gripspy Documentation

Release 0.1

Albert Y. Shih

May 01, 2017

Contents

CHAPTER 1

Introduction

This Python package is intended for analysis of data from the GRIPS long-duration flight of 2016 January 19–30. It can also be used to analyze the pre-flight calibration data that was already being recorded using flight-like packet formats. The code is still very much a work in progress, so not only are many elements still missing, even the parts that have been coded are likely to go through changes in their interfaces and organization.

CHAPTER 2

Table of contents

Installation

Prerequisites

This package depends on:

- Python 3.5 or 2.7 (untested on other versions)
- NumPy
- SciPy
- matplotlib
- Cython, and corresponding build environment for compiling C code
- scikit-image
- Astropy

However, rather than installing these packages individually, it is highly recommended that one instead use a scientific Python distribution (e.g., Anaconda), which will include useful packages such as Jupyter Notebook (formerly IPython Notebook).

The build environment for C code is a bit trickier to set up on Windows than on the other platforms:

- Python 3.5:
  - Install Microsoft Visual C++ Build Tools 2015 (check both Windows 8.1 SDK and Windows 10 SDK)
- Python 2.7:
  - Install Microsoft Visual C++ Compiler for Python 2.7
  - Patch `C:\yourpythoninstall\Lib\distutils\msvc9compiler.py` by adding the following highlighted lines at the top of the `find_vcvarsall()` function:

```
def find_vcvarsall(version):
    """Find the vcvarsall.bat file

    At first it tries to find the productdir of VS 2008 in the registry. If
    that fails it falls back to the VS90COMNTOOLS env var.
    """
    vsbase = VS_BASE % version
    vcpath = os.environ['ProgramFiles']
    vcpath = os.path.join(vcpath, 'Common Files', 'Microsoft',
        'Visual C++ for Python', '9.0', 'vcvarsall.bat')
    if os.path.isfile(vcpath): return vcpath
    vcpath = os.path.join(os.environ['LOCALAPPDATA'], 'Programs', 'Common',
↪'Microsoft', 'Visual C++ for Python', '9.0', 'vcvarsall.bat')
    if os.path.isfile(vcpath): return vcpath
    ...
```

– Create a file `distutils.cfg` in `C:\yourpythoninstall\Lib\distutils\` with the following:

```
[build]
compiler=msvc
```

You should now be able to build extensions. If these steps don't work for you, or you are using a different version of Python, this page or this other page may be helpful.

Installing `gripspy`

You can download `gripspy` either as a Git repository or just the code itself. Installing `gripspy` can be done through the basic ways (e.g., `python setup.py`), although I prefer using `pip` so that tracking installations is easier. I recommend installing `gripspy` in "editable" mode so that it is more convenient for active development, e.g.:

```
pip install -e .
```

Quick Start

Simply import `gripspy` to start.

```
import gripspy
```

The classes for handling science data are under `gripspy.science`. The classes for handling housekeeping data are under `gripspy.housekeeping`. Here are some examples of what has been implemented so far:

```
data = gripspy.science.GeData(detector_number, filename)
data = gripspy.science.BGOCounterData(filename)
data = gripspy.science.BGOEventData(filename)
data = gripspy.science.PYSequence(filelist)
data = gripspy.housekeeping.GPSData(filename)
```

Supported Packets

Only some of the types of GRIPS packets are currently supported.

Support?	SystemID	TmType	Description
	0x00	0x02	Flight software data rates
	0x00	0x03	Flight software telemetry settings
	0x00	0x0E	Flight software error
	0x00	0x0F	Flight software message
	0x01	0x02	Flight software watchdog
	0x02	0x02	Flight software watchpup
partial	0x03	0x02	To pointing control system computer (PCSC)
partial	0x03	0x03	From pointing control system computer (PCSC)
full	0x05	0x02	GPS
	0x07	0x02	Temperatures, through flight computer
	0x07	0x03	Thermostats
	0x0A	0x02	PDU voltages and currents
	0x0A	0x03	Cryostat temperatures
	0x0A	0x04	Power statuses
	0x0C	0x02	Cryocooler housekeeping
	0x0D	0x02	Not-dead-yet information
	0x0F	0x02	MPPT housekeeping
	0x10	0x1?	Ge quicklook spectrum and rates
	0x10	0x20	Ge quicklook hitmap strip
	0x40	0x02	Aspect housekeeping
	0x40	0x03	Aspect settings
	0x40	0x04	Temperatures, through aspect computer
	0x40	0x10	Aspect science
	0x4A	0x20	Aspect pitch-yaw image row
	0x4B	0x20	Aspect roll image row
partia	0x8?	0x02	Card cage (Ge) housekeeping
	0x8?	0x04	Card cage (Ge) ASIC registers, LV top
	0x8?	0x05	Card cage (Ge) ASIC registers, LV bottom
	0x8?	0x06	Card cage (Ge) ASIC registers, HV top
	0x8?	0x07	Card cage (Ge) ASIC registers, HV bottom
partial	0x8?	0x08	Card cage (Ge) counters
	0x8?	0x09	Card cage (Ge) temperatures
full	0x8?	0xF1	Card cage (Ge) event, raw format, LV side
full	0x8?	0xF2	Card cage (Ge) event, raw format, HV side
full	0x8?	0xF3	Card cage (Ge) event, normal format
	0x8?	0xFC	Card cage (Ge) channel enables
	0x9?	0x04	Card cage (Ge) ASIC register differences, LV top
	0x9?	0x05	Card cage (Ge) ASIC register differences, LV bottom
	0x9?	0x06	Card cage (Ge) ASIC register differences, HV top
	0x9?	0x07	Card cage (Ge) ASIC register differences, HV bottom
	0xB6	0x02	Shield (BGO) housekeeping
	0xB6	0x09	Shield (BGO) temperatures
	0xB6	0x13	Shield (BGO) configuration
full	0xB6	0x80	Shield (BGO) events, 8 bytes per event
full	0xB6	0x81	Shield (BGO) counter rates
full	0xB6	0x82	Shield (BGO) events, 5 bytes per event

Continued on next page

Table 2.1 – continued from previous page

Support?	SystemID	TmType	Description
	0xC0	0x02	SMASH housekeeping
	0xC0	0x03	SMASH log messages
	0xC0	0x10	SMASH science

Code Reference

Code reference for gripspy

Science

Ge

Module for analyzing data from the germanium detectors

class `gripspy.science.ge.`**GeData**(*detector*, *telemetry_file=None*, *save_file=None*, *max_triggers=None*, *reject_glitches=False*)

Class for analyzing event data from a germanium detector

Parameters

- **detector** (*int*) – The detector number (0 to 5, usually)
- **telemetry_file** (*str (or list)*) – The name of the telemetry file to analyze. If None is specified, a save file must be specified.
- **save_file** (*str (or list)*) – The name of a save file from a telemetry file that was previously parsed.
- **max_triggers** (*int*) – If not None, cull any events with more than this number of triggers on either detector side.
- **reject_glitches** (*bool*) – If True, cull any events that have any glitch bit set (even on non-triggered channels)

Notes

`event_time` is stored in 10-ns steps

a
Shorthand for the `adc` attribute

append(*other*, *defer_processing=False*)
Append the information in another GeData instance

c
Shorthand for the `cms` attribute

d
Shorthand for the `delta_time` attribute

e
Shorthand for the `event_time` attribute

g
Shorthand for the `glitch` attribute

hitmap
The hitmap of single-trigger events

lightcurve(*side*, *binning*, *time_step_in_seconds=1.0*, *energy_coeff=None*, *max_triggers=1*)
Return a detector-side-integrated lightcurve, excluding events that exceed a specified maximum number of triggers on that side. Note that `max_triggers == 1` is not quite the same as "single-trigger" events because no selection cut is imposed on the the opposite side.

Parameters

- **side** (*0 or 1*) – 0 for LV side, 1 for HV side
- **binning** (*array-like*) – The binning to use for the underlying data
- **time_step_in_seconds** (*float*) – The size of the time step in seconds
- **energy_coeff** (2x512 ndarray) – If not None, apply these linear coefficients (intercept, slope) to convert to energy in keV
- **max_triggers** (*int*) – Exclude events that have more than this number of triggers on the specified side

plot_conversion_vs_time(*asiccha*, *event_mask=None*, *time_binning=(250, 501, 1)*, *max_triggers=None*, ***hist2d_kwargs*)
Plot the conversions as a function of time since the preceding conversion. This type of plot is useful for looking for post-conversion baseline effects. This plot only makes sense if no events have been culled from the event list upon parsing (e.g., from having too many triggers or having glitches).

Binning of ADC values is currently automatic around the median of the values.

Parameters

- **asiccha** (*tuple or int*) – Either (ASIC#, channel#) if a tuple or ASIC# * 64 + channel# if an int
- **event_mask** (ndarray) – If not None, bool array where True values are which events to plot. Otherwise, the default is those events where `asiccha` did not trigger.
- **max_triggers** (*int*) – If not None, only show conversions that do not exceed this number of triggers on the same side as `asiccha`.
- **time_binning** (*tuple or array-like*) – The binning to use for the time since the preceding conversion in microseconds

plot_depth(*binning=array([-595, -585, -575, -565, -555, -545, -535, -525, -515, -505, -495, -485, -475, -465, -455, -445, -435, -425, -415, -405, -395, -385, -375, -365, -355, -345, -335, -325, -315, -305, -295, -285, -275, -265, -255, -245, -235, -225, -215, -205, -195, -185, -175, -165, -155, -145, -135, -125, -115, -105, -95, -85, -75, -65, -55, -45, -35, -25, -15, -5, 5, 15, 25, 35, 45, 55, 65, 75, 85, 95, 105, 115, 125, 135, 145, 155, 165, 175, 185, 195, 205, 215, 225, 235, 245, 255, 265, 275, 285, 295, 305, 315, 325, 335, 345, 355, 365, 375, 385, 395, 405, 415, 425, 435, 445, 455, 465, 475, 485, 495, 505, 515, 525, 535, 545, 555, 565, 575, 585, 595])*, ***hist_kwargs*)
Plot the depth-information plot, for only single-trigger events

Parameters **binning** (*array-like*) – The binning to use for the time-difference axis in nanoseconds

plot_depth_vs_energy(*side*, *depth_binning=array([-595, -585, -575, -565, -555, -545, -535, -525, -515, -505, -495, -485, -475, -465, -455, -445, -435, -425, -415, -405, -395, -385, -375, -365, -355, -345, -335, -325, -315, -305, -295, -285, -275, -265, -255, -245, -235, -225, -215, -205, -195, -185, -175, -165, -155, -145, -135, -125, -115, -105, -95, -85, -75, -65, -55, -45, -35, -25, -15, -5, 5, 15, 25, 35, 45, 55, 65, 75, 85, 95, 105, 115, 125, 135, 145, 155, 165, 175, 185, 195, 205, 215, 225, 235, 245, 255, 265, 275, 285, 295, 305, 315, 325, 335, 345, 355, 365, 375, 385, 395, 405, 415, 425, 435, 445, 455, 465, 475, 485, 495, 505, 515, 525, 535, 545, 555, 565, 575, 585, 595])*, *energy_binning=array([-128, -120, -112, -104, -96, -88, -80, -72, -64, -56, -48, -40, -32, -24, -16, -8, 0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, 128, 136, 144, 152, 160, 168, 176, 184, 192, 200, 208, 216, 224, 232, 240, 248, 256, 264, 272, 280, 288, 296, 304, 312, 320, 328, 336, 344, 352, 360, 368, 376, 384, 392, 400, 408, 416, 424, 432, 440, 448, 456, 464, 472, 480, 488, 496, 504, 512, 520, 528, 536, 544, 552, 560, 568, 576, 584, 592, 600, 608, 616, 624, 632, 640, 648, 656, 664, 672, 680, 688, 696, 704, 712, 720, 728, 736, 744, 752, 760, 768, 776, 784, 792, 800, 808, 816, 824, 832, 840, 848, 856, 864, 872, 880, 888, 896, 904, 912, 920, 928, 936, 944, 952, 960, 968, 976, 984, 992, 1000, 1008, 1016, 1024, 1032, 1040, 1048, 1056, 1064, 1072, 1080, 1088, 1096, 1104, 1112, 1120, 1128, 1136, 1144, 1152, 1160, 1168, 1176, 1184, 1192, 1200, 1208, 1216, 1224, 1232, 1240, 1248, 1256, 1264, 1272, 1280, 1288, 1296, 1304, 1312, 1320, 1328, 1336, 1344, 1352, 1360, 1368, 1376, 1384, 1392, 1400, 1408, 1416, 1424, 1432, 1440, 1448, 1456, 1464, 1472, 1480, 1488, 1496, 1504, 1512, 1520, 1528, 1536, 1544, 1552, 1560, 1568, 1576, 1584, 1592, 1600, 1608, 1616, 1624, 1632, 1640, 1648, 1656, 1664, 1672, 1680, 1688, 1696, 1704, 1712, 1720, 1728, 1736, 1744, 1752, 1760, 1768, 1776, 1784, 1792, 1800, 1808, 1816, 1824, 1832, 1840, 1848, 1856, 1864, 1872, 1880, 1888, 1896, 1904, 1912, 1920, 1928, 1936, 1944, 1952, 1960, 1968, 1976, 1984, 1992, 2000, 2008, 2016, 2024, 2032, 2040])*, *energy_coeff=None*, *min_lv_triggers=1*, *max_lv_triggers=1*, *min_hv_triggers=1*, *max_hv_triggers=1*, ***hist2d_kwargs*)

Plot the depth information versus 'energy', integrated over a detector side

Parameters

- **side** (*0 or 1*) – 0 for LV side, 1 for HV side
- **depth_binning** (*array-like*) – The binning to use for the time-difference axis in nanoseconds
- **energy_binning** (*array-like*) – The binning to use for the 'energy' axis in ADC bins or in keV
- **energy_coeff** (2x512 ndarray) – If not None, apply these linear coefficients (intercept, slope) to convert to energy in keV
- **min_lv_triggers** (*int*) – Do not include events with fewer than this number of triggers on the LV side
- **max_lv_triggers** (*int*) – Do not include events with more than this number of triggers on the LV side
- **min_hv_triggers** (*int*) – Do not include events with fewer than this number of triggers on the HV side
- **max_hv_triggers** (*int*) – Do not include events with more than this number of triggers on the HV side

plot_hitmap(***imshow_kwargs*)
Plot the hitmap of single-trigger events

plot_multiple_trigger_veto(*side*)
Plot the distribution of multiple-trigger events and veto information

Parameters **side** (*0 or 1*) – 0 for LV side, 1 for HV side

plot_spatial_spectrum(*side*, *binning=array([-128, -120, -112, -104, -96, -88, -80, -72, -64, -56, -48, -40, -32, -24, -16, -8, 0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, 128, 136, 144, 152, 160, 168, 176, 184, 192, 200, 208, 216, 224, 232, 240, 248, 256, 264, 272, 280, 288, 296, 304, 312, 320, 328, 336, 344, 352, 360, 368, 376, 384, 392, 400, 408, 416, 424, 432, 440, 448, 456, 464, 472, 480, 488, 496, 504, 512, 520, 528, 536, 544, 552, 560, 568, 576, 584, 592, 600, 608, 616, 624, 632, 640, 648, 656, 664, 672, 680, 688, 696, 704, 712, 720, 728, 736, 744, 752, 760, 768, 776, 784, 792, 800, 808, 816, 824, 832, 840, 848, 856, 864, 872, 880, 888, 896, 904, 912, 920, 928, 936, 944, 952, 960, 968, 976, 984, 992, 1000, 1008, 1016, 1024, 1032, 1040, 1048, 1056, 1064, 1072, 1080, 1088, 1096, 1104, 1112, 1120, 1128, 1136, 1144, 1152, 1160, 1168, 1176, 1184, 1192, 1200, 1208, 1216, 1224, 1232, 1240, 1248, 1256, 1264, 1272, 1280, 1288, 1296, 1304, 1312, 1320, 1328, 1336, 1344, 1352, 1360, 1368, 1376, 1384, 1392, 1400, 1408, 1416, 1424, 1432, 1440, 1448, 1456, 1464, 1472, 1480, 1488, 1496, 1504, 1512, 1520, 1528, 1536, 1544, 1552, 1560, 1568, 1576, 1584, 1592, 1600, 1608, 1616, 1624, 1632, 1640, 1648, 1656, 1664, 1672, 1680, 1688, 1696, 1704, 1712, 1720, 1728, 1736, 1744, 1752, 1760, 1768, 1776, 1784, 1792, 1800, 1808, 1816, 1824, 1832, 1840, 1848, 1856, 1864, 1872, 1880, 1888, 1896, 1904, 1912, 1920, 1928, 1936, 1944, 1952, 1960, 1968, 1976, 1984, 1992, 2000, 2008, 2016, 2024, 2032, 2040])*, *energy_coeff=None*, *use_strip_number=False*, ***hist2d_kwargs*)
Plot the subtracted single-trigger spectrum versus channel as a 2D image

Parameters

- **side** (*0 or 1*) – 0 for LV side, 1 for HV side
- **binning** (*array-like*) – The binning to use for the underlying data
- **energy_coeff** (2x512 ndarray) – If not None, apply these linear coefficients (intercept, slope) to convert to energy in keV
- **use_strip_number** (*bool*) – If True, use strip number as the horizontal axis instead of channel number

plot_spectrogram(*side*, *time_step_in_seconds=1.0*, *binning=array([-128, -120, -112, -104, -96, -88, -80, -72, -64, -56, -48, -40, -32, -24, -16, -8, 0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, 128, 136, 144, 152, 160, 168, 176, 184, 192, 200, 208, 216, 224, 232, 240, 248, 256, 264, 272, 280, 288, 296, 304, 312, 320, 328, 336, 344, 352, 360, 368, 376, 384, 392, 400, 408, 416, 424, 432, 440, 448, 456, 464, 472, 480, 488, 496, 504, 512, 520, 528, 536, 544, 552, 560, 568, 576, 584, 592, 600, 608, 616, 624, 632, 640, 648, 656, 664, 672, 680, 688, 696, 704, 712, 720, 728, 736, 744, 752, 760, 768, 776, 784, 792, 800, 808, 816, 824, 832, 840, 848, 856, 864, 872, 880, 888, 896, 904, 912, 920, 928, 936, 944, 952, 960, 968, 976, 984, 992, 1000, 1008, 1016, 1024, 1032, 1040, 1048, 1056, 1064, 1072, 1080, 1088, 1096, 1104, 1112, 1120, 1128, 1136, 1144, 1152, 1160, 1168, 1176, 1184, 1192, 1200, 1208, 1216, 1224, 1232, 1240, 1248, 1256, 1264, 1272, 1280, 1288, 1296, 1304, 1312, 1320, 1328, 1336, 1344, 1352, 1360, 1368, 1376, 1384, 1392, 1400, 1408, 1416, 1424, 1432, 1440, 1448, 1456, 1464, 1472, 1480, 1488, 1496, 1504, 1512, 1520, 1528, 1536, 1544, 1552, 1560, 1568, 1576, 1584, 1592, 1600, 1608, 1616, 1624, 1632, 1640, 1648, 1656, 1664, 1672, 1680, 1688, 1696, 1704, 1712, 1720, 1728, 1736, 1744, 1752, 1760, 1768, 1776, 1784, 1792, 1800, 1808, 1816, 1824, 1832, 1840, 1848, 1856, 1864, 1872, 1880, 1888, 1896, 1904, 1912, 1920, 1928, 1936, 1944, 1952, 1960, 1968, 1976, 1984, 1992, 2000, 2008, 2016, 2024, 2032, 2040])*, *energy_coeff=None*, *max_triggers=1*, ***hist2d_kwargs*)

Plot a detector-side-integrated spectrogram, excluding events that exceed a specified maximum number of triggers on that side. Note that `max_triggers == 1` is not quite the same as "single-trigger" events because no selection cut is imposed on the the opposite side.

Parameters

- **side** (*0 or 1*) – 0 for LV side, 1 for HV side
- **time_step_in_seconds** (*float*) – The size of the time step in seconds
- **binning** (*array-like*) – The binning to use for the underlying data
- **energy_coeff** (2x512 ndarray) – If not None, apply these linear coefficients (intercept, slope) to convert to energy in keV
- **max_triggers** (*int*) – Exclude events that have more than this number of triggers on the specified side

plot_spectrum(*asiccha*, *binning=array([0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, 128, 136, 144, 152, 160, 168, 176, 184, 192, 200, 208, 216, 224, 232, 240, 248, 256, 264, 272, 280, 288, 296, 304, 312, 320, 328, 336, 344, 352, 360, 368, 376, 384, 392, 400, 408, 416, 424, 432, 440, 448, 456, 464, 472, 480, 488, 496, 504, 512, 520, 528, 536, 544, 552, 560, 568, 576, 584, 592, 600, 608, 616, 624, 632, 640, 648, 656, 664, 672, 680, 688, 696, 704, 712, 720, 728, 736, 744, 752, 760, 768, 776, 784, 792, 800, 808, 816, 824, 832, 840, 848, 856, 864, 872, 880, 888, 896, 904, 912, 920, 928, 936, 944, 952, 960, 968, 976, 984, 992, 1000, 1008, 1016, 1024, 1032, 1040, 1048, 1056, 1064, 1072, 1080, 1088, 1096, 1104, 1112, 1120, 1128, 1136, 1144, 1152, 1160, 1168, 1176, 1184, 1192, 1200, 1208, 1216, 1224, 1232, 1240, 1248, 1256, 1264, 1272, 1280, 1288, 1296, 1304, 1312, 1320, 1328, 1336, 1344, 1352, 1360, 1368, 1376, 1384, 1392, 1400, 1408, 1416, 1424, 1432, 1440, 1448, 1456, 1464, 1472, 1480, 1488, 1496, 1504, 1512, 1520, 1528, 1536, 1544, 1552, 1560, 1568, 1576, 1584, 1592, 1600, 1608, 1616, 1624, 1632, 1640, 1648, 1656, 1664, 1672, 1680, 1688, 1696, 1704, 1712, 1720, 1728, 1736, 1744, 1752, 1760, 1768, 1776, 1784, 1792, 1800, 1808, 1816, 1824, 1832, 1840, 1848, 1856, 1864, 1872, 1880, 1888, 1896, 1904, 1912, 1920, 1928, 1936, 1944, 1952, 1960, 1968, 1976, 1984, 1992, 2000, 2008, 2016, 2024, 2032, 2040, 2048, 2056, 2064, 2072, 2080, 2088, 2096, 2104, 2112, 2120, 2128, 2136, 2144, 2152, 2160, 2168, 2176, 2184, 2192, 2200, 2208, 2216, 2224, 2232, 2240, 2248, 2256, 2264, 2272, 2280, 2288, 2296, 2304, 2312, 2320, 2328, 2336, 2344, 2352, 2360, 2368, 2376, 2384, 2392, 2400, 2408, 2416, 2424, 2432, 2440, 2448, 2456, 2464, 2472, 2480, 2488, 2496, 2504, 2512, 2520, 2528, 2536, 2544, 2552, 2560, 2568, 2576, 2584, 2592, 2600, 2608, 2616, 2624, 2632, 2640, 2648, 2656, 2664, 2672, 2680, 2688, 2696, 2704, 2712, 2720, 2728, 2736, 2744, 2752, 2760, 2768, 2776, 2784, 2792, 2800, 2808, 2816, 2824, 2832, 2840, 2848, 2856, 2864, 2872, 2880, 2888, 2896, 2904, 2912, 2920, 2928, 2936, 2944, 2952, 2960, 2968, 2976, 2984, 2992, 3000, 3008, 3016, 3024, 3032, 3040, 3048, 3056, 3064, 3072, 3080, 3088, 3096, 3104, 3112, 3120, 3128, 3136, 3144, 3152, 3160, 3168, 3176, 3184, 3192, 3200, 3208, 3216, 3224, 3232, 3240, 3248, 3256, 3264, 3272, 3280, 3288, 3296, 3304, 3312, 3320, 3328, 3336, 3344, 3352, 3360, 3368, 3376, 3384, 3392, 3400, 3408, 3416, 3424, 3432, 3440, 3448, 3456, 3464, 3472, 3480, 3488, 3496, 3504, 3512, 3520, 3528, 3536, 3544, 3552, 3560, 3568, 3576])*)

Plot the raw spectrum for a specified channel

Parameters

- **asiccha** (*tuple or int*) – Either (ASIC#, channel#) if a tuple or ASIC# * 64 + channel# if an int
- **binning** (*array-like*) – The binning to use for the underlying data

plot_subtracted_spectrum(*asiccha*, *binning=array([-128, -120, -112, -104, -96, -88, -80, -72, -64, -56, -48, -40, -32, -24, -16, -8, 0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, 128, 136, 144, 152, 160, 168, 176, 184, 192, 200, 208, 216, 224, 232, 240, 248, 256, 264, 272, 280, 288, 296, 304, 312, 320, 328, 336, 344, 352, 360, 368, 376, 384, 392, 400, 408, 416, 424, 432, 440, 448, 456, 464, 472, 480, 488, 496, 504, 512, 520, 528, 536, 544, 552, 560, 568, 576, 584, 592, 600, 608, 616, 624, 632, 640, 648, 656, 664, 672, 680, 688, 696, 704, 712, 720, 728, 736, 744, 752, 760, 768, 776, 784, 792, 800, 808, 816, 824, 832, 840, 848, 856, 864, 872, 880, 888, 896, 904, 912, 920, 928, 936, 944, 952, 960, 968, 976, 984, 992, 1000, 1008, 1016, 1024, 1032, 1040, 1048, 1056, 1064, 1072, 1080, 1088, 1096, 1104, 1112, 1120, 1128, 1136, 1144, 1152, 1160, 1168, 1176, 1184, 1192, 1200, 1208, 1216, 1224, 1232, 1240, 1248, 1256, 1264, 1272, 1280, 1288, 1296, 1304, 1312, 1320, 1328, 1336, 1344, 1352, 1360, 1368, 1376, 1384, 1392, 1400, 1408, 1416, 1424, 1432, 1440, 1448, 1456, 1464, 1472, 1480, 1488, 1496, 1504, 1512, 1520, 1528, 1536, 1544, 1552, 1560, 1568, 1576, 1584, 1592, 1600, 1608, 1616, 1624, 1632, 1640, 1648, 1656, 1664, 1672, 1680, 1688, 1696, 1704, 1712, 1720, 1728, 1736, 1744, 1752, 1760, 1768, 1776, 1784, 1792, 1800, 1808, 1816, 1824, 1832, 1840, 1848, 1856, 1864, 1872, 1880, 1888, 1896, 1904, 1912, 1920, 1928, 1936, 1944, 1952, 1960, 1968, 1976, 1984, 1992, 2000, 2008, 2016, 2024, 2032, 2040]*)

Plot the common-mode-subtracted spectrum for a specified channel

Parameters

- **asiccha** (*tuple or int*) – Either (ASIC#, channel#) if a tuple or ASIC# * 64 + channel# if an int
- **binning** (*array-like*) – The binning to use for the underlying data

s

Shorthand for the `single_triggers` attribute

s_hv

Shorthand for the `single_triggers_lv` attribute

s_lv

Shorthand for the `single_triggers_lv` attribute

save(*save_file=None*, *use_current_directory=False*)

Save the parsed data for future reloading. The data is stored in gzip-compressed binary pickle format.

Parameters

- **save_file** (*str*) – The name of the save file to create. If none is provided, the default is the name of the telemetry file with the extension ".ge?.pgz" appended if a single telemetry file is the source.
- **use_current_directory** (*bool*) – If True, remove any directory specification from `save_file`

Notes

Save files may not be compatible between Python 2 and 3

t

Shorthand for the `trigger` attribute

v
Shorthand for the `veto` attribute

vh
Shorthand for the HV guard-ring vetoes

vl
Shorthand for the LV guard-ring vetoes

vm
Shorthand for the multiple-trigger vetoes

vs
Shorthand for the shield vetoes

BGO

Module for analyzing data from the BGO shields

class `gripspy.science.bgo.`**BGOEventData**(*telemetry_file=None*, *save_file=None*)
Class for analyzing event data from the BGO shields

Parameters

- **telemetry_file** (*str (or list)*) – The name of the telemetry file to analyze. If None is specified, a save file must be specified.
- **save_file** (*str (or list)*) – The name of a save file from a telemetry file that was previously parsed.

Notes

`event_time` is stored in 10-ns steps

append(*other*)
Append the information in another BGOEventData instance

c
Shorthand for the `channel` attribute

e
Shorthand for the `event_time` attribute

l
Shorthand for the `level` attribute

l0
Indices for the events of crossing threshold level 0

l1
Indices for the events of crossing threshold level 1

l2
Indices for the events of crossing threshold level 2

l3
Indices for the events of crossing threshold level 3

save(*save_file=None*, *use_current_directory=False*)
Save the parsed data for future reloading. The data is stored in gzip-compressed binary pickle format.

Parameters

- **save_file** (*str*) – The name of the save file to create. If none is provided, the default is the name of the telemetry file with the extension ".bgoe.pgz" appended if a single telemetry file is the source.
- **use_current_directory** (*bool*) – If True, remove any directory specification from save_file

Notes

Save files may not be compatible between Python 2 and 3

class gripspy.science.bgo.**BGOCounterData**(*telemetry_file=None*, *save_file=None*)
Class for analyzing event data from the BGO shields

Parameters

- **telemetry_file** (*str (or list)*) – The name of the telemetry file to analyze. If None is specified, a save file must be specified.
- **save_file** (*str (or list)*) – The name of a save file from a telemetry file that was previously parsed.

append(*other*)
Append the information in another BGOCounterData instance

c
Shorthand for the channel_count attribute

l
Shorthand for the channel_livetime attribute

save(*save_file=None*, *use_current_directory=False*)
Save the parsed data for future reloading. The data is stored in gzip-compressed binary pickle format.

Parameters

- **save_file** (*str*) – The name of the save file to create. If none is provided, the default is the name of the telemetry file with the extension ".bgoc.pgz" appended if a single telemetry file is the source.
- **use_current_directory** (*bool*) – If True, remove any directory specification from save_file

Notes

Save files may not be compatible between Python 2 and 3

t
Shorthand for the counter_time attribute

tl
Shorthand for the total_livetime attribute

v
Shorthand for the veto_count attribute

Aspect

Module for processing aspect data

class gripspy.science.aspect.**PYFrame**(*filename*, *uid=158434*)
Class for a pitch-yaw image

Parameters

- **filename** (*str*) – The name of the FITS file of the camera frame
- **uid** (*int*) – Defaults to the UID of the pitch-yaw camera, but can be set to a different camera's UID or to None

plot_image(***imshow_kwargs*)
Plots the pitch-yaw image, including Sun/fiducial detections.

class gripspy.science.aspect.**PYSequence**(*list_of_files*)
Class for a sequence of pitch-yaw images

Parameters **list_of_files** (*list of str*) – List of FITS files with pitch-yaw images

dataframe
Obtain a pandas DataFrame

plot_centers(***dataframe_plot_kwargs*)
Plot the center of the brightest Sun across the entire image sequence

class gripspy.science.aspect.**RFrame**(*filename*, *uid=142974*)
Class for a roll image

Parameters

- **filename** (*str*) – The name of the FITS file of roll image
- **uid** (*int*) – Defaults to the UID of the roll camera, but can be set to a different camera's UID or to None

static atmosphere_asym(*x*, *midpoint*, *scale*, *amp1*, *amp2*, *dc*)
Exponential atmospheric model with asymmetry in normalization

static atmosphere_sym(*x*, *midpoint*, *scale*, *amp*, *dc*)
Exponential atmospheric model with symmetry (i.e., hyperbolic cosine)

plot_image(***imshow_kwargs*)
Plots the roll image, including annotations.

class gripspy.science.aspect.**RSequence**(*list_of_files*)
Class for a sequence of roll images

Parameters **list_of_files** (*list of str*) – List of FITS files with roll images

dataframe
Obtain a pandas DataFrame

plot_midpoint(***dataframe_plot_kwargs*)
Plot the midpoints

Housekeeping

Card cages

Module for analyzing card-cage information. The information is currently split between CardCageControl and CardCageInfo because of the two types of packets. This awkward implementation is likely to change in the future.

class gripspy.housekeeping.cc.**CardCageControl**(*telemetry_file=None*, *save_file=None*)
Class for analyzing card-cage information from the housekeeping packet

Parameters

- **telemetry_file** (*str (or list)*) – The name of the telemetry file to analyze. If None is specified, a save file must be specified.
- **save_file** (*str*) – The name of a save file from a telemetry file that was previously parsed.

Notes

This implementation extracts only a subset of the information, and the API is subject to change.

append(*other*)
Append the information in another CardCageControl instance

attributes_all = ['systime', 'count_watchpup_reset', 'daq_running', 'last_cmdtype', 'mode_abort_ramp', 'mode_co

attributes_convert = ['period_counters', 'period_housekeeping', 'time_since_last_reset']

attributes_simple = ['systime', 'count_watchpup_reset', 'daq_running', 'last_cmdtype', 'mode_abort_ramp', 'mod

save(*save_file=None*, *use_current_directory=False*)
Save the parsed data for future reloading. The data is stored in gzip-compressed binary pickle format.

Parameters

- **save_file** (*str*) – The name of the save file to create. If none is provided, the default is the name of the telemetry file with the extension ".cccontrol.pgz" appended if a single telemetry file is the source.
- **use_current_directory** (*bool*) – If True, remove any directory specification from save_file

Notes

Save files may not be compatible between Python 2 and 3

class gripspy.housekeeping.cc.**CardCageInfo**(*telemetry_file=None*, *save_file=None*)
Class for analyzing card-cage information from the counters packet

Parameters

- **telemetry_file** (*str (or list)*) – The name of the telemetry file to analyze. If None is specified, a save file must be specified.
- **save_file** (*str*) – The name of a save file from a telemetry file that was previously parsed.

Notes

This implementation extracts all of the information, but the API is subject to change.

append(*other*)
Append the information in another CardCageInfo instance

attributes_all = ['systime', 'elapsed_time', 'busy_time', 'busy_count', 'veto_mult_trig_hard', 'veto_mult_trig_soft',

attributes_simple = ['systime', 'elapsed_time', 'busy_time', 'busy_count', 'veto_mult_trig_hard', 'veto_mult_trig_s

save(*save_file=None*, *use_current_directory=False*)
Save the parsed data for future reloading. The data is stored in gzip-compressed binary pickle format.

Parameters

- **save_file** (*str*) – The name of the save file to create. If none is provided, the default is the name of the telemetry file with the extension ".ccinfo.pgz" appended if a single telemetry file is the source.
- **use_current_directory** (*bool*) – If True, remove any directory specification from `save_file`

Notes

Save files may not be compatible between Python 2 and 3

GPS

Module for analyzing GPS and pressure data from the SIP

class `gripspy.housekeeping.gps.`**GPSData**(*telemetry_file=None*, *save_file=None*)
Class for analyzing GPS and pressure data from the SIP

Parameters

- **telemetry_file** (*str*) – The name of the telemetry file to analyze. If None is specified, a save file must be specified.
- **save_file** (*str*) – The name of a save file from a telemetry file that was previously parsed.

Notes

Trusts the "user" GPS information Averages the SIP1 and SIP2 pressure information

save(*save_file=None*)
Save the parsed data for future reloading. The data is stored in gzip-compressed binary pickle format.

Parameters **save_file** (*str*) – The name of the save file to create. If none is provided, the default is the name of the telemetry file with the extension ".gps.pgz" appended.

Notes

Save files may not be compatible between Python 2 and 3

Pointing

Module for analyzing pointing data

class gripspy.housekeeping.pointing.**PointingData**(*telemetry_file=None*, *save_file=None*)
Class for analyzing pointing data

Parameters

- **telemetry_file** (*str*) – The name of the telemetry file to analyze. If None is specified, a save file must be specified.
- **save_file** (*str*) – The name of a save file from a telemetry file that was previously parsed.

Notes

This implementation is still incomplete!

save(*save_file=None*)
Save the parsed data for future reloading. The data is stored in gzip-compressed binary pickle format.

Parameters **save_file** (*str*) – The name of the save file to create. If none is provided, the default is the name of the telemetry file with the extension ".pointing.pgz" appended.

Notes

Save files may not be compatible between Python 2 and 3

Telemetry

Subpackage for parsing telemetry files

gripspy.telemetry.**packet_generator**(*f*, *verbose=False*, *raise_exceptions=False*)
Generator that yields valid packets from a telemetry-file object. Packets that have invalid checksums are skipped past.

Parameters

- **f** (*file object*) – e.g., an already open file object
- **verbose** (*boolean*) – If True, output the percentage of the file as a progress indicator
- **raise_exceptions** (*boolean*) – If True, raise exceptions (e.g., if an invalid checksum is encountered)

gripspy.telemetry.**parser_generator**(*f*, *filter_systemid=None*, *filter_tmtype=None*, *verbose=False*)
Generator that yields parsed packet contents from a telemetry-file object. Packets that have invalid checksums are skipped past.

Parameters

- **f** (*file object*) – e.g., an already open file object
- **filter_systemid** (*int*) – If specified, only yield packets that have a matching SystemId
- **filter_tmtype** (*int*) – If specified, only yield packets that have a match TmType

- **verbose** (*boolean*) – If True, output the percentage of the file as a progress indicator

gripspy.telemetry.**inspect**(*filename*)
Report top-level information about the contents of one or more telemetry files

Parameters **filename** (*str or list of str*) – One or more telemetry files

gripspy.telemetry.**parser**(*packet*, *filter_systemid=None*, *filter_tmtype=None*)
Parse a telemetry packet for its contents

Parameters

- **packet** (*bytearray-like*) – A full telemetry packet including the 16-byte header
- **filter_systemid** (*int*) – If specified, only parse the packet if the SystemID matches
- **filter_tmtype** (*int*) – If specified, only parse the packet if the TmType matches

Returns **out** – The contents of the telemetry packet

Return type dict

gripspy.telemetry.**print_parsers**()
Print the SystemID and TmType pairs that have a parsing function defined. The parsing function may not be fully implemented.

Utilities

gripspy.util.checksum

Adapted from the C implmentation written by Lammert Bies

gripspy.util.checksum.**crc16**(*__Pyx_memviewslice data*) → unsigned short
Compute the CRC16 checksum of a bytearray or equivalent. Because of Cython limitations, the input must be writeable (e.g., not a string)

gripspy.util.coincidence

gripspy.util.coincidence.**find_nearest_in**
For two sorted arrays a and b, returns a list of the indices of the nearest element in b for each element in a. The two arrays must be of the same numeric type.

gripspy.util.time

Module for time conversion

gripspy.util.time.**oeb2utc**(*systime_array*, *seconds_of_gps_delay=6*)
Converts 6-byte system time (AKA gondola time) to UTC during the flight. Note that the detectors report a higher-precision "event" time that must first be divided by 10 to convert to system time.

Parameters

- **systime_array** (ndarray) – Array of system times (normally int64)
- **seconds_of_gps_delay** (*int*) – Lag in seconds between GPS measurement and GPS recording (default: 6 seconds)

Returns **utc_array** – Array of UTC times datetime64

Return type ndarray

Notes

This conversion function works as intended for only system times during the flight. The flight computer was restarted on 2016 Jan 25, which leads to a discontinuity in system times, and this discontinuity is taken into account by the function.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

s

t

u

Index

O

P

R

S

T

V