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

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.,).

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 under development (e.g., THEMIS-ASI, REGO, MANGO, TREx, among others). These modern ASIs work continuously in high temporal and spatial resolution, where each camera produces 7.2k images per night for which robust analysis tools are required.

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 con-

junction 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?)

I don't think that this section is necessary. 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

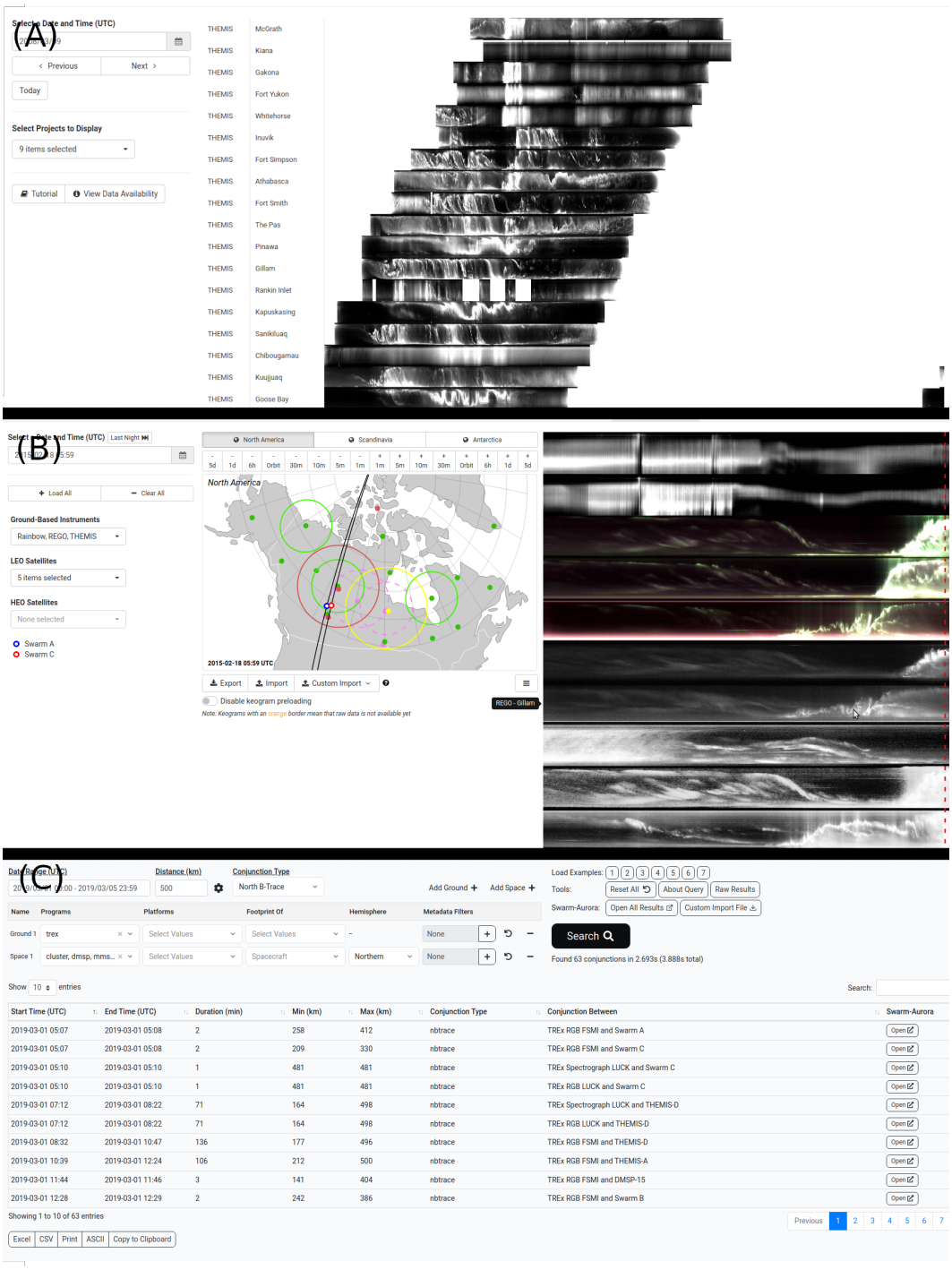

Figure 1. The `https://aurorax.space/` interface. Panel a shows the daily keogram browser. Panel b shows the conjunction finder with keograms. And lastly, panel c shows the conjunctions in list form.

call `asilib.plot_keogram()`, it will automatically load (and optionally download) the ASI data for you. Lastly, Figs. 2-4 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 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 assumed aurora emission altitude (typically 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 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 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. 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.

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.

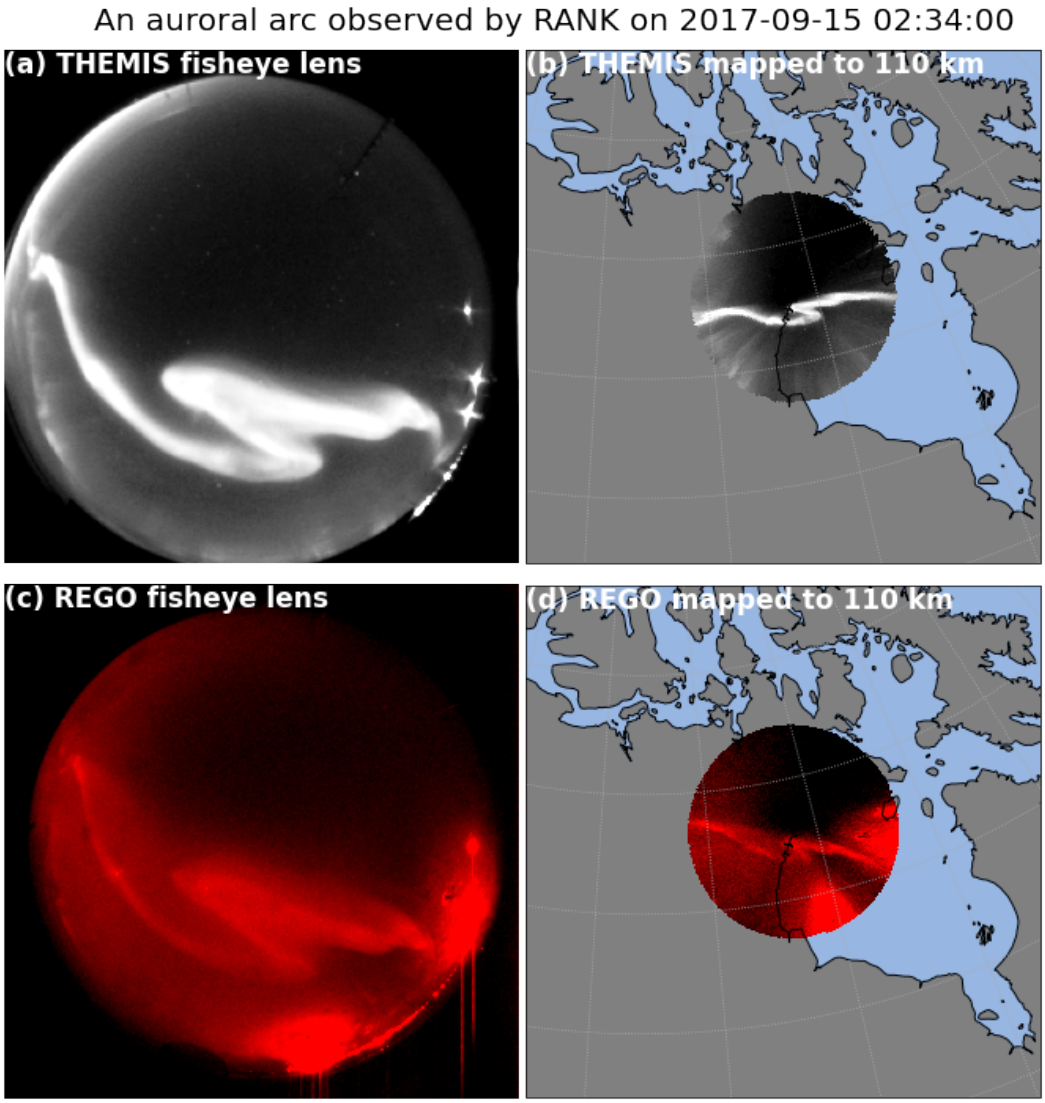

Figure 2. ASI image of an auroral arc taken simultaneously by the REGO and THEMIS imagers at Rankin Inlet, Canada. Panels and a and c show the raw fisheye lens view, while panels b and d show the same images projected to the 110 km assumed aurora emission altitude. Only the pixels with $> 10^{\circ}$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

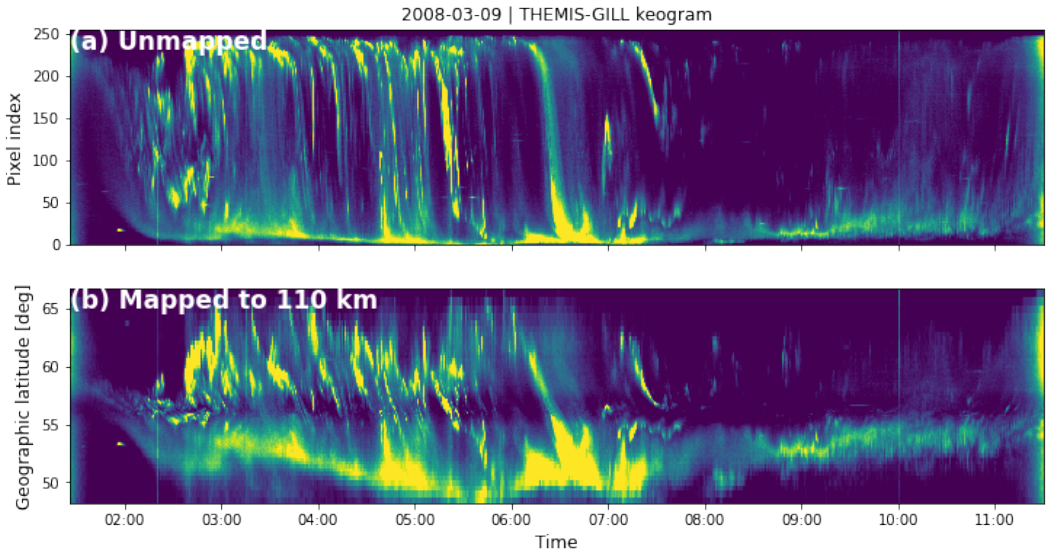

Figure 3. A full-night keogram showing the dynamic aurora observed at Gillam, Canada. Panel a shows the unmapped keogram with the pixel index vertical axis, while panel b shows the latitude of the pixels mapped to 110 km altitude.

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? In this context, 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. 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 and plot the satellite location and calculate the mean intensity of the aurora. We will discuss this application later.

4.5 ASI analysis tools

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, 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 `FULL_AZIMUTH` and `FULL_ELEVATION` skymap arrays. This function returns both the AzEl coordinates as well as the corresponding pixels in the image.

And the last analysis function that we describe here is `asilib.area2pixels()` that calculates a pixel mask of pixels inside an 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 has 1s inside of the area and `numpy.nan` outside of it. You multiply the image with the mask: the pixel intensities outside of the area are then `numpy.nan` and unchanged in 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)

If you give it one LLA coordinate, the mask will have the same shape as the ASI image. On the other hand, if you pass multiple LLA coordinates, the mask's will have a new first dimension corresponding to the number of coordinates.

By default, the area is 5x5 km. Think about how this function works for LLA values outside of the ASI's field of view. Maybe have a check for LLA values are completely outside of lat map or lon map? Also raise a warning if some of the 2d mask arrays are all nans. 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.

4.6 An example: a satellite-ASI conjunction

In this example we combine the aforementioned analysis functions into an example of a conjunction between a THEMIS ASI and a fictional satellite in 500-km altitude low Earth orbit. We will calculate and show the mean ASI intensity in a 20x20 km area, at 110 km altitude, around the satellite footprint and animate it.

First, to make this plot we use the satellite's location in LLA coordinates and time stamps. Here we use an imaginary satellite trajectory. 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.

We proceed with the analysis with three main steps. In step 1 we map the satellites position to 110 km using `asilib.lla2footprint()`. In step 2, we calculate where in the sky (azimuth and elevation) the satellite's 110 km footprint was as a function of time using `asilib.lla2azel()`. Lastly, in step 3 we calculate the pixels contained in a 20x20 km area at 110 km and calculate the mean ASI intensity in it using `asilib.area2pixels()` and `numpy.nanmean()`. Don't worry if this is hard to follow. These steps are implemented in the "Figure 4" section of the `asilib_figures.ipynb` notebook.

Movie S2 shows the result of this conjunction analysis. Also, for a paper-friendly way to visualize the conjunction, Fig. 4 shows a five-frame montage. Fig. 4a-d show the fisheye lens image at the annotated time stamp with the full satellite footprint path represented by the red line and the instantaneous footprint by the red dot. The yellow area shows what pixels are contained in a 20x20 km area surrounding the footprint. 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.

5 Quality Assurance

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/`

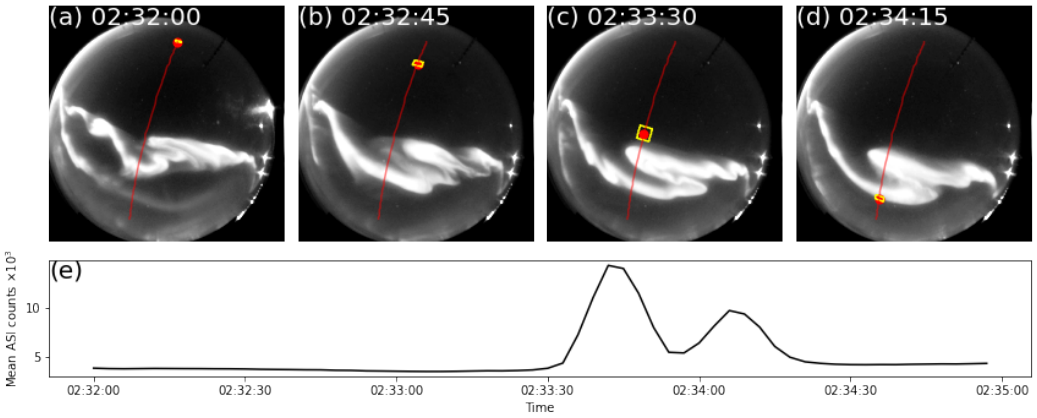

Figure 4. A conjunction montage of Movie S2. Panels a-d shows the auroral arc evolution. The red line is the satellite track and the red dot is its instantaneous position. The yellow quadrilateral shows the pixels contained in a 20x20 km area at a 110 km altitude. Lastly, panel e shows the mean ASI intensity, 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 ASI counts were enhanced.

`mshumko/aurora-asi-lib`. It is also archived on Zenodo and the Python Packaging Index (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.

6 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 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/`.