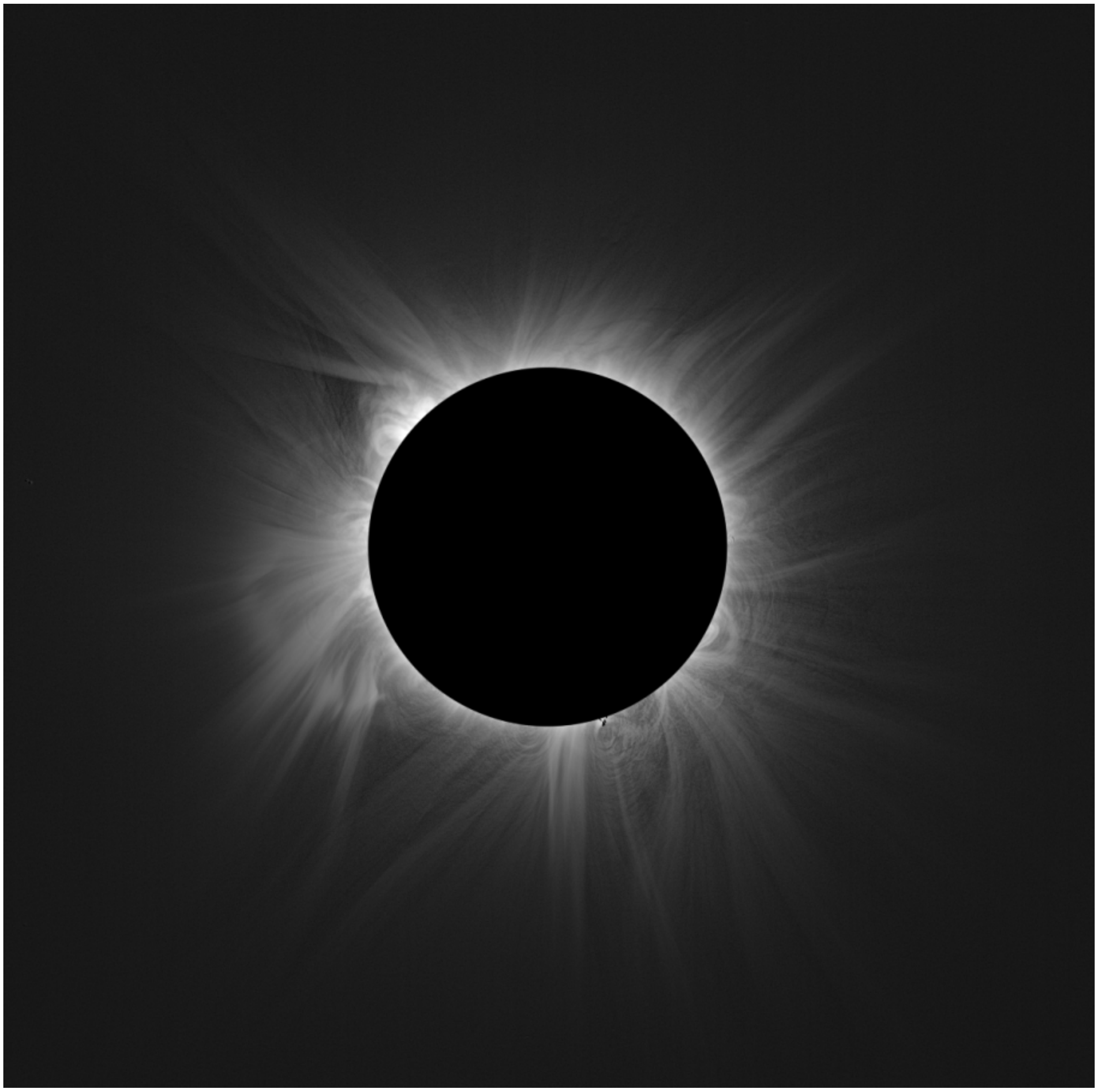
# ACES Data Pipleline Guide



**2026**

## Table of Contents

***Overview*** **.......... *3***

***Code*** **.......... *4***

**ACES Analyzer .......... 4**

**ACES Processor .......... 5**

**ACES Tool Kit .......... 5**

Camera_Log_Interpolation .......... 5

Scan_Selector .......... 6

Sampler .......... 6

ACES_Interpolator .......... 6

ACES_Transformer .......... 7

**ReadSeqMod Ganged .......... 7**

**ACES Camera Data .......... 7**

Time_Stamper .......... 7

Frame_Count_Fixer .......... 7

ACES_Camera_Data .......... 7

**ACES Darks .......... 8**

**ACES Log Data .......... 8**

**ACES Visualizer .......... 8**

***Data Notes*** **.......... *9***

***Future Work*** **.......... *9***

## Overview

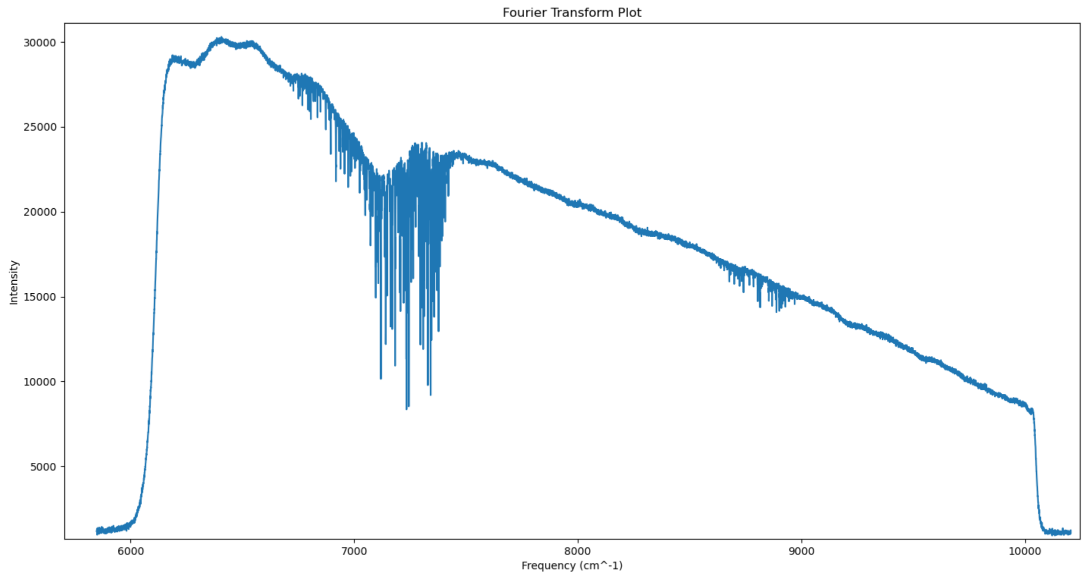
This code was written in late 2024 through 2025 for analysis work performed on a prototype version of the ACES instrument. The purpose of this code is to read in camera files and log files taken from ACES, process the data, and convert the intensity data into spectra for each pixel in the files. This code was mostly experimental and needs some work to be brought into full pipeline status for ACES data processing but should at least serve as a start point on how to do some of the basic tasks required for ACES data processing. Note that this code was written by a scientist/engineer, not a software engineering.

The code is comprised of several different python scripts. Each one of them contains several python functions used for a different step of data processing. Each script and the important functions are summarized in the sections below.

Data should be placed at a determined location for processing. This code assumes that there will be a set directory containing all datasets that you can define in the code. Each data directory should be setup to have a Camera directory containing the .seq files and a directory called Logs holding the log files. So there would be something like */PATH2ACES/Data/20260112/Camera* and */PATH2ACES//Data/20260112/Logs* to start processing a datafile.

A version of the code from early 2026 is hosted on GitHub here.

Included with this code is code to control the scanning mirror. This should be saved on the ACES computer, but a version is included here as well if needed.



# Code

## ACES Analyzer

This is the high-level script that was used to import all the other scripts and call them in the correct order. If written into a true pipeline, work would need to be done to organize this file correctly to run from a single call. At the current state, blocks of code were simply copied from this file and run in modular sections.

First, this code imports a bunch of other functions from the rest of the python scripts.

It then calls the main processing function, ACES_Processor. This returns processed data file arrays, info from the log files, a list of meta objects for the meta data, and the directory that was selected to work in.

Next are a set of scripts used to visualize the data to determine what kind of binning and subframe selection should be applied.

Once a binning mode and data subsection is determined, there are scripts to bin and cut the data according to these values. This is done on a dataset-by-dataset basis, but for the final pipeline set values should be determined and coded in.

Data can now be saved using the function to save as a .h5 and .pkl file. This is useful to avoid needing to read in .seq files in the future and just start back with the binned and processed data. But the file format used here can be modified if preferred. There is a block of code for skipping the earlier steps and just restoring datasets from this point.

After that is the Scan Selector function. This will plot the data on a position vs. time plot. After setting the desired scan length, it will then highlight the data on the plot that will be used for the given path length. It has an option to remove scans if needed, especially if the first or last scan isn't a full dataset.

The next steps save the measured position and intensity arrays to preserve as the 'real data' as the next steps will involve interpolation of the data into evenly spaced samples. It is helpful to maintain the original data here.

To get the evenly sampled data needed for Fourier Transforms, the code uses interpolation. First a sample size is set to determine the space between each sample. This is saved in both mm and cm. The code uses 1/3 of a HeNe wavelength as the default sample size right now. It then used the function ACES_Interpolator to do the interpolation and returned the evenly space dataset.

With the evenly spaced data, a Fourier Transform can be performed. This is done via the ACES_Transformer function which utilizes the python packages for Fourier Transforms. It returns the values of the FT and the frequency list assuming you provided the data sample size.

There is then code to only display the FTs over a selected wavelength, such as the one matching the camera taking the data. This data can then be plotted, modified, and analyzed with various blocks of code at the bottom.

## ACES Processor

This is the workhorse program for converting log and .seq files into data that can be read with Python. The script is setup to be run and give the user the opportunity to select which dataset will be processed and what will be done with it.

NOTE: The number of 'ganged' images on a single camera image is hard coded here. Currently 4 files are read out on a single image from the camera to improve the cadence. The base data location is also hard coded. If either change you will need to update these lines of code.

Once the dataset is selected, it will first read in the logs with ACES_Log_Data. Next it will read in the data with the function ACES_Camera_Data. This will read the .seq files and return the meta and data arrays. It will then call ACES_Darks to perform a dark subtraction on the data, assuming a dark file exists.

It will use the Camera_Log_Interpolation function to assign a positional value to each frame based on the log files. This positional value will be added to the meta object of each file. This processed data will then be saved under a directory called Processed.

It will return the data, meta, logs, and the directory being used when it is called.

It also has the option to restored saved processed data or overwrite datasets.

## ACES Tool Kit

This python script is comprised of a bunch of different functioned used in the pipeline. These are imported by various files as needed. The major functions are summarized below. There are other functions, but they are relatively straight forward.

### Camera_Log_Interpolation

This is the function that takes in the data, meta, and log files and appends a position to each frame's metadata based on the log data. It reads in the log positions from the correct value and also reads in the frame count values. It then uses interpolation between the camera frame counts and the log frame counts to assign a position to each individual frame. This data is appended to the meta data.

Note that the log updates less frequently than the camera reads out files, so interpolation is needed to get unique positions for each frame.

## Scan_Selector

Scan selector uses the positional values from the meta data and a given scan length to determine what data to use and what to ignore. It will select data with the correct distance from the center of the scan and ignore the rest of the data outside that range. It will then return the indices of data to use.

## Sampler

This function sorts the camera positional and real data arrays by position then creates an evenly sampled version of the data via interpolation. Based on the path of travel and the set sample size, it will create an evenly spaced array of data and interpolate the intensity values to match the data sample size.

For white light data, it assumes that the position of zero path length difference (ZPD) is where the data is the brightest. Since ZPD is harder to determine with laser data (peaks every integer wavelength rather than just at ZPD) it just finds the average middle position for the data and calls that ZPD.

NOTE: To account for difference across the camera sensor, each pixel has a unique ZPD determined. A minor angle relative to the path of light was shown to impact the position of ZPD for each pixel, so this was found to actually matter.

Once it has a sorted array for each scan in Optical Path Difference (OPD) space, it does the interpolation part using numpy's interp function to get interpolated intensity data for the evenly sampled positional array. Note the factor of two for converting to OPD, since OPD is the distance from the ZPD location times two.

It will return the original position array in OPD space (OPD), the interpolated positional data in OPD space (OPD_sample), and the interpolated intensity data (sample_data)

## ACES_Interpolator

This is the function that actually calls the Sampler function. It takes in the scan_list indices, the positional data, the real intensity data, the sample size, the path length, and if white light or laser light is being analyzed.

It will then run Sampler for each scan and build a series of arrays to return for each scan.

### ACES_Transformer

This is the function that takes in the evenly spaced sample data and performs a Fourier Transform on it using the python rfft function. For a dataset, it will run over each scan separately. For each pixel provided it will perform a Fourier Transform and return the results including both the FT list and the Frequency list. Be sure to verify the units used in the frequency calculations.

## ReadSeqMod Ganged

This is code that was previously written and modified to work with ganged image files, where multiple images are contained on each individual frame. It reads the files, splits it into individual images, and read the meta data contained at the bottom of each file.

## ACES Camera Data

### Time_Stamper

This function was created to assign a UTC time to each individual data frame. While this code worked, using UTC to align with the log files was found to be inaccurate. To avoid this issue, camera frame is given a positional value from the log based on the more accurate frame count rather than the UTC time. A time stamp is still applied in the code, but it is not used for processing purposes.

### Frame_Count_Fixer

This function is used as there are sometimes error with the frame count reset in the log files. It may run a few times before the reset takes place. So the first few frames read out may start from wherever the last run left off before resetting to zero. This function corrects that and reassigns all frame counts correctly.

### ACES_Camera_Data

This is the main function for reading in ACES data from the .seq files. It grabs all of the .seq files from the selected Camera directory and uses ReadSeqMod_Ganged to convert into data arrays and meta data.

To speed this process up and prevent it from getting held up when using multiple .seq files, it saves each file it reads into a temporary dataset to be saved locally. Once it has read in all these datasets, it merges them together, returns the data and meta, and deletes the temporary files. There may be a way to improve this process but doing it this way prevented the script from crashing when trying to read in 10 or more files at once.

## ACES Darks

This is pretty straightforward script that creates a dark file for a given exposure by created a median dark for a given exposure time and then subtract it from each data file.

NOTE: Dark data will need to be taken and saved with each dataset. If labeled correctly, the script will automatically read in the exposure time of the data and apply the correct dark file.

## ACES Log Data

This script reads in the log data from the files, merges it together, and returns a single log data frame object. It is currently set up to append UTC time (which isn't really used) and to delete extra entries in the log files that aren't currently used.

This requires using a more formally written (by an actual software person) set of scripts called aces-reader. The ACES_Log_Data.py script will call the function mimic_log_parser to do the actual reading of the files and return a data frame object.

## ACES Visualizer

Series of multiple scripts to do basic plotting tasks. Not essential to the pipeline, but nice to have for data visualization as many things are plotted repeatedly during testing.

## Data Notes

One of the main challenges of ACES data is the size of the files produced. Because it operates at such a high cadence, even a few minutes' worth of data can easily get up to hundreds of GBs. So testing should be done on smaller datasets and full processing of datasets may be rare. However, since ACES by design doesn't have many real data runs, this shouldn't be too much of a problem.

Data during testing was taken in two forms: white light and laser data. The processing for the different data types is slightly different for each mode. So code exists for both types, mostly related with how to determine the location of Zero Path Difference. There are ways in the code to switch back and forth between each version.

## Future Work

- Determine consistent processing steps such as the binning size and averaging techniques to apply to all processed data.
- Create a full processing flow such that data can be selected then automatically be processed through all required steps without the need for intervention.
- Rewrite the main file to allow it to be run straight through for processing and analyzing a dataset.
- Output finalized data as fits files with proper header. May be challenging due to file size and number of files.
- Create scripts for easily reading back in and analyzing processed ACES data at all levels (raw data files, processed data files, positions and intensities, spectra etc...)