AuroraX, aurorax-api (or pyaurorax?), and aurorax-asilib: a user-friendly auroral all-sky imager analysis framework

M. Shumko[1,2], B. Gallardo-Lacourt[1,2], A.J. Halford[1], E. Donovan[3], E.L. Spanswick[3], D. Chaddock[3], I. Thompson, and K.R. Murphy

[1]NASA's Goddard Space Flight Center, Greenbelt, Maryland, USA
[2]Universities Space Research Association, Columbia, Maryland, USA
[3]University of Calgary, Calgary, Alberta, Canada

Key Points:

- AuroraX is an online interface to visualize the aurora and calculate conjunctions
- aurorax-asilib is a companion Python package for detailed analysis of auroral all-sky imager data
- Together, these tools enable effortless end-to-end discovery and analysis of the aurora

Corresponding author: M. Shumko, `msshumko@gmail.com`

Abstract

Abstract

Plain Language Summary

1 Introduction

OUTLINE

- Brief history of ASIs and ASI arrays. Talk about why THEMIS ASI exists. Discuss CANOPUS? Linage.
- Breadth of possible science questions that can be answered with aurora image data.
- Problem: modern ASI arrays produce an immense volume of data.
- Why this software? Aurora ASI data formats very greatly, each with their own caveats. This centralized software package is maintained by the AuroraX team.
- Benefits: Maintained by the AuroraX team so it's usability is of paramount importance
- Reduce the barrier to entry into auroral physics. Reduce the technical requirements and enable rapid discovery of new science.
- Instead of case study results, larger statistical behavior will likely appear.
- remove the need for scientists needing to write duplicate code to use these popular missions. AS a result, this will enable scientists to dive right into the science and not need to know the details of data management (downloading and loading data, as well as applying routine data processing steps

The rapidly increasing amount of imager data, together with unique data formats, significantly burdens space physicists with mondale and duplicated software engineering tasks—download data, load and parse the data correctly, etc. This unnecessary burden can also lead to mistakes in analysis software that may require unnecessary troubleshooting time from the ASI team. The goal of AuroraX is to overcome these drawbacks by providing a set of robust tools that most researchers need to analyze all-sky images.

We describe our progress towards that goal in this article. First, we showcase the main features of the online AuroraX interface (https://aurorax.space/) such as the conjunction finder and the keogram finder. Second, we describe the aurorax-api, a Python library containing the interface to the AuroraX server to automatically download keograms and identify conjunctions. Third, we describe aurorax-asilib, the Python all-sky imager library. aurorax-asilib provides functions to the download, load, process data, and visualize THEMIS and REGO ASI data.

2 Design Philosophy (Principals?)

OUTLINE

- The primary design philosophy is to offer a robust set of functions that are useful for most researchers studying the aurora. We strived to strike a balance between complicated and user-friendly tools.
- Online keogram and conjunction interface accessible anywhere with internet connection.
- Comprehensive ASI data analysis functionality on a PC.
- Abstract away data management steps: downloading data, loading data, applying routine data processing steps, and common visualizations.

3 AuroraX

OUTLINE

- What is it?
- A highly optimized conjunction search
- On-demand keograms
- Virtual Observatory
- pyaurorax (aurorax-api) to directly access AuroraX services.
- Figure 1: a) a screenshot of the nightly keograms, b) screenshot of the conjunction search tool.

4 aurorax-asilib

OUTLINE

- What is it? A Python library that helps researchers analyze THEMIS and REGO ASI images.The main functions are summarized in Table 1. It is designed to be simple and runnable on personal machines (relatively low memory usage). We strived to strike a balance between complicated and user-friendly tools.
- A table of function names and one sentence to describe their functions.
- The large file sizes lead to relatively long processing time. This is a fact that can be partly mitigated by an SSD.
- Plug-in based architecture that allows new ASI arrays to be added and called by the core aurorax-asilib software.

aurorax-asilib allows researchers to analyze ASI data on a PC. It provides a set of functions for common data analysis tasks using ASI data. Here we overview the functions and the online documentation has more examples, a tutorial, and a thorough API reference `https://aurora-asi-lib.readthedocs.io/`

As we tour the asilib functions, keep in mind that asilib is designed to help you with the lower-level tasks. For example, if you want to load the image data via `asilib.load_image()`, asilib will attempt to download it if it is not already saved on the PC. Likewise, if you call `asilib.plot_keogram()`, it will automatically load (and optionally download) the ASI data for you. Lastly, Figs. 2-X were made using the code in a Jupyter Notebook that is provided with this article (in both the ipynb and pdf formats).

4.1 Download and load ASI image and skymap data

OUTLINE

- Handles the downloading and loading of ASI images. Main design principle: Ultimately, ASI image files consists of time stamps and images, so the asilib functions really only need to return that data
- Similarly with skymap calibration files
- If a file is already downloaded, you do not need an internet connection to work with the data

Let us start with the four functions that download and load ASI image and skymap data:

- `asilib.download_image()`,
- `asilib.download_skymap()`,

- `asilib.load_image()`, and
- `asilib.load_skymap()`

that are described below.

The `asilib.download_image()` function downloads the level 1 hourly Common Data Format (CDF) files from `http://themis.ssl.berkeley.edu/data/themis/thg/l1/asi/` for THEMIS, and `http://themis.ssl.berkeley.edu/data/themis/thg/l1/reg/` for REGO. The files are saved in the `asilib.config['ASI_DATA_DIR']` directory that at `~/asilib-data/` be default, and is customizable.

The `asilib.download_skymap()` function downloads all of the skymap files from `https://data.phys.ucalgary.ca/sort_by_project/THEMIS/asi/skymaps/` and `https://data.phys.ucalgary.ca/sort_by_project/GO-Canada/REGO/skymap/` for a given set of `asi_array_code` and `location_code`. The skymap files are in the Interactive Data Language (IDL) `.sav` file format. Noteworthy is that `asilib` downloads all of the skymap files for a given imager because the skymaps are only valid for a set time period (typically a year; see the logic for `asilib.load_skymap()` that is described below).

As the name implies, `asilib.load_image()` loads into memory and returns the ASI time stamps and images for a specified imager. This function loads both single and multiple images: a single time stamp and image if `time` is provided, and an array of time stamps and images if `time_range` is provided. As previously mentioned,`asilib.load_image()` will try to download an hourly CDF file if it does not exist locally.

`asilib.load_skymap()` is the last noteworthy input function; it loads the relevant skymap file into memory and returns the data in a dictionary. A relevant skymap file is the latest one before the specified `time`. As with `asilib.load_image()`, `asilib.load_skymap()` will attempt to download the skymap functions if they are not already downloaded.

Before we discuss the plotting functions, we emphasize that the `asilib.download_image()` and `asilib.download_skymap()` functions are often unnecessary to call since they are called by `asilib.load_image()` and `asilib.load_skymap()`. However, the download functions are very useful if you need to download ASI image and calibration data in bulk—useful to analyze data offline, for example.

4.2 Plotting single images

The `asilib` provides two ways to plot a single ASI image:

- `asilib.plot_fisheye()` and
- `asilib.plot_map()`.

One common way to visualize all-sky images is with `asilib.plot_fisheye()`. It plots the raw ASI images oriented with North at the top and East to the right of each image. The term fisheye comes from the fisheye lens that expands the imager's field of view to nearly 180°. For reference, the `azel_contours` keyword argument superposes contours of elevation and azimuth in the fisheye image. Figure 1a,c show an example of an auroral arc observed concurrently by the THEMIS and REGO ASIs stationed at Rankin Inlet (RANK). If you don't override the parameters, the color map is automatically chosen: black-to-white for THEMIS and black-to-red for REGO. Also the color scale is dynamically calculated using percentile logic described in the documentation.

Another common way to visualize images is by projecting the fisheye image onto a geographic map using `asilib.plot_map()`. asilib uses the skymap files to map each pixel's vertices to a (latitude, longitude) point at an assumed aurora emission altitude (typically 110 km for THEMIS and 230 km for REGO). Figure 1b,d show the fisheye im-

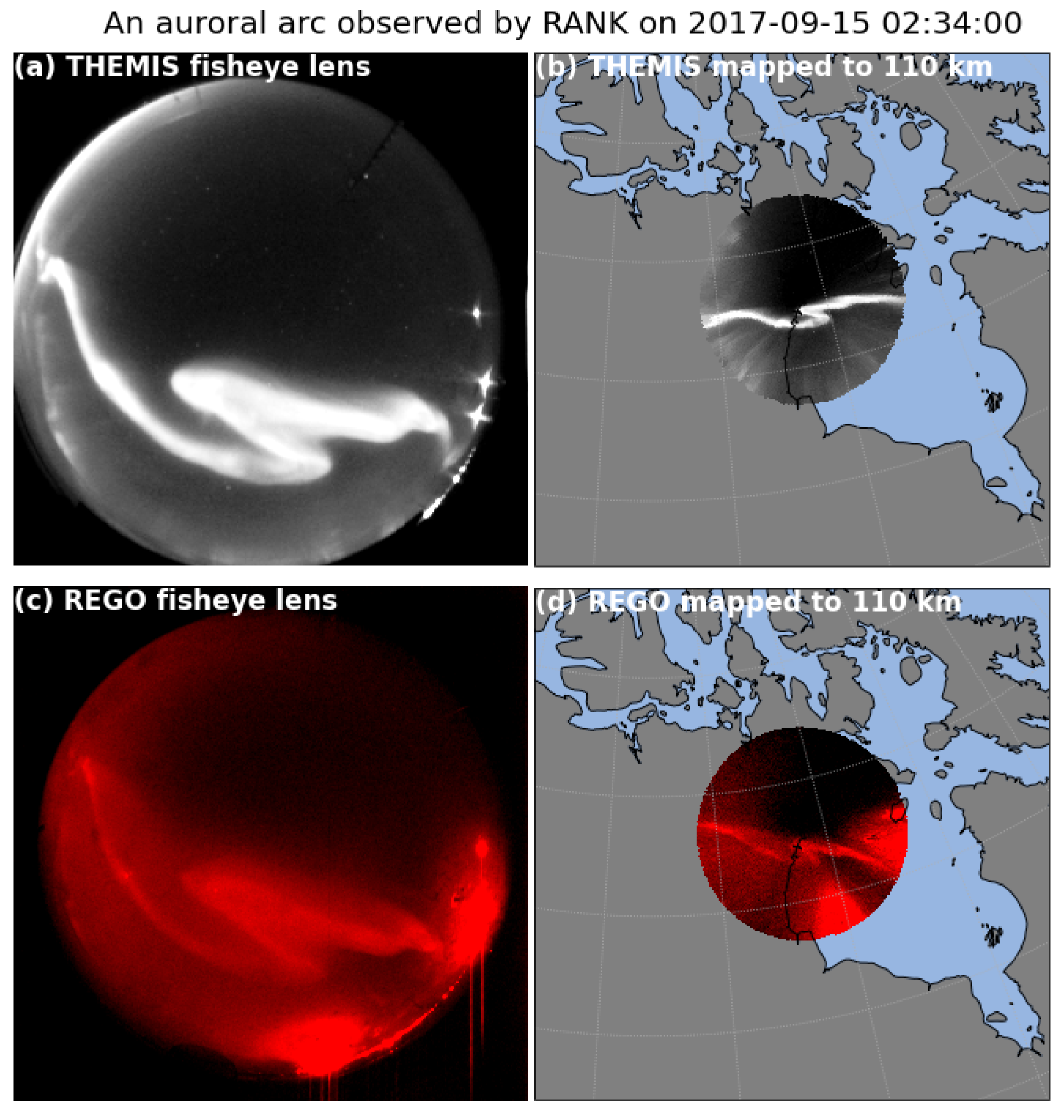

Figure 1. ASI image of an auroral arc taken simultaneously by the REGO and THEMIS imagers at Rankin Inlet, Canada. Panels and a and c show the raw fisheye lens view, while panels b and d show the same images projected to the 110 km assumed aurora emission altitude. Only the pixels with $> 10^\circ$ elevation are plotted.

ages mapped to 110 km altitude. By default, pixels that look at $< 10^\circ$ elevations are not mapped due to the stretching of pixels closest to the horizon. And lastly, `asilib.plot_map()` provides default map styles that can be overwritten by your custom `cartopy` map passed in via the `ax` keyword argument.

4.3 Keograms

A ubiquitous way to visualize ASI images and identify different types of aurora is with keograms. A keogram is a compression of many sequential ASI images into a single image showing pixel intensity as a function of latitude and time. Typically, they are assembled by looping over every image and slicing pixels that are oriented North-South through zenith (or through a custom path such a path of a stellite). Keograms are an

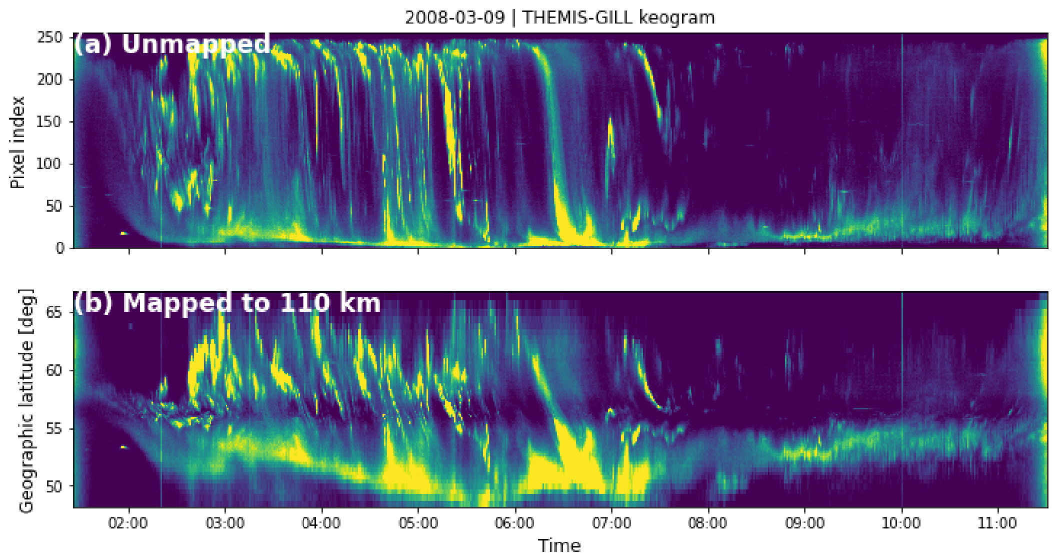

Figure 2. A full-night keogram showing the dynamic aurora observed at Gillam, Canada. Panel a shows the unmapped keogram with the pixel index vertical axis, while panel b shows the latitude of the pixels mapped to 110 km altitude.

essential tool that compress the information contained in hours of images into one plot. Objects in the sky such as auroral arcs, pulsating aurora, substorms, clouds, the moon, etc. all have unique keogram signatures that allow you to quickly classify what the imager observed.

You can make a keogram using the `asilib.plot_keogram()` function (that in turn calls `asilib.keogram()`). Similar to `asilib.plot_map()`, `asilib.plot_keogram()` takes an optional `map_alt` keyword argument. If it is not provided, the keogram's vertical axis is pixel index, but if a valid map altitude is provided, the vertical axis is geographic latitude. To minimize the PC's memory usage, `asilib.keogram()` loads image data using `asilib.load_image_generator()` that loads one image file at a time. The keogram shown in Fig. 2 shows the dynamic nature of the aurora. Furthermore, the latitude mapping transformation between panels a and b is substantial—low elevation pixels map to much wider sections of latitude as compared to the pixels near zenith.

4.4 Animating Images

You can easily animate ASI fisheye images using `asilib.animate_fisheye()`. It first saves png images in a unique subfolder in the `~/asilib-data/movies/images/` folder, and then animates them using the `ffmpeg` library. Movie S1 in the supporting information (SI) document shows an example of X type of aurora.

Animating just the fisheye images is somewhat limiting. Thus, `asilib` also includes `asilib.animate_fisheye_generator()` (which is technically a coroutine) to iterate over and pause after each ASI image to allow you to modify it. Then, after the iteration is complete, `asilib.animate_fisheye_generator()` combines the modified images into a movie. What sort of modifications can you make? For example, you can use it to superpose a satellite path and estimate the auroral intensity at its footprint during a conjunction. This is further described in the next two sections.

We mentioned earlier that the `asilib.animate_fisheye_generator()` is technically a coroutine. So what does that mean? Here, it means that before looping over each ASI image, you can request the raw data from `asilib.animate_fisheye_generator()` using the `.send('data')` method. You can use this to, for example, use the available ASI time stamps to appropriately sample the satellite location.

4.5 ASI analysis tools

OUTLINE

- lla2azel
- lla2footprint (requires IRBEM)
- area2pixels

Currently, `asilib` provides three analysis functions that are useful for conjunction analysis: `asilib.lla2footprint`, `asilib.lla2azel`, and `asilib.area2pixels`.

`asilib.lla2footprint` uses IRBEM-Lib (CITE; requires a separate installation and compilation of the Fortran source code) to trace a satellite's position along the magnetic field line using one of the magnetic field models that are supported by IRBEM. The primary use of this function is to map the satellite's location at, say, 500 km altitude, to the assumed auroral emission altitude (previously mentioned).

4.6 An example: a satellite-ASI conjunction

OUTLINE

- Combine everything above into an example showing where the footprint of a LEO satellite is, and what
- Figure 4: A conjunction montage and a time series
- (Implement an Imager.conjunction function)
- Reference movie S2

5 Quality Assurance

OUTLINE

- asilib on GitHub. unit and integration tests run automatically before every release.
- THEMIS and REGO data formats are set and won't change.

6 Future Development

OUTLINE

- Switch from CDF to pgm files.
- Support more ASI arrays such as TREx
- Community challenge. This package is made for the space physics community, so we will seek feedback (in the form of issues and bugs) that are discovered during a community challenge time period.
- Combine into a class architecture to share data. This is proving to be difficult to balance usability and maintainability.

While aurorax-asilib is feature complete, we plan to keep developing it.

7 Conclusion

OUTLINE

- AuroraX, aurorax-api, and aurorax-asilib tools provide the science community with a simple and a robust set of analysis tools
- Enable system-level science to be easily done
- Quickly sift through an immense volume of data to uncover new physics
- This is an end-to-end solution
- Plan to add support for other ASI arrays and satellites
- Help promote a uniform ASI data format for future cameras

Acknowledgments

We are thankful for the engineers and scientists who made AuroraX, THEMIS ASI, and REGO ASI projects possible. M. Shumko and B. Gallardo-Lacourt acknowledge the support provided by the NASA Postdoctoral Program at the NASA's Goddard Space Flight Center, administered by Universities Space Research Association under contract with NASA. The THEMIS and REGO ASI data is available from `https://data.phys.ucalgary.ca/`.