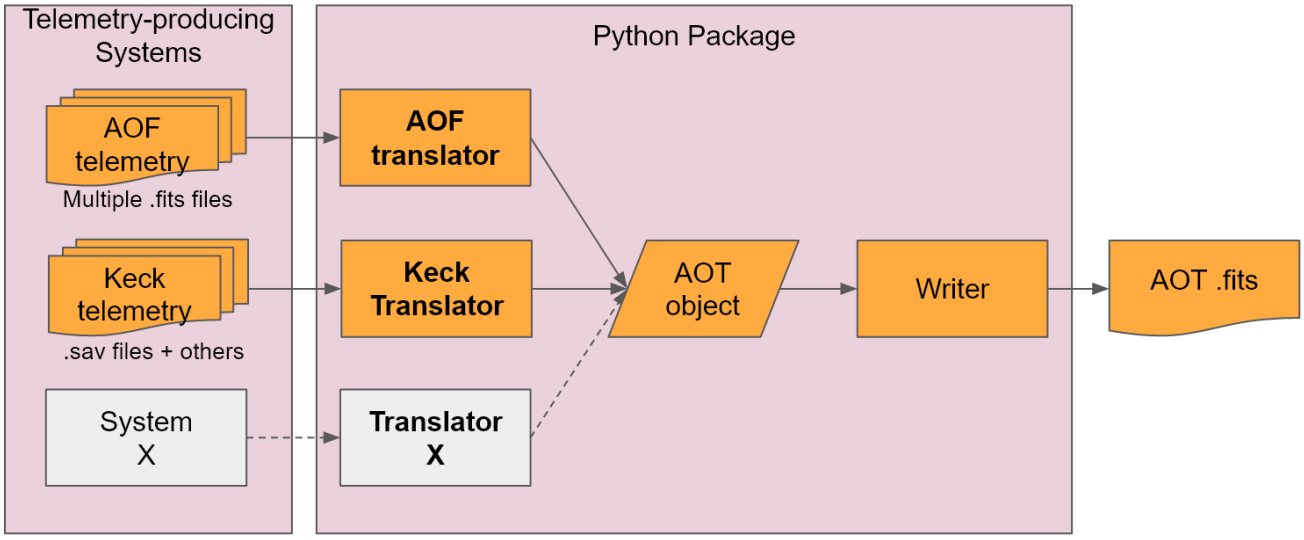

Figure 11. Diagram representing the functionality of Translators in *aotpy*. The translators for AOF and Keck are currently implemented, while X represents other translators that could be supported, *e.g.* NAOMI or CIAO. The standard *AOSystem* object could be directly used for any pipeline that accepts these objects, so writing the object to a file is purely optional.

AOF telemetry data is usually recorded manually and takes the form of more than 50 distinct FITS files per observation, which together contain intensity and gradient data of both of its wavefront sensors, the pixels of the NGS WFS detector, the positions of its DSM and other correctors present in its laser launch telescopes, control/offload/interaction matrices and a wide range of data regarding the characteristics of the detectors in the system. The AOF translator is able to package all of the relevant data made available in these files, along with a few configuration matrices that were sourced externally, into the standardised *AOSystem* object.

The Translator for Keck currently supports both its NGS and LGS mode, for the NIRC2 instrument. Telemetry data is provided in the form of an IDL SAVE file (for which SciPy[38]'s *readsav* routine is used), which mostly contains data regarding its wavefront sensors (specifically slopes for one or two of its WFS, depending on the mode), the commands sent to its deformable and tip-tilt mirrors and the corresponding control/interaction matrices. Relevant atmospheric data can be obtained through the Mauna Kea Weather Center seeing page.[39] Other relevant data such as pupil masks, NCPAs and filter characteristics has been sourced externally.

2.4 Code Demonstrations

In order to exemplify the usage of this package and to familiarise the reader with its workflow, in the following sections we provide three examples of common tasks that may be performed with *aotpy*, each having code excerpts along with explanations of the logic behind them.

2.4.1 Creating an AOT file from scratch

Let's say that we have some AO telemetry data that we want to store following the AOT specification. For this demonstration we will assume that we have a system with one NGS and one LGS, each being sensed by a different Shack-Hartmann wavefront sensor (with 4 subapertures), and the RTC uses data from these WFSs to calculate commands for a single Deformable Mirror in the system (32 actuators). The data is assumed to have been recorded for 10000 frames. For simplicity, we will focus mostly on the slopes and commands data, but keep in mind that we could add large amounts of data related to physical characteristics of all the objects.

We'll start by importing all the classes that we will need for this example, as well as NumPy, which we will use to create some dummy data for demonstration purposes.

```
import numpy as np
from aotpy import AOSystem, RTC, MainTelescope, NaturalGuideStar, LaserGuideStar,\
    ShackHartmann, ControlLoop, DeformableMirror, Image
from aotpy.fits import write_to_fits
```

Then, we create our main telescope, as well as its single DM.

```
tel = MainTelescope(name="Example telescope")
cor = DeformableMirror(name="Example DM", n_actuators=32, telescope=tel)
```

We then create an object that represents our NGS, followed by the wavefront sensor that is sensing that source. For this wavefront sensor, we provide a 10000-by-8 image, representing the vertical and horizontal slopes of the 4 subapertures during all of the recorded frames.

```
ngs = NaturalGuideStar(name="Example NGS")
ngs_wfs = ShackHartmann(name="WFS1", source=ngs,
                        slopes=Image(name="NGS slopes", data=np.ones((10000, 8))))
```

After that, we do a similar process for the LGS, making sure we use the appropriate class.

```
lgs = LaserGuideStar(name="Example LGS")
lgs_wfs = ShackHartmann(name="WFS2", source=lgs,
                        slopes=Image(name="LGS slopes", data=np.ones((10000, 8))))
```

Please note that, if in a certain scenario we knew that two different objects in our system were referencing the same multi-dimensional data, instead of creating two separate *Image* objects we could create a single *Image* object that is referenced by both of them. This would be preferable as it would avoid data duplication in the *AOSystem* object and also in any resulting AOT files. That said, while in this demonstration the slopes data happens to be the exact same for both wavefront sensors, in a real-world example this would be unlikely, so we kept these as two separate *Image* objects.

With the previous steps completed, we now can create the control loops in the system. Let's assume we have a High Order (HO) loop that uses data from the LGS, and a Low Order (LO) loop that uses data from the NGS. For each loop, we have a 10000-by-32 image that represents the commands sent to the DM. We also have the two 8-by-32 control matrices that were used to create the commands.

```
lo_loop = ControlLoop(name="LO loop", input_wfs=ngs_wfs, commanded_corrector=cor,
                      commands=Image(name="LO commands", data=np.ones((10000, 32))),
                      control_matrix=Image(name='LO control matrix', data=np.ones((8, 32))))
ho_loop = ControlLoop(name="HO loop", input_wfs=lgs_wfs, commanded_corrector=cor,
                      commands=Image(name="HO commands", data=np.ones((10000, 32))),
                      control_matrix=Image(name='HO control matrix', data=np.ones((8, 32))))
```

Then we need to create an *RTC* object that contains the loops that we created.

```
rtc = RTC(loops=[lo_loop, ho_loop])
```

To wrap it all up into a single object, we create an *AOSystem* object, which receives references to the objects that we created.

```
system = AOSystem(sources=[ngs, lgs],
                  telescope=tel,
                  wavefront_sensors=[ngs_wfs, lgs_wfs],
                  rtc=rtc,
                  wavefront_correctors=[cor])
```

Finally, we write this object into a file, by providing the Writer function with the relevant path (including filename), which may be relative or absolute.

```
write_to_fits(system, "example.fits")
```

The structure of the resulting FITS file can be seen in Fig. 12. Note that, while NumPy handles arrays in row-major order by default, arrays stored in FITS always use column-major order, which is why the order of the dimensions appears reversed in the figure. However, the user is completely abstracted from this conversion, as it is handled automatically by *aotpy*.

Index	Extension	Type	Dimension
0	Primary	Image	0
1	AOT_ATMOSPHERIC_PARAMETERS	Binary	9 cols X 0 rows
2	AOT_DETECTORS	Binary	19 cols X 0 rows
3	AOT_OPTICAL_RELAYS	Binary	3 cols X 0 rows
4	AOT_OPTICAL_ABERRATIONS	Binary	4 cols X 0 rows
5	AOT_SCORING_CAMERAS	Binary	11 cols X 0 rows
6	AOT_WAVEFRONT_SENSORS	Binary	21 cols X 2 rows
7	AOT_RTC	Binary	15 cols X 2 rows
8	AOT_SOURCES	Binary	10 cols X 2 rows
9	AOT_TELESCOPES	Binary	11 cols X 1 rows
10	AOT_WAVEFRONT_CORRECTORS	Binary	10 cols X 1 rows
11	NGS SLOPES	Image	8 X 10000
12	LGS SLOPES	Image	8 X 10000
13	HO COMMANDS	Image	32 X 10000
14	HO CONTROL MATRIX	Image	32 X 8
15	LO COMMANDS	Image	32 X 10000
16	LO CONTROL MATRIX	Image	32 X 8

Figure 12. Screenshot of the "example.fits" file created in Sec. 2.4.1 being opened with fv FITS viewer.[40,41] As expected, we have two entries in the wavefront sensors, sources and RTC tables, while we have just one entry for the telescope and wavefront correctors tables. All the remaining tables have no entries. We also have 6 image extensions, which match the images created for the slopes, commands and control matrices.

2.4.2 Reading and editing an existing AOT file

Now that we have created an AOT file in the previous section, it can be shared with anyone that might be interested in the data. While they will be able to read the file with any FITS reader that supports the latest specification, they can also read the data through *aotpy*, which provides useful abstractions and enables the user to easily edit the contents.

To exemplify this, we can start by importing the relevant Reader function and then providing it with the path to the file we want to read (in this case, the file created in the previous section).

```
from aotpy.fits import read_from_fits
system = read_from_fits('example.fits')
```

After the reading process is finished, *system* is an *AOSystem* object whose contents are completely equivalent to the *system* variable that we had at the end of the previous section. This means that we are essentially picking back up from where we left off in the previous section.

This variable can now be explored in the same way as any Python object can be explored, and it is also directly editable by changing the values associated with its attributes. For example, we could specify the framerate of the LO and HO loops as such:

```
system.rtc.loops[0].framerate = 500
system.rtc.loops[1].framerate = 1000
```

While in the previous step we have locally edited this variable, keep in mind that this change is not automatically reflected in the file itself, by design. In order to achieve this, one would have to use a Writer to write this edited variable into a new FITS file.

2.4.3 Translating from non-standard files

When a Translator for a certain dataset is available, handling that data is a simple process. The first step is to import the relevant translation method and then calling it by passing the path to the folder which contains all the relevant files. For AOF data, this could be achieved in the following way:

```
from aotpy.translators.aof import load_from_aof
system = load_from_aof('path/to/folder')
```

From this point on, *system* is a standard *AOSystem* object which can be freely edited and explored. We could, for example, write this object into an AOT FITS file:

```
from aotpy.fits import write_to_fits
write_to_fits(system, 'aof.fits')
```

An overview of the contents of the resulting "aof.fits" file can be seen in Fig. 13.

3. EXAMPLE USE-CASES

To demonstrate the usefulness of AO telemetry data, we provide an example of a scientifically relevant use-case with real-world data. Specifically, a topic of recent investigation efforts is the estimation of atmospheric turbulence parameters from Shack–Hartmann slopes.[6]

Using a work-in-progress *aotpy* Translator for NAOMI data, we created an *AOSystem* object that was then fed into an internal pipeline for estimating atmospheric turbulence parameters. The resulting estimation is illustrated in Fig. 14.

4. FUTURE WORK

Given the importance of community acceptance, we are developing this project following a "bottom-up" approach. We start by releasing the *aotpy* package to the community (fully open-sourced) in its current state, as a beta version. These supporting tools allow the community to experiment with the standard and provide feedback at an earlier stage, allowing us to rapidly iterate the standard and package to fix any issues that arise and to better match the needs of the community.

Then, in Fall 2022, we will release a paper fully describing the AOT FITS standard, which will mark the release of its version 1.0. Along with it the version 1.0 of the *aotpy* package will also be released, marking

Index	Extension	Type	Dimension
0	Primary	Image	0
1	AOT_ATMOSPHERIC_PARAMETERS	Binary	9 cols X 0 rows
2	AOT_DETECTORS	Binary	19 cols X 5 rows
3	AOT_OPTICAL_RELAYS	Binary	3 cols X 0 rows
4	AOT_OPTICAL_ABERRATIONS	Binary	4 cols X 0 rows
5	AOT_SCORING_CAMERAS	Binary	11 cols X 0 rows
6	AOT_WAVEFRONT_SENSORS	Binary	21 cols X 5 rows
7	AOT_RTC	Binary	13 cols X 13 rows
8	AOT_SOURCES	Binary	10 cols X 5 rows
9	AOT_TELESCOPES	Binary	11 cols X 5 rows
10	AOT_WAVEFRONT_CORRECTORS	Binary	10 cols X 9 rows
11	LGSACQ.DET1.DARK	Image	240 X 240
12	LGSACQ.DET1.WEIGHT	Image	240 X 240
13	WFS1_GRADIENTS	Image	2480 X 10000
14	LGSACQ.DET1.REFSLP_WITH_OFFSETS	Image	2480 X 1
15	SAVALID_LGS1	Image	1098 X 1
16	WFS1_INTENSITIES	Image	1240 X 10000
17	LGSACQ.DET2.DARK	Image	240 X 240
18	LGSACQ.DET2.WEIGHT	Image	240 X 240
19	WFS2_GRADIENTS	Image	2480 X 10000
20	LGSACQ.DET2.REFSLP_WITH_OFFSETS	Image	2480 X 1
21	SAVALID_LGS2	Image	1111 X 1
22	WFS2_INTENSITIES	Image	1240 X 10000
23	LGSACQ.DET3.DARK	Image	240 X 240
24	LGSACQ.DET3.WEIGHT	Image	240 X 240
25	WFS3_GRADIENTS	Image	2480 X 10000
26	LGSACQ.DET3.REFSLP_WITH_OFFSETS	Image	2480 X 1
27	SAVALID_LGS3	Image	1087 X 1
28	WFS3_INTENSITIES	Image	1240 X 10000
29	LGSACQ.DET4.DARK	Image	240 X 240
30	LGSACQ.DET4.WEIGHT	Image	240 X 240
31	WFS4_GRADIENTS	Image	2480 X 10000
32	LGSACQ.DET4.REFSLP_WITH_OFFSETS	Image	2480 X 1
33	SAVALID_LGS4	Image	1079 X 1
34	WFS4_INTENSITIES	Image	1240 X 10000
35	NGS PIXELS	Image	16 X 16 X 100
36	IRACQ.DET1.DARK	Image	16 X 16
37	IRACQ.DET1.WEIGHT	Image	16 X 16
38	WFS_GRADIENTS	Image	8 X 5000
39	IRACQ.DET1.REFSLP_WITH_OFFSETS	Image	8 X 1

Figure 13. Partial screenshot of fv opening an AOT file containing standardised AOF telemetry. Of 81 total extensions (HDUs) contained in this particular file, an overview of the first 40 is shown.

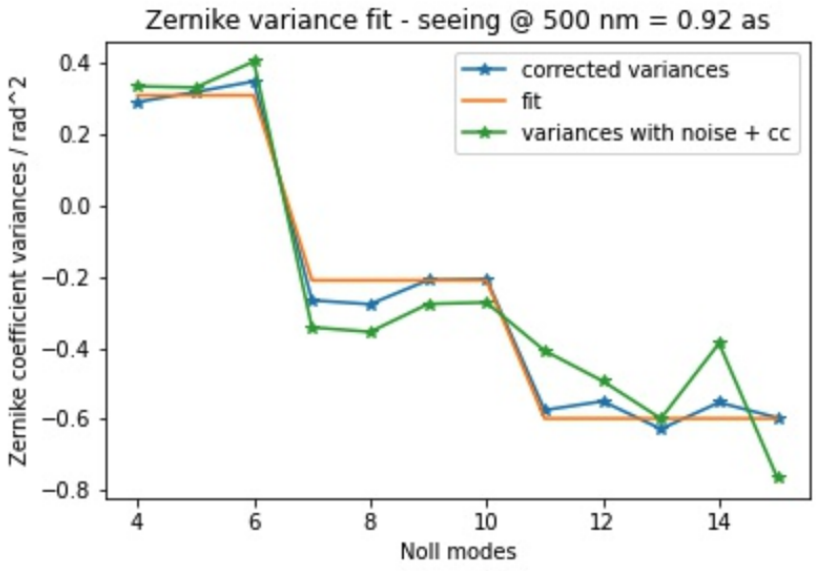

Figure 14. Turbulence estimation fit utilising telemetry data for the NAOMI system. Utilising the methods described in Ref. 6 we can obtain an estimation of the characteristic parameters of turbulence, the Fried parameter and the outer-scale, by correcting for the effects of cross-talk and aliasing inherently present in the finite sensor measurement. From the obtained fit we can estimate a r_0 of (11.36 ± 1.02) cm and a L_0 of (14.1 ± 1.5) m.

the end of its beta phase. From this point forward, file backwards compatibility will be guaranteed, ensuring that a file that respects any version of the AOT standard will always be readable by the latest version *aotpy*. Although, it is expected that the new versions of AOT FITS standard may be developed in the future (with expansions to meet advancements in the field and user needs), these will be accompanied by new *aotpy* versions that guarantee compatibility. Specifically, we aim to adequately support future ELT-class observatories, by making any necessary updates as their requirements become clearer.

In the short term, *aotpy* will provide Translator support for additional ESO systems (NAOMI and CIAO). A selection of datasets from ESO systems will be made available in the ESO Science Archive, following the AOT standard.

ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No. 730562 (OPTICON) and 101004719 (OPTICON–RadioNet Pilot).

REFERENCES

[1] Vidal, F., Gendron, E., and Rousset, G., "Tomography approach for multi-object adaptive optics," *Journal of the Optical Society of America A* **27**, A260000 (Oct. 2010).

[2] Guesalaga, A., Neichel, B., Cortés, A., Béchet, C., and Carmine, G., "Using the c_n^2 and wind profiler method with wide-field laser-guide-stars adaptive optics to quantify the frozen-flow decay," *Monthly Notices of the Royal Astronomical Society* **440**, 1925–1933 (2014).

[3] Martin, O. A., Correia, C. M., Gendron, E., Rousset, G., Vidal, F., Morris, T. J., Basden, A. G., Myers, R. M., Ono, Y. H., Neichel, B., and Fusco, T., "William Herschel Telescope site characterization using the MOAO pathfinder CANARY on-sky data," in [*Adaptive Optics Systems V*], Marchetti, E., Close, L. M., and Véran, J.-P., eds., **9909**, 1143 – 1157, International Society for Optics and Photonics, SPIE (2016).

[4] Jolissaint, L., Ragland, S., Christou, J., and Wizinowich, P., "Determination of the optical turbulence parameters from the adaptive optics telemetry: critical analysis and on-sky validation," *Appl. Opt.* **57**, 7837–7856 (Sep 2018).

[5] Laidlaw, D. J., Osborn, J., Morris, T. J., Basden, A. G., Gendron, E., Rousset, G., Townson, M. J., and Wilson, R. W., "Automated wind velocity profiling from adaptive optics telemetry," *Monthly Notices of the Royal Astronomical Society* **491**, 1287–1294 (11 2019).

[6] Andrade, P. P., Garcia, P. J., Correia, C. M., Kolb, J., and Carvalho, M. I., "Estimation of atmospheric turbulence parameters from Shack–Hartmann wavefront sensor measurements," *Monthly Notices of the Royal Astronomical Society* **483**(1), 1192–1201 (2019).

[7] Veran, J. P., Rigaut, F., Maitre, H., and Rouan, D., "Estimation of the adaptive optics long-exposure point-spread function using control loop data.," *Journal of the Optical Society of America A* **14**, 3057–3069 (Nov. 1997).

[8] Gendron, E., Clénet, Y., Fusco, T., and Rousset, G., "New algorithms for adaptive optics point-spread function reconstruction," *Astronomy & Astrophysics* **457**, 359–363 (Oct. 2006).

[9] Gilles, L., Correia, C., Véran, J.-P., Wang, L., and Ellerbroek, B., "Simulation model based approach for long exposure atmospheric point spread function reconstruction for laser guide star multiconjugate adaptive optics," *Applied Optics* **51**(31), 7443–7458 (2012).

[10] Beltramo-Martin, O., Correia, C. M., Ragland, S., Jolissaint, L., Neichel, B., Fusco, T., and Wizinowich, P. L., "PRIME: PSF Reconstruction and Identification for Multiple-source characterization Enhancement - application to Keck NIRC2 imager," *Monthly Notices of the Royal Astronomical Society* **487**, 5450–5462 (Aug. 2019).

[11] Fusco, T., Bacon, R., Kamann, S., Conseil, S., Neichel, B., Correia, C., Beltramo-Martin, O., Vernet, J., Kolb, J., and Madec, P. Y., "Reconstruction of the ground-layer adaptive-optics point spread function for MUSE wide field mode observations," *Astronomy & Astrophysics* **635**, A208 (Mar. 2020).

[12] Beltramo-Martin, O., Marasco, A., Fusco, T., Massari, D., Milli, J., Fiorentino, G., and Neichel, B., "Pushing point-spread function reconstruction to the next level: application to SPHERE/ZIMPOL," *Monthly Notices of the Royal Astronomical Society* **494**, 775–788 (May 2020).

[13] Sivo, G., Kulcsár, C., Conan, J.-M., Raynaud, H.-F., Gendron, É., Basden, A., Vidal, F., Morris, T., Meimon, S., Petit, C., et al., "First on-sky scao validation of full lqg control with vibration mitigation on the canary pathfinder," *Optics express* **22**(19), 23565–23591 (2014).

[14] Petit, C., Sauvage, J.-F., Fusco, T., Sevin, A., Suarez, M., Costille, A., Vigan, A., Soenke, C., Perret, D., Rochat, S., et al., "Sphere extreme ao control scheme: final performance assessment and on sky validation of the first auto-tuned lqg based operational system," in [*Adaptive Optics Systems IV*], **9148**, 214–230, SPIE (2014).

[15] Sinquin, B., Prengere, L., Kulcsár, C., Raynaud, H.-F., Gendron, E., Osborn, J., Basden, A., Conan, J.-M., Bharmal, N., Bardou, L., et al., "On-sky results for adaptive optics control with data-driven models on low-order modes," *Monthly Notices of the Royal Astronomical Society* **498**(3), 3228–3240 (2020).

[16] Hirst, P., Jenkins, D., and Sivo, G., "Adaptive optics telemetry as a science product for users," in [*Adaptive Optics Systems VII*], Schreiber, L., Schmidt, D., and Vernet, E., eds., **11448**, International Society for Optics and Photonics, SPIE (2020).

[17] Myers, R. M., Hubert, Z., Morris, T. J., Gendron, E., Dipper, N. A., Kellerer, A., Goodsell, S. J., Rousset, G., Younger, E., Marteaud, M., Basden, A. G., Chemla, F., Guzman, C. D., Fusco, T., Geng, D., Roux, B. L., Harrison, M. A., Longmore, A. J., Young, L. K., Vidal, F., and Greenaway, A. H., "CANARY: the on-sky NGS/LGS MOAO demonstrator for EAGLE," in [*Adaptive Optics Systems*], Hubin, N., Max, C. E., and Wizinowich, P. L., eds., **7015**, 52 – 60, International Society for Optics and Photonics, SPIE (2008).

[18] Bardou, Lisa, Gendron, Éric, Rousset, Gérard, Gratadour, Damien, Basden, Alastair, Bonaccini Calia, Domenico, Buey, Tristant, Centrone, Mauro, Chemla, Fanny, Gach, Jean-Luc, Geng, Deli, Hubert, Zoltán, Laidlaw, Douglas J., Morris, Timothy J., Myers, Richard M., Osborn, James, Reeves, Andrew P., Townson, Matthew J., and Vidal, Fabrice, "ELT-scale elongated LGS wavefront sensing: on-sky results," *Astronomy & Astrophysics* **649**, A158 (2021).

[19] Bardou, Lisa and Schulz, Christine, "ESO Science Archive - Wendelstein Laser Guide Star Unit Data Products." https://archive.eso.org/wdb/wdb/eso/wlgsu/form (2018). Accessed: 2022-07-02.

[20] Bardou, Lisa and Schulz, Christine, "Manual for the open loop LGS-AO data taken with elongated LGS in ELT geometry configuration." https://archive.eso.org/wdb/help/eso/WLGSU_CANARY_AO_Manual.pdf (2018). Accessed: 2022-07-02.

[21] Hirst, P. and Cardenes, R., "The new Gemini Observatory archive: a fast and low cost observatory data archive running in the cloud," in [*Software and Cyberinfrastructure for Astronomy IV*], Chiozzi, G. and Guzman, J. C., eds., **9913**, 531 – 539, International Society for Optics and Photonics, SPIE (2016).

[22] Pauls, T. A., Young, J. S., Cotton, W. D., and Monnier, J. D., "A data exchange standard for optical (visible/IR) interferometry," in [*New Frontiers in Stellar Interferometry*], **5491**, 1231–1239, International Society for Optics and Photonics (Oct. 2004).

[23] Pauls, T. A., Young, J. S., Cotton, W. D., and Monnier, J. D., "A Data Exchange Standard for Optical (Visible/IR) Interferometry," *Publications of the Astronomical Society of the Pacific* **117**, 1255–1262 (Nov. 2005). Publisher: IOP Publishing.

[24] Duvert, G., Young, J., and Hummel, C. A., "OIFITS 2: the 2nd version of the data exchange standard for optical interferometry," *Astronomy & Astrophysics* **597**, A8 (Jan. 2017). Publisher: EDP Sciences.

[25] Wells, D. C. and Greisen, E. W., "FITS-a flexible image transport system," in [*Image Processing in Astronomy*], 445 (1979).

[26] Chiappetti, L., Currie, M. J., Allen, S., Dobrtzycki, A., Pence, W. D., Rots, A., Shaw, R., and Thompson, W. D., "Definition of the Flexible Image Transport System (FITS) The FITS Standard Version 4.0: updated 2016 July 22 by the IAUFWG Original document publication date: 2016 July 22 Language-edited document publication date: 2018 August 13," tech. rep., Italian National Institute for Astrophysics (INAF) (2018).

[27] Van Rossum, G. and Drake, F. L., [*Python 3 Reference Manual*], CreateSpace, Scotts Valley, CA (2009).

[28] Conan, R. and Correia, C., "Object-Oriented Matlab Adaptive Optics toolbox," in [*Adaptive optics systems IV*], **9148**, 2066–2082, SPIE (2014).

[29] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van 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., and Oliphant, T. E., "Array programming with NumPy," *Nature* **585**, 357–362 (Sept. 2020).

[30] Greenfield, P., Droettboom, M., and Bray, E., "ASDF: A new data format for astronomy," *Astronomy and Computing* **12**, 240–251 (Sept. 2015).

[31] The HDF Group, "Hierarchical Data Format, version 5." https://www.hdfgroup.org/HDF5/ (1997-2022). Accessed: 2022-06-27.

[32] 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., Vand erPlas, 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., and Astropy Contributors, "The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package," *The Astronomical Journal* **156**, 123 (Sept. 2018).

[33] Arsenault, R., Madec, P.-Y., Hubin, N., Paufique, J., Stroebele, S., Soenke, C., Donaldson, R., Fedrigo, E., Oberti, S., Tordo, S., Downing, M., Kiekebusch, M., Conzelmann, R., Duchateau, M., Jost, A., Hackenberg, W., Calia, D. B., Delabre, B., Stuik, R., Biasi, R., Gallieni, D., Lazzarini, P., Lelouarn, M., and Glindeman, A., "ESO adaptive optics facility," in [*Adaptive Optics Systems*], Hubin, N., Max, C. E., and Wizinowich, P. L., eds., **7015**, 577 – 588, International Society for Optics and Photonics, SPIE (2008).

[34] Wizinowich, P., Acton, D. S., Shelton, C., Stomski, P., Gathright, J., Ho, K., Lupton, W., Tsubota, K., Lai, O., Max, C., Brase, J., An, J., Avicola, K., Olivier, S., Gavel, D., Macintosh, B., Ghez, A., and Larkin, J., "First light adaptive optics images from the keck II telescope: A new era of high angular resolution imagery," *Publications of the Astronomical Society of the Pacific* **112**, 315–319 (mar 2000).

[35] Aller-Carpentier, E., Dorn, R., Delplancke-Stroebele, F., Paufique, J., Andolfato, L., Dupuy, C., Fedrigo, E., Gitton, P., Jolley, P., Lilley, P., Louarn, M. L., Duc, T. P., Rakich, A., Reyes, J., Ridings, R., Woillez, J., Marchetti, E., Valles, M. S., Schmid, C., Hubin, N., Berger, J.-P., Quentin, J., Delabre, B.-A., McLay, S., and Pasquini, L., "NAOMI: a new adaptive optics module for interferometry," in [*Optical and Infrared Interferometry IV*], Rajagopal, J. K., Creech-Eakman, M. J., and Malbet, F., eds., **9146**, 405 – 413, International Society for Optics and Photonics, SPIE (2014).

[36] Gonté, F. Y. J., Alonso, J., Aller-Carpentier, E., Andolfato, L., Berger, J.-P., Cortes, A., Delplancke-Strobele, F., Donaldson, R., Dorn, R. J., Dupuy, C., Egner, S. E., Huber, S., Hubin, N., Kirchbauer, J.-P., Louarn, M. L., Lilley, P., Jolley, P., Martis, A., Paufique, J., Pasquini, L., Quentin, J., Ridings, R., Reyes, J., Shchkaturov, P., Suarez, M., Duc, T. P., Valdes, G., Woillez, J., Bouquin, J.-B. L., Beuzit, J.-L., Rochat, S., Vérinaud, C., Moulin, T., Delboulbé, A., Michaud, L., Correia, J.-J., Roux, A., Maurel, D., Stadler, E., and Magnard, Y., "NAOMI: a low-order adaptive optics system for the VLT interferometer," in [*Optical*

and Infrared Interferometry and Imaging V], Malbet, F., Creech-Eakman, M. J., and Tuthill, P. G., eds., **9907**, 539 – 550, International Society for Optics and Photonics, SPIE (2016).

[37] Kendrew, S., Hippler, S., Brandner, W., Clénet, Y., Deen, C., Gendron, E., Huber, A., Klein, R., Laun, W., Lenzen, R., Naranjo, V., Neumann, U., Ramos, J., Rohloff, R.-R., Yang, P., Eisenhauer, F., Amorim, A., Perraut, K., Perrin, G., Straubmeier, C., Fedrigo, E., and Valles, M. S., "The GRAVITY Coudé Infrared Adaptive Optics (CIAO) system for the VLT Interferometer," in [*Ground-based and Airborne Instrumentation for Astronomy IV*], McLean, I. S., Ramsay, S. K., and Takami, H., eds., **8446**, 2391 – 2399, International Society for Optics and Photonics, SPIE (2012).

[38] 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., and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods* **17**, 261–272 (2020).

[39] Lyman, R., Businger, S., and Cherubini, T., "Mauna Kea Weather Center seeing page." http://mkwc.ifa.hawaii.edu/current/seeing/ (Sept. 2021). Accessed: 2022-06-25.

[40] Pence, W., Xu, J., and Brown, L., "Fv: A new FITS file visualization tool," in [*Astronomical Data Analysis Software and Systems VI*], **125**, 261 (1997).

[41] Pence, W. and Chai, P., "Fv: The interactive FITS file editor." https://heasarc.gsfc.nasa.gov/ftools/fv/ (2020). Accessed: 2022-06-20.