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]Oak Ridge Associated Universities, Oak Ridge, Tennessee, USA
[3]University of Calgary, Calgary, Alberta, Canada

Key Points:

- AuroraX and aurorax-api are online interfaces to quickly 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

COLOR GUIDE:

Red: General notes, issues, and TODOs

Blue: Questions that I believe Eric is best equipped to answer

Green: Questions that I believe Darren is best equipped to answer

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

Abstract

Here we introduce the AuroraX project which consists of open-source software designed to effortlessly discover and analyze auroral imager data. At this time AuroraX consists of three main tools: the AuroraX website (`https://aurorax.space/`); aurorax-api, a application program interface to the AuroraX website written in Python; and aurorax-asilib, an auroral all-sky imager library also written in Python. Together, these three tools enable a rapid and painless end-to-end analysis of the aurora. One such example is a conjunction study: AuroraX—or the aurorax-api—can be used to quickly identify conjunctions between numerous ground- and space-based instruments and quickly look at keograms from the available imagers to determine what the imagers observed. A list of conjunctions, together with aurorax-asilib, can be then used to download, load, and analyze all-sky imager data directly on a personal computer.

Plain Language Summary

The aurora is a ubiquitous light spectacle that has been observed in the polar regions for millennia. The regional scale of the aurora has led to the development of numerous imagers, and arrays of imagers, that continuously take images thorough the night. The image volume has quickly become unimaginable by any group of scientists, so the imager datasets are at best fragmented on the internet, and at worst hidden from the public. The AuroraX project aims to combine the publicly-available datasets into one easily accessible location. This project consists of three main tools: the AuroraX website (`https://aurorax.space/`); aurorax-api, a application program interface to the AuroraX website written in Python; and aurorax-asilib, an auroral all-sky imager library also written in Python. Together, these three tools make the auroral data easily accessible and to the public and the scientific community—so that we can all enjoy and wonder the beautiful spectacle of the aurora.

1 Introduction

Since the 1960s the space physics community has utilized optical ground-based instrumentation that has led to discoveries and from which great advancements in the field have been derided. Some of these discoveries include the early description of a substorm introduced by Akasofu (1964), and several descriptions of new phenomena that highlight the tight connections between the magnetosphere and ionosphere (e.g.,). Add citations and other discoveries/breakthroughs. These discoveries were possible with a multitude of instruments.

Meridian Scanning photometers (MSP), such as the Canadian Auroral Network for the OPEN (Origins of Plasmas in Earth's Neighborhood) Program Unified Study (CANOPUS) array, were first utilized for remote sensing the high-latitude ionosphere (e.g., Rostoker et al. 1985). These MSPs were later followed by two dimensional All-sky Imagers (ASIs) that took the space physics field into a deeper understanding of the ionosphere-magnetosphere coupled system. In the present day, the space physics community has carried a long legacy of ground-based optical instruments, with perhaps the THEMIS-ASI observatories being one of the most successful ones. Currently, many different optical arrays are in operation or are in development (e.g., THEMIS-ASI, REGO, MANGO, TREx, PWING, among others). These modern ASIs work continuously in high temporal and spatial resolution, where each camera produces thousands of images per night.

The rapidly increasing volume of imager data, together with unique data formats, significantly burdens space physicists with mundane and duplicated software engineering tasks—download data, load and correctly parse the data, etc. This unnecessary burden can also lead to mistakes in analysis software that may require unnecessary trou-

bleshooting time from the ASI team, or more worrisome—publish inaccurate findings that mislead researchers and the public. Thus, robust auroral analysis tools are required.

We introduce the AuroraX project that aims to overcome the above issues by providing three robust tools that most auroral researchers need. The first tool is the AuroraX website (https://aurorax.space/) that can quickly plot keograms, show what imagers operated at a given time, and calculate conjunctions between numerous ground- and space based instruments. The second tool is the aurorax-api, a Python library that interfaces with the AuroraX website to download the data to a personal computer and automate those tasks. The third tool is aurorax-asilib, a Python all-sky imager library. aurorax-asilib provides functions to the download, load, analyze, and visualize THEMIS and REGO ASI data.

2 AuroraX

Much of the text in this section is copied from `https://docs.aurorax.space/` and `https://docs.aurorax.space/python_libraries/pyaurorax/basic_usage/basics_intro/`

2.1 Website

The AuroraX website, located at (`https://aurorax.space/`) is designed to be the first place to start your auroral analysis. The primary objective of AuroraX is to enable mining and exploitation of existing and future auroral data, enabling key science and enhancing the benefits of the world's investment in auroral instrumentation. This is being accomplished with the development of key systems/standards for uniform metadata generation and search, image content analysis, interfaces to leading international tools, and a community involvement that includes more than 80% of the world's data providers.

What is the difference between Swarm-Aurora and AuroraX? Is Swarm-Aurora the legacy website? It is confusing to me as is. The AuroraX website contains three browser applications: the Swarm-Aurora Keogram Browser, a Conjunction Search Engine, and the virtual observatory.

The primary goal of the Keogram Browser is to provide an easy and scientifically productive tool for viewing summary data from ground-based auroral imaging data. The keogram explanation is duplicate. Remove either this one or the one in the asilib section. Keograms are an 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. To browse the keograms you'll need to select the date and one or more ASI arrays. Figure 1a shows the THEMIS ASI keograms from the available imagers as a function of time.

The Conjunction Search Engine is a tool for finding conjunctions between ground-based ASIs and space-based instruments. It helps researchers streamline the process of "event discovery", quickly narrowing down the possible times that may contain interesting data. There are a few ways to specify and search for conjunctions and the drop-down menus is the most intuitive. The menus include the start and end time, maximum distance between instruments, conjunction type (such as geographic or magnetic footprints in either hemispheres using SSCWeb for space-based instruments, and using AACGM for ground-based instruments), "criteria blocks" to specify what instruments to calculate the conjunctions of, and metadata filters to filter the results by weather and auroral conditions. Figure 1c shows an example of the conjunctions identified between the TREx ASIs and all space-based instruments. Clicking on one of the conjunctions leads to a detailed view that we show in Fig. 1b. This view shows a map with the ASI fields

of view and the satellite orbits superposed. Also, the keograms from the available ASIs are shown on the right. In one quick search can you identify, filter, and quantify the conjunction quantity and quality.

The Virtual Observatory aims at providing interactive visualizations and data browsing tools to quickly navigate the vast amount of data available. These web-based applications are geared towards identifying auroral events of interest using summary data of different types. The Virtual Observatory currently has two web interfaces available: Keogramist, and the Event Explorer. The Keogramist web tool is designed to allow for rapid "event discovery" using ASI keograms and movies. I found a bug in the movie player where the Play/Pause button is rapidly toggles without my input. Pressing it doesn't do anything. Lastly, the Event Explorer is a 3D visualization of the conjunctions. Add more info once complete.

2.2 aurorax-api

The AuroraX servers can also send data to a user's personal computer without an internet browser using a Python Application Program Interface (API) called aurorax-api. The `pyaurorax.sources` contains the DataSource class as well as functions for searching, retrieving, creating, and updating data sources in the AuroraX database. The `pyaurorax.ephemeris` module contains the Ephemeris class as well as functions for searching, creating, and updating ephemeris records in the AuroraX database. The `pyaurorax.data_products` module contains the DataProduct class as well as functions for searching, creating, and updating data product records in the AuroraX database. And lastly, `pyaurorax.conjunctions` contains the Conjunction class as well as functions for searching for conjunctions between ephemeris records in the AuroraX database.

Can you add a paragraph or two that describe how to accomplish a few common tasks using aurorax-api?

3 aurorax-asilib

The final component of AuroraX is aurorax-asilib, henceforth referred to (and imported) as asilib. It enables researchers to apply common data analysis tasks to the THEMIS and REGO ASI image data on a personal computer. Here we overview the main functions, while the online documentation has more examples, a tutorial, and a thorough API reference `https://aurora-asi-lib.readthedocs.io/`. Merge these docs with `https://docs.aurorax.space/`?

As we tour the asilib functions, keep in mind that asilib manages 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 your computer. Likewise, if you call `asilib.plot_keogram()`, it will automatically load (and download if necessary) the ASI data before plotting it. And for reference, Figs. 2-4 were made using the code in a Jupyter Notebook that is provided as supplemental material in both the ipynb and pdf formats.

3.1 Download and load ASI image and skymap 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()`

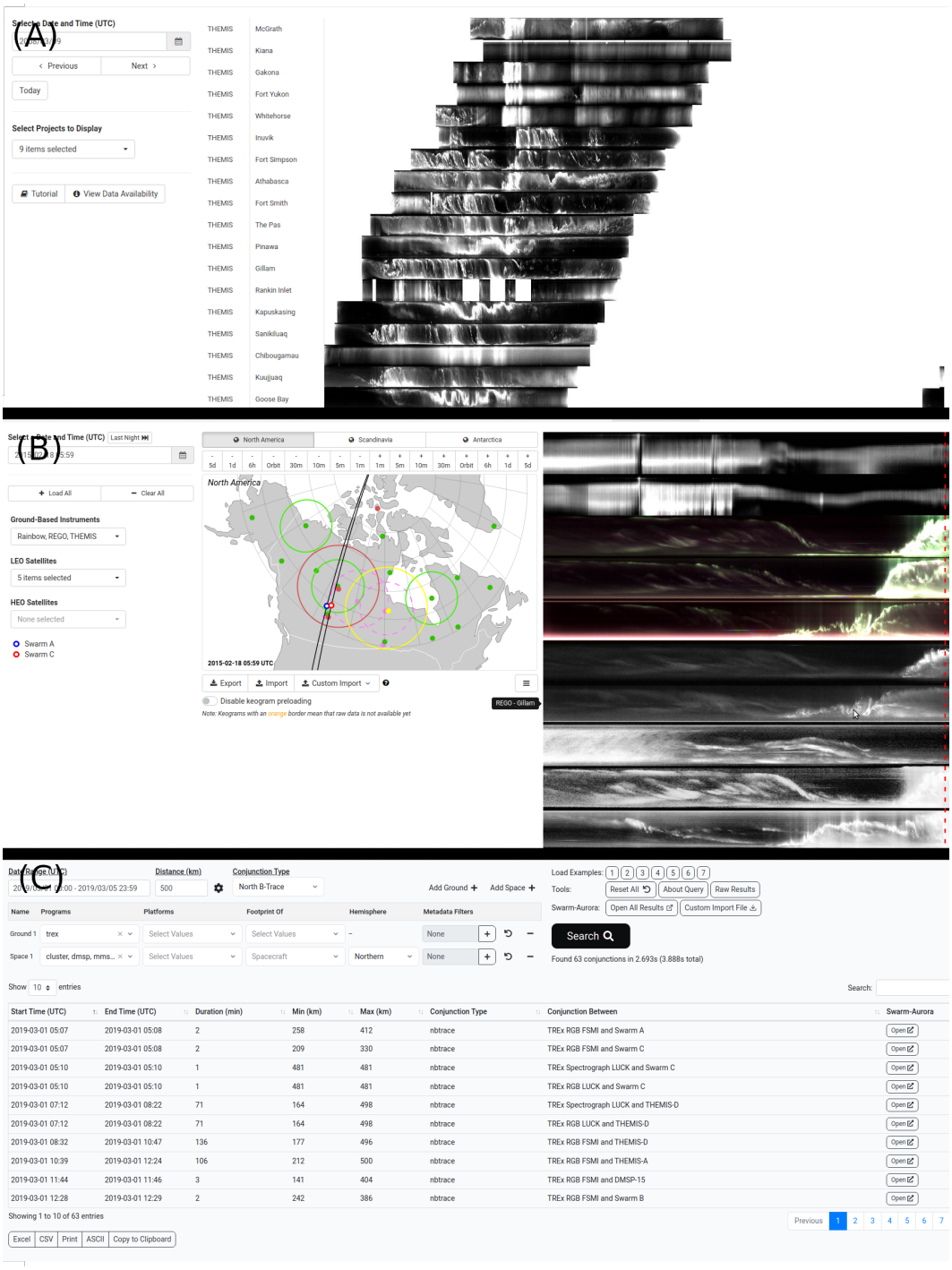

Figure 1. The `https://aurorax.space/` interface. Panel a shows the daily keogram browser. Panel b shows the conjunction finder with keograms. And lastly, panel c shows the conjunctions in list form. Switch panels b and c?

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. TODO: Switch to PGM.

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). Lastly, asilib converts the longitude values from $[0°, 360°]$ to $[-180°, 180°]$.

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 files 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.

3.2 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°. Figure 2a,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 aurora emission altitude (typically assumed 110 km for THEMIS and 230 km for REGO). Figure 2b,d show the fisheye images mapped to 110 km altitude. By default, pixels that look at $< 10°$ elevations are not mapped due to nearby obstructions and the stretching of pixels closest to the hori-

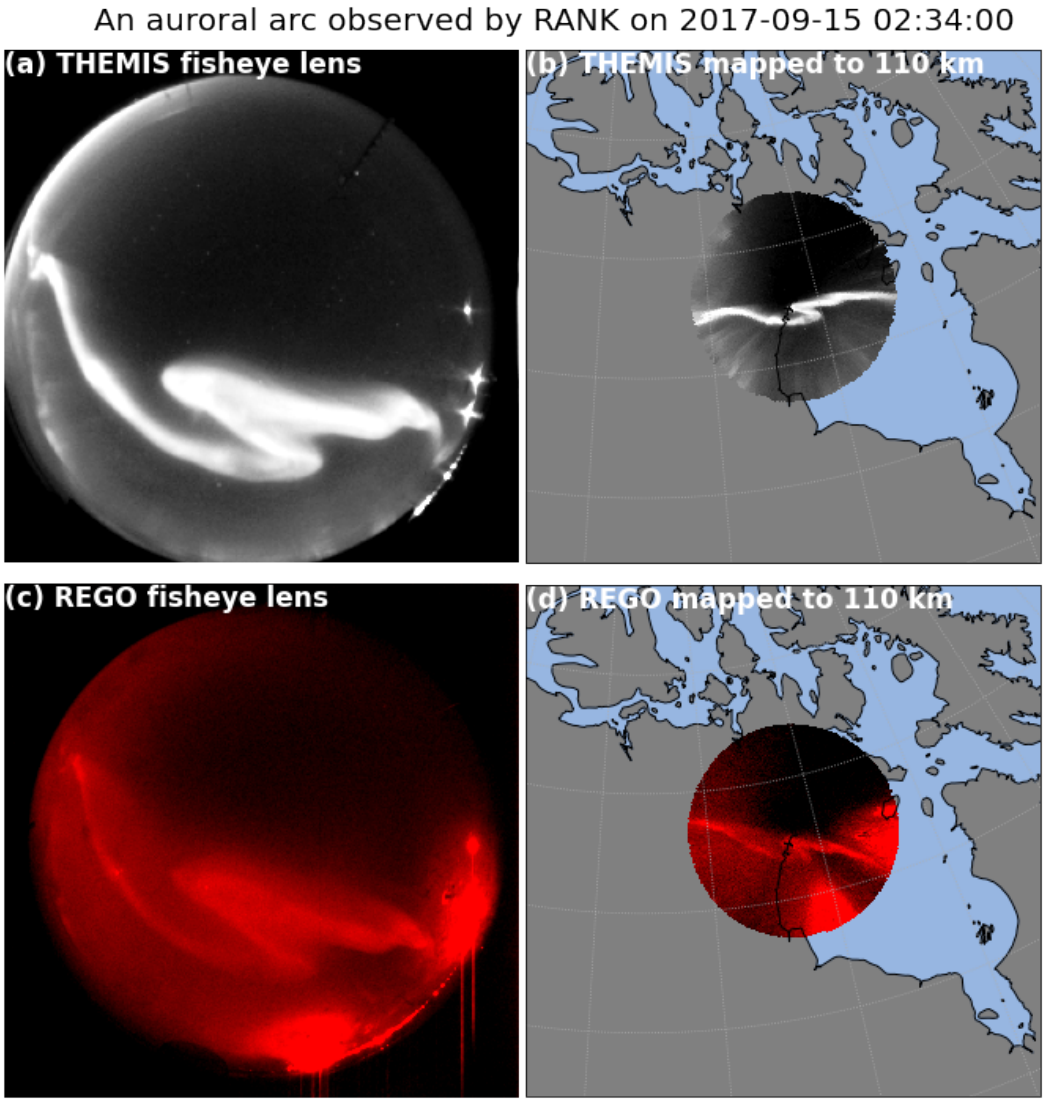

Figure 2. ASI image of an auroral arc taken simultaneously by the REGO and THEMIS imagers at Rankin Inlet, Canada. Panels and a and c show the raw fisheye lens view, while panels b and d show the same images projected to the 110 km assumed aurora emission altitude. Only the pixels with $> 10^{\circ}$ elevation are plotted. The THEMIS skymap appears to be incorrect.

zon. And lastly, `asilib.plot_map()` provides default map styles that can be overwritten with a custom `cartopy` cite map passed in via the `ax` keyword argument.

3.3 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 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,

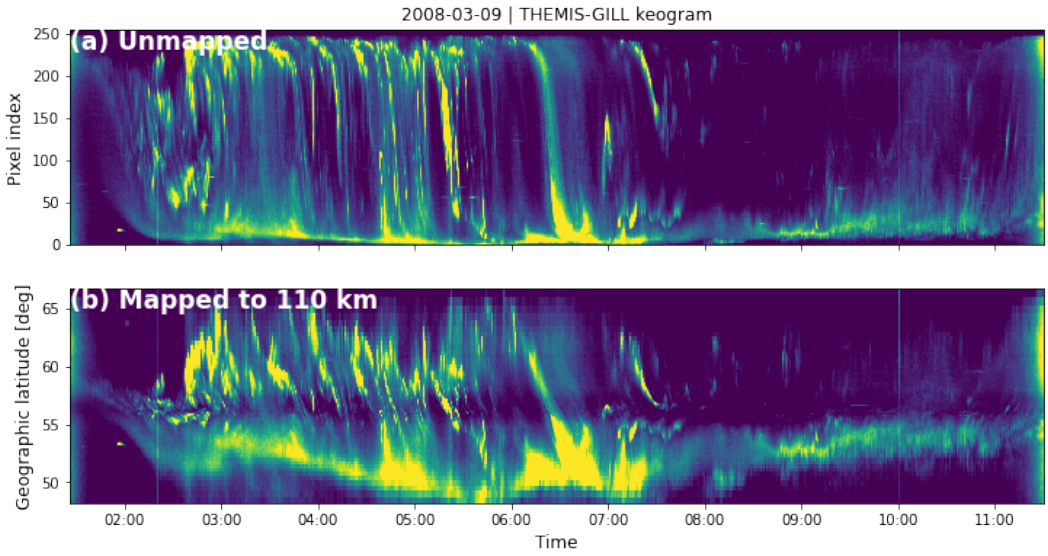

Figure 3. A full-night keogram showing the dynamic aurora observed at Gillam, Canada. Panel a shows the unmapped keogram with the pixel index vertical axis, while panel b shows the latitude of the pixels mapped to 110 km altitude. Add an AACGM keogram as well?

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. Also, the `path` keyword argument allows you to make a keogram along a custom geographic path.

To minimize 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. 3 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.

3.4 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 plotting each image, to allow you to modify the movie frame. 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.

As the names imply, `asilib.animate_map()` and `asilib.animate_map_generator()` animate the projected images on a map. You can use them as easily as the `animate_fisheye` functions.

We mentioned earlier that the `asilib.animate_fisheye_generator()` is technically a coroutine. So what does that mean? In this context, it means that before looping over each ASI image, you can request the images and time stamps from `asilib.animate_fisheye_generator()` using the `.send("data")` method. This approach appears rather contrived, however, a major advantage of this design is that the user sees exactly what ASI data the generator function will loop over. This is useful, for example, if you need the ASI time stamps to appropriately sample the images, plot the satellite location, and calculate the mean intensity of the aurora along the satellite trajectory and superpose it on the movie frames. We will discuss this application in detail later.

3.5 ASI analysis tools

Currently, `asilib` provides three functions that are useful for analyzing conjunctions: `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, in (latitude, longitude, altitude) (LLA) coordinates, along a magnetic field line. This field line is defined using one of the magnetic field models that are supported by IRBEM. The primary use of this function is to map a low Earth orbiting satellite's location at, say, 500 km altitude, to its magnetic footprint at the assumed auroral emission altitude, e.g. 110 km for THEMIS or 230 km for REGO as previously mentioned.

The next function is `asilib.lla2azel()`. It maps the satellite's footprint location, in LLA coordinates, to the ASI's (azimuth, elevation) coordinates (AzEl) using the skymaps. This function returns both the AzEl coordinates as well as the corresponding pixels in the image.

Lastly, `asilib.area2pixels()` calculates a pixel mask of pixels inside an auroral emission area. This function is useful to calculate the mean ASI intensity (or another statistical method) using pixels that map to a physical area in the sky. The mask contains 1s inside of the area and `numpy.nan` outside of it. You then multiply the image with the mask: the pixel intensities outside of the area are then `numpy.nan` and unchanged inside the area. We chose to use `numpy.nan` to ensure that the mean of the intensity is correctly applied—it will fail if you call `numpy.mean(image*mask)`, but `numpy.nanmean(image*mask)` will ignore NaNs and correctly calculate the mean intensity inside the area)

By default, the area is 5x5 km. If the area is so small that not even one pixel is contained inside it (but it is above the horizon), `asilib.area2pixels()` will slowly increase the area tolerance until at least one pixel is found. Lastly, if the area is completely outside of the skymap, `asilib` will raise a warning and the corresponding mask will be all `numpy.nan`.

3.6 An example: a satellite-ASI conjunction

In this example we combine the aforementioned analysis functions to calculate the mean auroral intensity at the an imaginary satellite's footprint during a conjunction with a THEMIS ASI. The satellite orbits at a 500-km altitude low Earth orbit. We will calculate and the mean ASI intensity in a 20x20 km area at a 110 km altitude. Lastly, we will animate the conjunction.

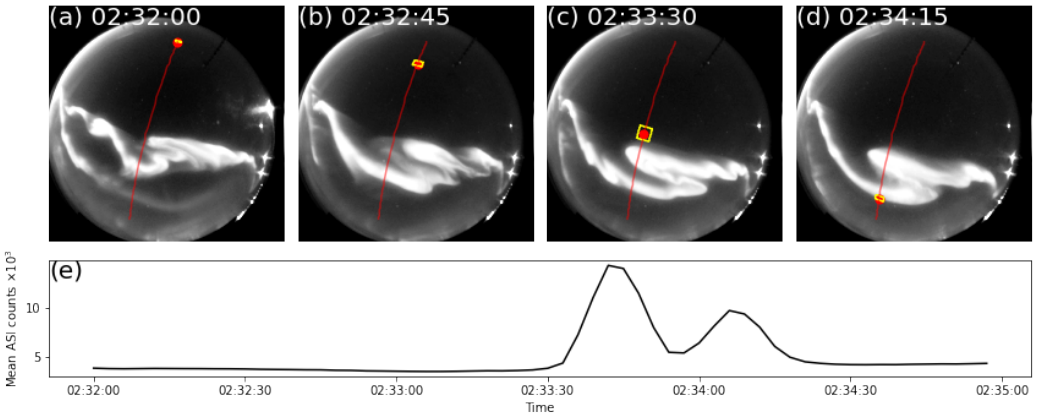

Figure 4. A conjunction montage of Movie S2. Panels a-d shows the auroral arc evolution. The red line is the satellite track and the red dot is its instantaneous position. The yellow quadrilateral bounds the pixels inside the a 20x20 km area surrounding the satellite's 110 km altitude footprint. Lastly, panel e shows the mean ASI intensity, as a function of time, inside the yellow quadrilaterals. When the satellite passed through the arc between 2:33:30 and 2:34:15, the mean ASI show two corresponding intensity enhancements.

For this example we use the satellite's location in LLA coordinates with time stamps that line up with the ASI times. In reality, the satellite and ASI time stamps are unlikely to line up, so you'll need to interpolate the satellite locations to the ASI time stamps.

Our analysis consists of three main steps. Step 1: we use `asilib.lla2footprint()` to trace the satellite's position along the magnetic field line to 110 km. Step 2: we use `asilib.lla2azel()` to find where in the imagers field of view (azimuth and elevation) the satellite's footprint was. Lastly, Step 3: we use `asilib.area2pixels()` and `numpy.nanmean()` to identify the image pixels in the 20x20 km area around the footprint. We then use these pixels to calculate the mean ASI intensity as a function of time (and satellite position). Don't worry if this is hard to follow—these steps are implemented in the "Figure 4" section of the `asilib_figures.ipynb` notebook.

Animation Or call them movies? Need to be consistent. S2 shows the result of this conjunction analysis and Fig. 4 shows a five-frame montage summarizing the animation. Fig. 4a-d show the fisheye lens images at the annotated time stamps. The complete satellite footprint path is represented by the red line and the instantaneous footprint by the red dot. The yellow areas around the footprints bound the 20x20 km area. Lastly, Fig. 4e shows the mean ASI intensity time series for the conjunction—it clearly shows the signature of the auroral arc between 2:33:30 and 2:34:15.

4 Quality Assurance

We developed AuroraX with useability and maintainability at the forefront. The source code for the the tools described here is open source and hosted on GitHub. It is archived on Zenodo.

We developed `asilib` with useability and maintainability at the forefront. The Python source code is available on `aurora-asi-lib` GitHub repository `https://github.com/mshumko/aurora-asi-lib`. It is also archived on Zenodo and the Python Packaging In-

dex (PyPI). On GitHub you can submit an Issue for bugs or feature requests, and you can contribute to `asilib` using a Pull Request.

Since documentation is critically important to the survival and usability of software, `readthedocs.org` hosts the `aurora-asi-lib` documentation at `https://aurora-asi-lib.readthedocs.io/`. It contains installation instructions, examples, a comprehensive tutorial, and a detailed Application Programming Interface (API). The documentation source code is in the `aurora-asi-lib/docs/` folder, useful in case a user needs to recreate the documentation on their computer with the Sphinx Python document generator.

To ensure code stability, the `asilib` source code includes a few dozen unit and integration tests that you can run locally, and are automatically run on GitHub Actions every time a change is pushed to the repository. These tests test the main library functions and will quickly warn us if a proposed change broke anything.

Furthermore, we use semantic versioning and a `CAHNGELOG.md` to communicate all major, minor, and patch changes with the community. We made many backward-incompatible changes before version 1.0.0, but henceforth we will indicate future backward-incompatible changes with major releases (i.e. version X.0.0).

Lastly, the data format is integral to `asilib`. The REGO and THEMIS data formats are fixed and are guaranteed to not change in the future.

5 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
- Add a paragraph to discuss the future work for this library includes...

Hopefully we made a convincing case that AuroraX and aurorax-asilib are a useful set of tools to analyze the aurora. Our aim is to keep the implementation as simple as possible: enough for beginners who are starting to use this data, while also enabling sophisticated analysis.

AuroraX can be used anywhere with an internet connection. It is simple to use and it allows us to quickly identify times of interest (quickly as in during a lunch with collaborators in the middle of a conference).

aurorax-asilib, on the other hand, is designed to minimally use the internet. It only needs it to download the data. Once saved locally, aurorax-asilib allows you to easily make common aurora plots, and provides you with functions to get you the raw data if you need to implement something more sophisticated. We hope that the conjunction example demonstrates how useful aurorax-asilib functions are to transform your data and compare to the images.

Acknowledgments
We are thankful for the engineers and scientists who made the 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 Oak Ridge Associated Universities under contract with NASA. Other funding sources/acknowledgments.

Open Research

Code repositories: AuroraX website is located at: AuroraX URL, the aurorax-api is at `https://github.com/aurorax-space/pyaurorax` and aurorax-asilib is at `https://github.com/mshumko/aurora-asi-lib` Migrate asilib to aurorax-space GitHub org. The in-depth documentation and examples are at `https://docs.aurorax.space/`.