pylleo Documentation

Release 0.1.1

Ryan J. Dillon

Dec 13, 2019

Contents:

CHAPTER 1

Guide

1.1 Installation

pylleo is written in *Python 3.5* and does not currently support prior versions. It is available from the PyPi repository and can be installed using *pip*:

```
pip3 install pylleo
```

It is preferable to use a Python virtual environment, particularly to avoid any problems if you are have multiple Python versions installed.

```
cd <project path>
virtualenv --python=python3 venv
source venv/bin/activate
pip install pylleo
```

If you have installed *pylleo* using a virtual environment, be sure to activate that environment before running the *pylleo-cal* script described in the *Calibration* documentation.

1.2 Loading data

The data must first be downloaded from the datalogger using the Little Leonardo software. *pylleo* uses the filename for loading the data, so care should be taken to name the files correctly (shown below). While *pylleo* will automatically try to identiy for the timestamp format used, it is recommended that you follow the *ISO 8601* date format without underscores, i.e. *YYYYMMDD*.

```
<date>_<tag model>_<tag serial>_<animal_name>_<modification>_suffix.TXT
```

Below is an example of how the contents of a Little Leonardo data directory should look:

```
./20160418_W190PD3GT_34840_Skinny_2Neutral
├── 20160418_W190PD3GT_34840_Skinny_2Neutral-Acceleration-X.TXT
├── 20160418_W190PD3GT_34840_Skinny_2Neutral-Acceleration-Y.TXT
├── 20160418_W190PD3GT_34840_Skinny_2Neutral-Acceleration-Z.TXT
├── 20160418_W190PD3GT_34840_Skinny_2Neutral-Depth.TXT
├── 20160418_W190PD3GT_34840_Skinny_2Neutral-Propeller.TXT
└── 20160418_W190PD3GT_34840_Skinny_2Neutral-Temperature.TXT
```

The code can then be loaded to a pandas dataframe, by first creating a meta-data dictionary (saved as a YAML format file to the data directory), and then loading the data using the created meta-data.

```
import pylleo

path_dir = './'
meta = pylleo.lleoio.read_meta(path_dir, 'W190PD3GT', 34840)
data = pylleo.lleoio.read_data(meta, path_dir)
```

1.3 Calibration

The acclerometer and propeller data must be calibrated before being used for analysis. The sections below provide information on how to apply these calibrations. For instructions on how to a calibration file (i.e. *cal.yml*) or the propeller calibration *.csv* file, please see the *Calibration* documentation.

1.3.1 Calibrating accelerometer data

The calibration file *cal.yml* created during the calibration process is first loaded, and then the coefficents for the fit of the calibration data for each axis is applied to that axis data in the loaded dataframe.

```
from pylleo import lleocal

# Load calibrate data
cal_dict = yamlord.read_yaml('cal.yml')

# Apply calibration to accelerometer axes and
# save as new columns to the dataframe
data = lleocal.calibrate_acc(data, cal_dict, col_name)
```

1.3.2 Calibrating propeller data

```
cal_fname = './speed_calibrations.csv'

# Calibrate propeller measurements to speed m s^-2
data = calibrate_propeller(data_df, cal_fname)
```

1.4 Interpolation of sensor data

The data of sensors that sample at a lower frequency than another sensor (e.g. the accelerometer) can be interpolated using the *pandas.DataFrame* class method interpolate as shown below.

```
data.interpolate('linear', inplace=True)
```

CHAPTER 2

Calibration

Note: Currently only tag model W190PD3GT is covered, but other tags will be added as opportunity permits.

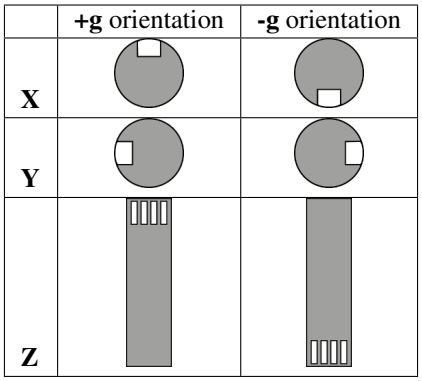
2.1 Accelerometer

Note: The calibration procedure described below needs review (particularly the orientation of the sensor for the associated gravitational forces). This will be updated to be a thorough explanation in subsequent releases of the documentation.

The data provided by the Little Leonardo dataloggers are presented in a raw format which need to be adjusted to units of gravity (**g**). Within a period of approximately one month adjacent to collection of data, a calibration file should be created using the process described below.

2.1.1 Collecting calibration data

First configure and activate the datalogger for recording. For a period of approximately 10 seconds orient the tag in each of the following orientations, one axis at a time. Longer durations make visually identifying these periods in the data easier.



2.1.2 Running the calibration software

Calibration is performed on accelerometer sensor data that does not have accompanying magnetometer or gyroscope data by performing by making a linear fit between a collection of points occurring at **-g** and **+g** orientations of an axis. This fit can then be applied to the original accelerometer count data to transform the data into units of **g**.

Launching the application

An application for simplifying the calibration process (made with the bokeh visualization library) has been included with *pylleo* as an executable script, which launches a bokeh application in your web-browser.

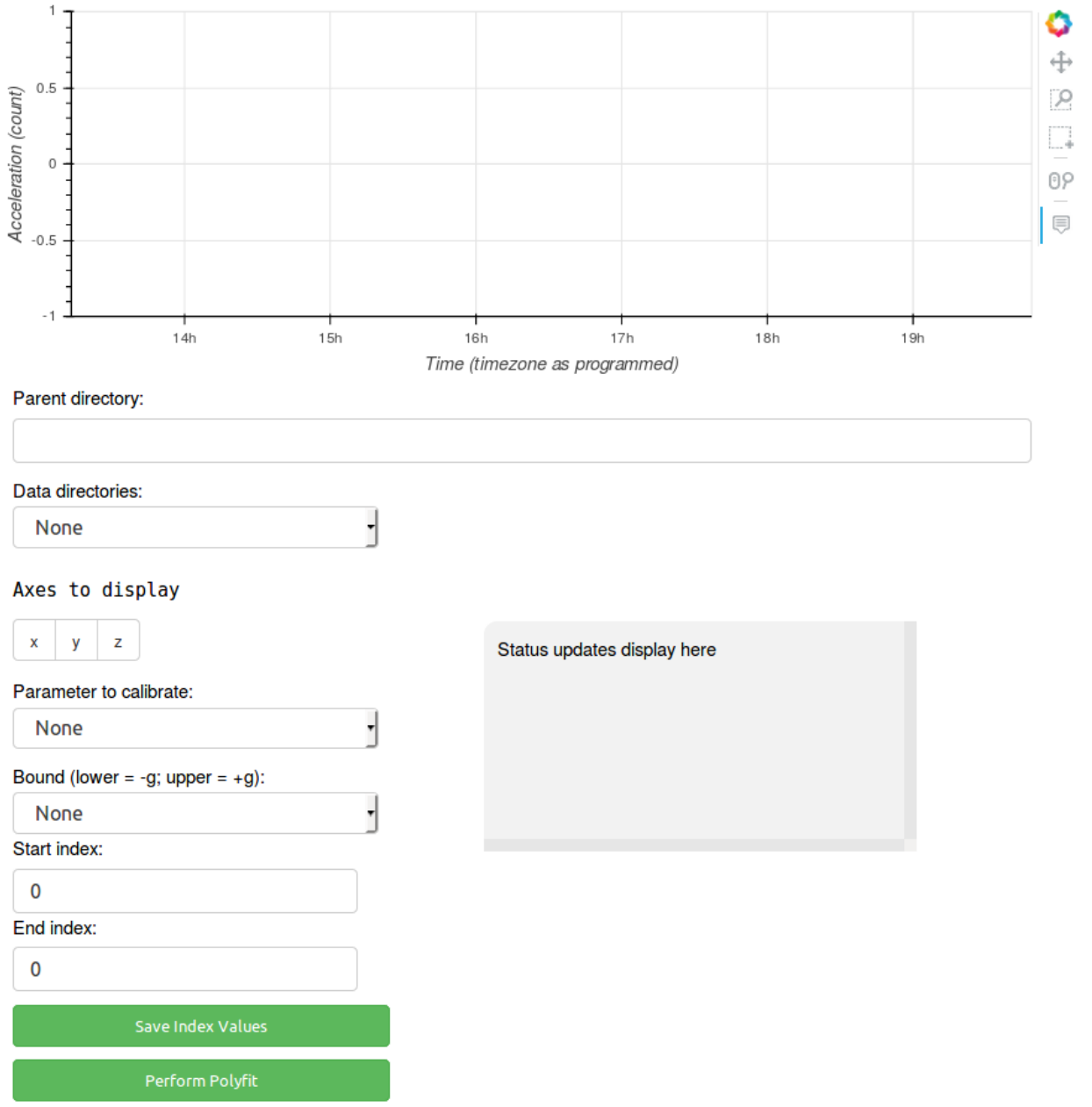
The script is automatically installed with *pylleo*, just run it from the command-line with an option for specifying how it opens in your browser:

```
Usage: pylleo-cal [OPTIONS]

  Calibrate accelerometer data

Options:
  --new TEXT  Method to open application in browser. "tab" opens the
              application in a new browser, and "window" opens it in a new
              browser window.
  --help      Show this message and exit.
```

The following page should appear in your browser, and the application will shut itself down when you close this page:



By zooming into segments of data when the datalogger was at one of the two orientations described above, a selection tool can be used to select those data points to be used for calibration. The start and stop index positions for each of these segments are saved to a file in the data directory *cal.yml*, and once all indices have been saved the fit coefficients can be calculated and saved to the same file. These coefficients can later be used for applying the fit to the data points using the routine *lleocal.calibrate_accelerometer()*.

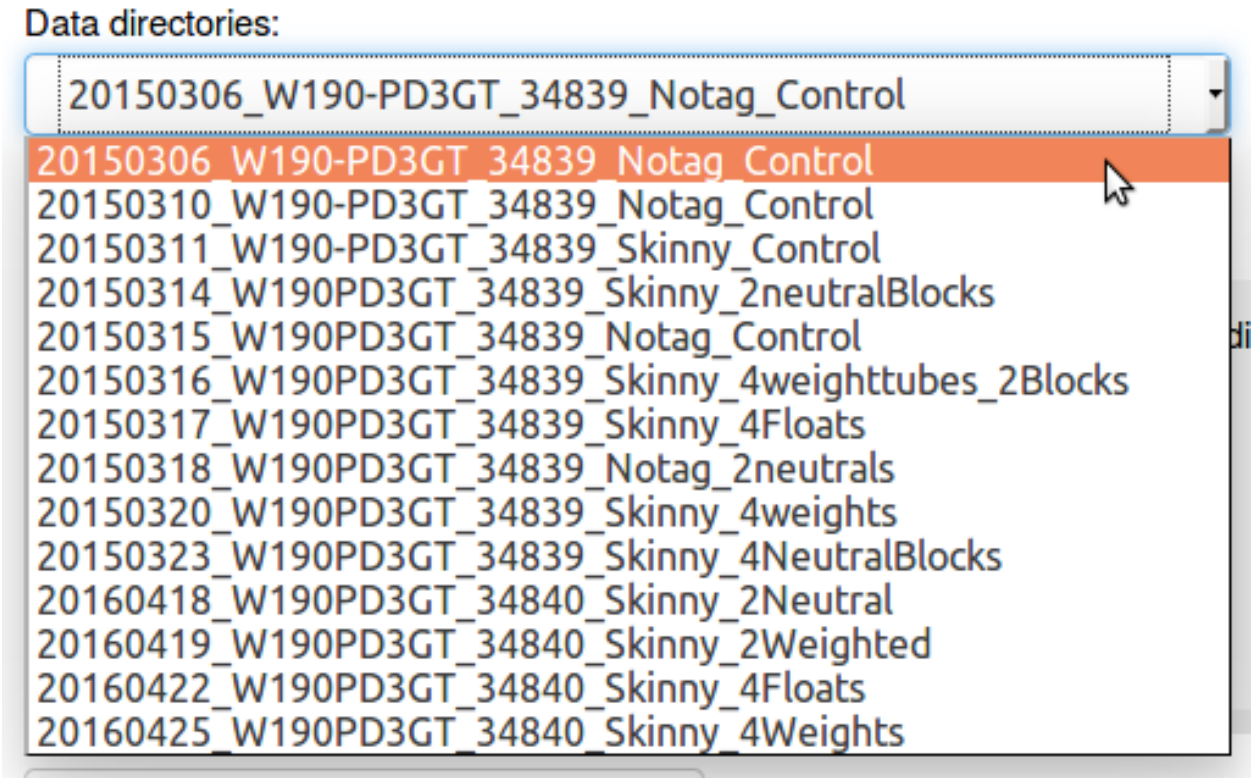
The tools for zooming and selecting the data are in the top right hand corner of the page. A summary table of the tools used in the app (shown below) have been taken from the *Bokeh* documentation for plot tools.

Icon	Bokeh documentation description
	The pan tool allows the user to pan the plot by left dragging the mouse or dragging a finger across the plot region.
	The box zoom tool allows the user to define a rectangular region to zoom the plot bounds too, by left-dragging a mouse, or dragging a finger across the plot area.
	The box selection tool allows the user to define a rectangular selection region by left-dragging a mouse, or dragging a finger across the plot area.
	The wheel zoom tool will zoom the plot in and out, centered on the current mouse location. It will respect any min and max value ranges preventing zooming in and out beyond these.
	The hover tool is a passive inspector tool. It is generally on at all times, but can be configured in the inspectors menu associated with the toolbar.

Loading data

All of your data should be organized in their own directories within one "parent" directory. Copy-paste the *full path* to this parent directory to the text input field labeled "Parent directory".

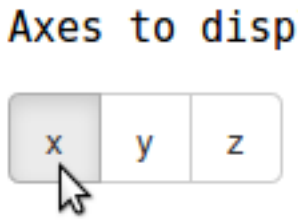
The drop-down list labeled "Data directories" will then propagate with a list of the directories in your parent directory. The data from the first directory in the list will be loaded into the plot. To select a different directory, select it from the list and its data will be loaded.



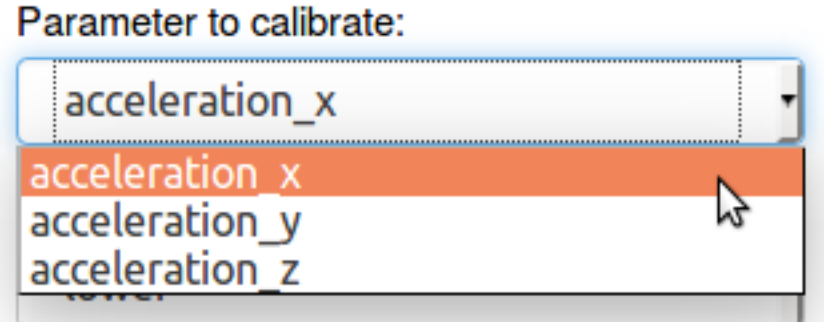
Selecting data

The app first loads all three axes of acceleration data, which from time to time may be helpful to view at the same time for comparing differences between the axes in +g/-g orientations. When selecting the index positions for the start and end of the calibration orientation regions, it is usually easiest to have only one axis displayed at a time.

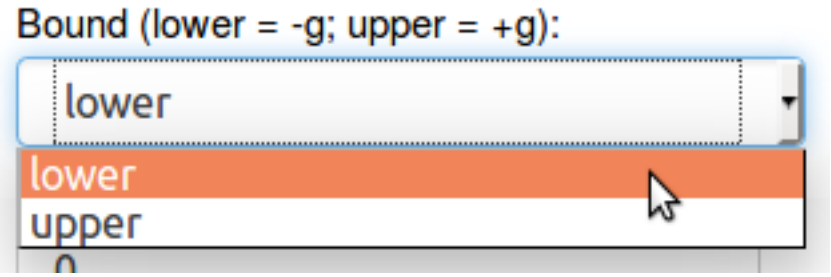
To start with the x-axis, de-select the y and z axes under the text "Axes to Display" by clicking on the buttons with their respective labels:



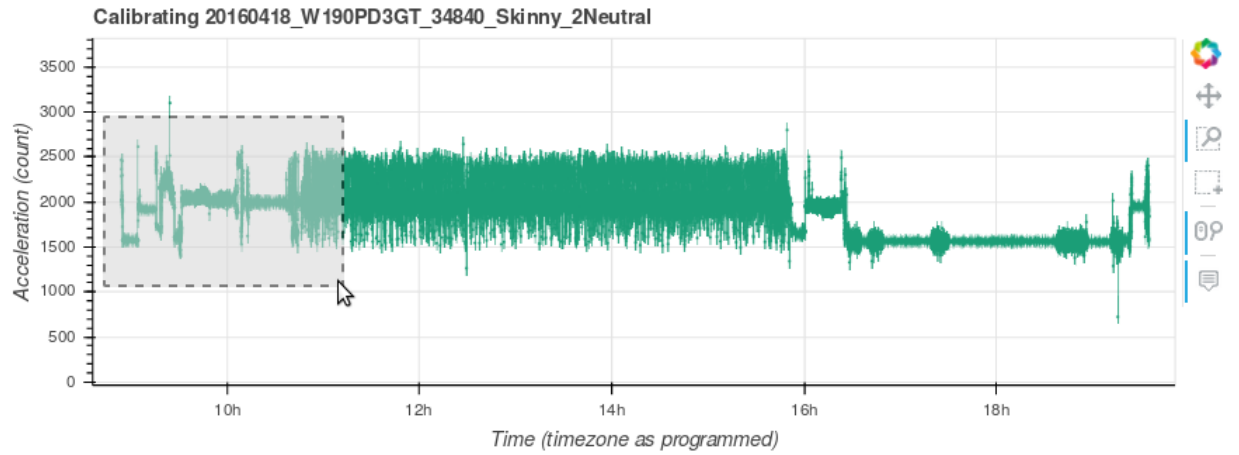
Make sure the data parameter (i.e. accelerometer axis) you wish to select calibration indices for is selected:



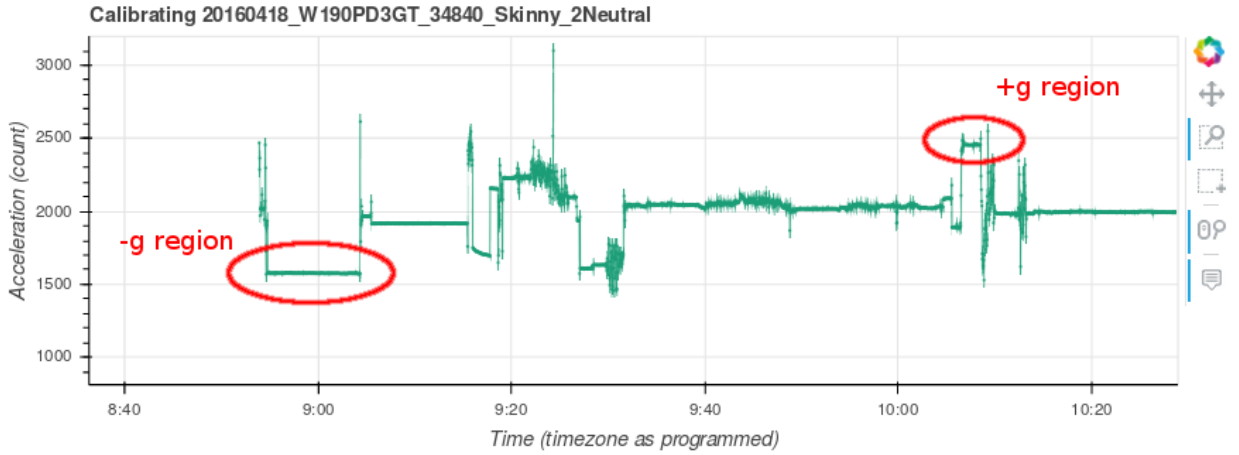
And the "bound" (i.e. the +g or -g position for that axis):



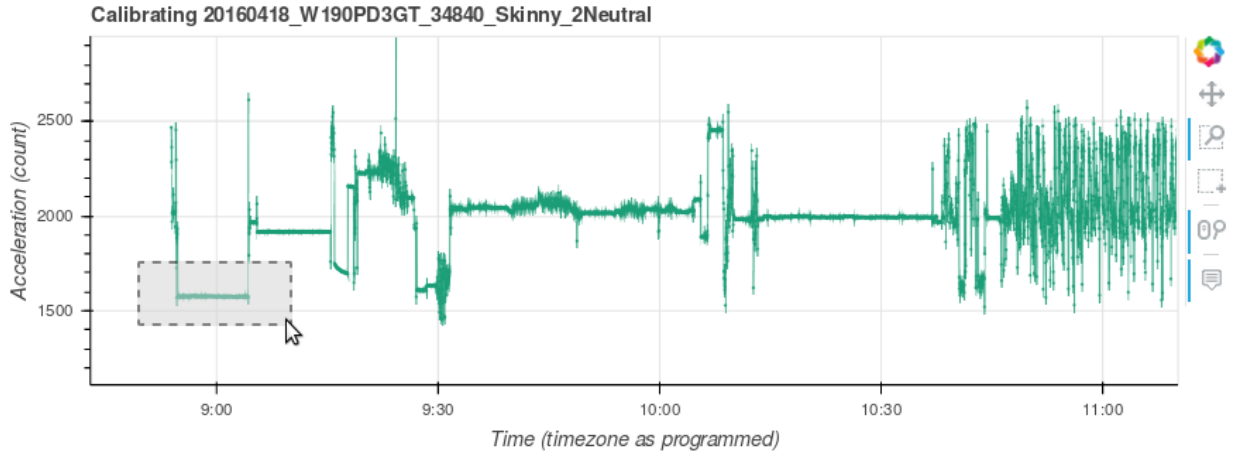
Assuming the calibration sequence was performed before the deployment of the datalogger, zoom this region of data using the  tool:



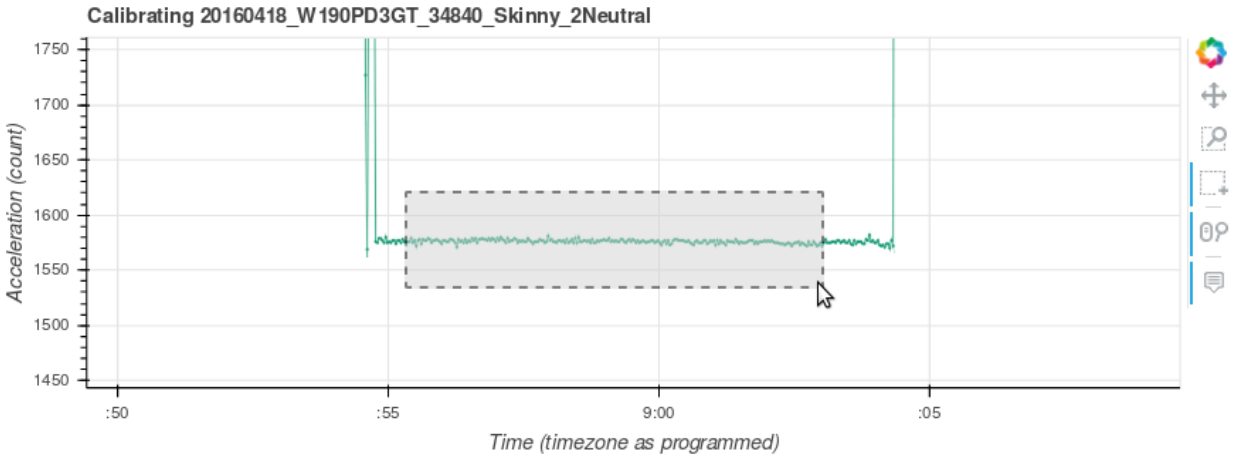
If the calibration sequence of orienting the datalogger was performed correctly, it should be obvious to see where the +g/-g positions are in the data:



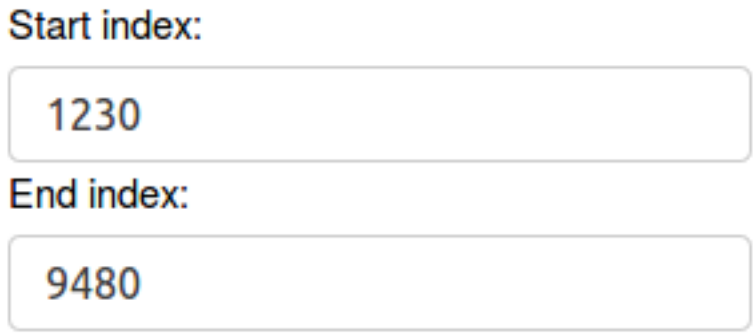
Zoom in again to the region corresponding with the bound you are selecting indices for, "lower" or "upper":



Then using the  tool, click and drag across a section of data without large amounts of variation.



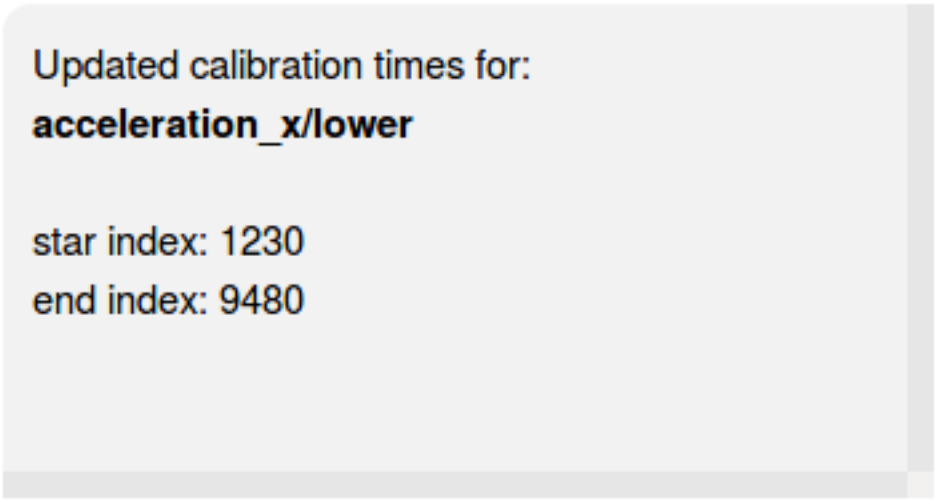
Notice that the *start* and *end* index position values have updated to the positions of the start and end of the horizontal area selected:



Saving the index values

Once you have *start* and *end* index values for the region you are working with (e.g. *accelerometer_x/lower*), Click the button labeled "Save Index Values":

You should then see a message displayed in the gray box to the right of the selection menu letting you know that the index positions for that region saved correctly to the *cal.yml*. This message includes the data parameter and bound you have selected and the *start* and *end* index positions you have selected:



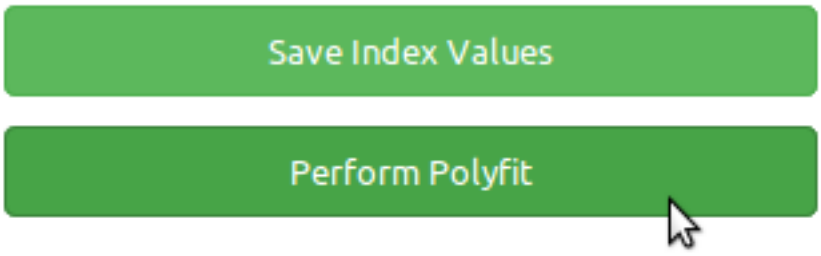
Once completed, you can zoom out again using the tool to perform these steps on the "upper" region. Be sure to select the correct data parameter and bound before saving the next index positions.

Then repeat these steps for the *x* and *y* axes until you have saved the index positions for all calibration orientation regions:

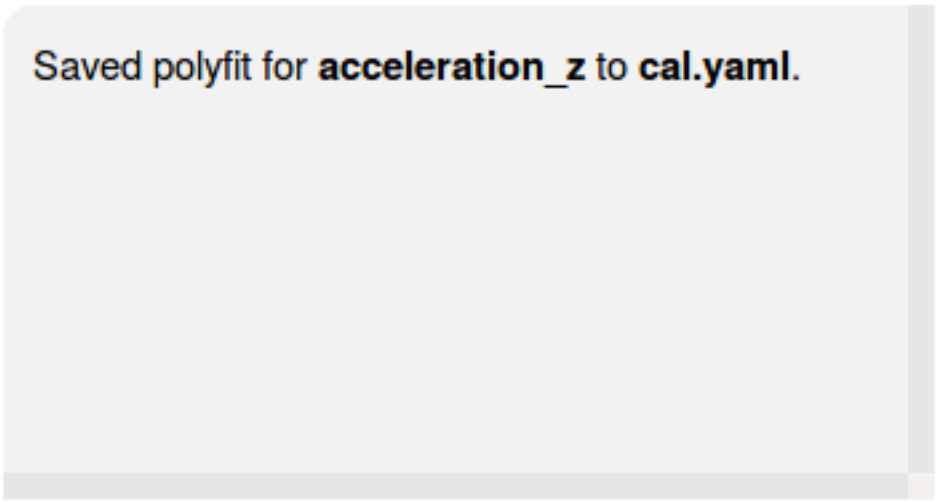
- *acceleration_x/lower*
- *acceleration_x/upper*
- *acceleration_y/lower*
- *acceleration_y/upper*
- *acceleration_z/lower*
- *acceleration_z/upper*

Saving the polyfit coefficients

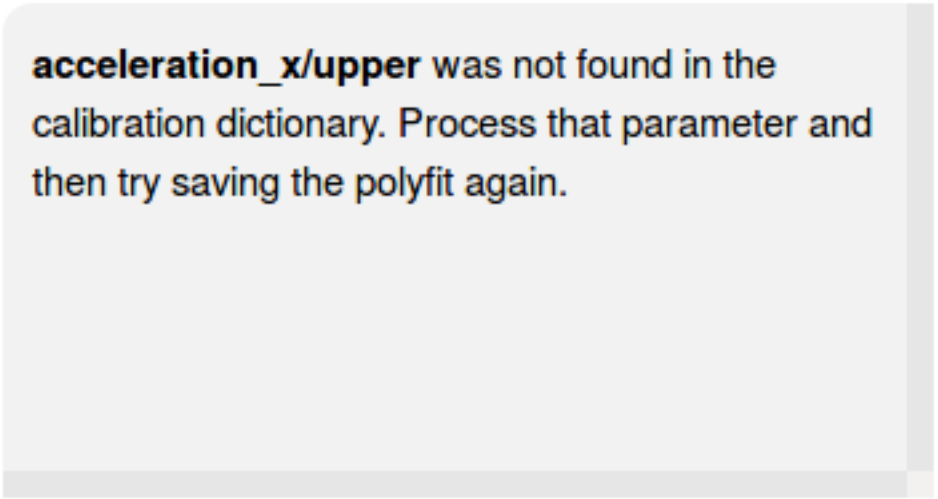
Once you have saved all of the index positions for all calibration orientation regions, click the button labeled "Perform Polyfit":



If the coefficients were able to successfully save to the *cal.yml* file, you should get a message in the gray box as follows:



If you are missing any index positions, you will get a message indicating the first of the missing regions you must select and save before you can perform the polyfit:



2.2 Propeller

2.2.1 Collecting calibration data

First configure and activate the datalogger for recording. You must then move water over the datalogger's propeller at known speeds, logging the speed of water movement, the exact start, and exact end times in spreadsheet with a preceding *id* column, saving it as a *csv* file as shown below.

As with the accelerometer file, a calibration of the propeller sensor should be performed within approximately 1 month of each deployment of the datalogger.

```
id,start,end,est_speed,count_average
00,start,end,speed,
...
99,start,end,speed,
```

2.2.2 Running the calibration software

With the collected data loaded using *pylleo*. Find the timestamp in *data['datetimes']* closest to the logged start and end times, then calculate the average count the propeller turned between each sample.

```
from pylleo import lleocal

cal_fname = 'speed_calibrations.csv'

cal = lleocal.create_speed_csv(cal_fname, data)
data = lleocal.calibrate_propeller(data, cal_fname)
```

CHAPTER 3

Building the documentation

3.1 napoleon

The source code is documented using the Numpy documentation style, which requires the extension napoleon for sphinx to correctly parse documentation from the source code docstrings.

Install napoleon:

```
pip install spinxcontrib-napoleon
```

See the end of the conf.py file to see the napoleon options for compiling.

3.2 Compiling

Then you can build the documentation using the sphinx Makfile by running the following in *pylleo*'s installation directory location:

```
make html
```

CHAPTER 4

Pylleo API documentaiton

4.1 lleoio

pylleo.lleoio.**read_data**(*meta*, *path_dir*, *sample_f=1*, *decimate=False*, *overwrite=False*)
Read accelerometry data from leonardo txt files

Parameters

- **meta** (*dict*) – Dictionary of meta data from header lines of lleo data files
- **path_dir** (*str*) – Parent directory containing lleo data files
- **sample_f** (*int*) – Return every *sample_f* data points

Returns

- **acc** (*pandas.DataFrame*) – Dataframe containing accelerometry data on x, y, z axes [m/s^2]
- **depth** (*pandas.DataFrame*) – Dataframe containing depth data [m]
- **prop** (*pandas.DataFrame*) – Dataframe containing speed data from propeller
- **temp** (*pandas.DataFrame*) – Dataframe containing temperature data

pylleo.lleoio.**read_meta**(*path_dir*, *tag_model*, *tag_id*)
Read meta data from Little Leonardo data header rows

Parameters

- **path_dir** (*str*) – Parent directory containing lleo data files
- **tag_model** (*str*) – Little Leonardo tag model name
- **tag_id** (*str, int*) – Little Leonardo tag ID number

Returns **meta** – dictionary with meta data from header lines of lleo data files

Return type dict

4.2 lleocal

pylleo.lleocal.**fit1d**(*lower*, *upper*)
Fit acceleration data at lower and upper boundaries of gravity

Parameters

- **lower** (*pandas dataframe*) – slice of lleo datafram containing points at -1g calibration position
- **upper** (*pandas dataframe*) – slice of lleo datafram containing points at -1g calibration position

Returns **p** – Polynomial coefficients, highest power first. If y was 2-D, the coefficients for k-th data set are in p[:,k]. From *numpy.polyfit()*.

Return type ndarray

Note: This method should be compared agaist alternate linalg method, which allows for 2d for 2d poly, see - http://stackoverflow.com/a/33966967/943773

A = numpy.vstack(lower, upper).transpose() y = A[:,1] m, c = numpy.linalg.lstsq(A, y)[0]

---

pylleo.lleocal.**get_cal_data**(*data_df*, *cal_dict*, *param*)
Get data along specified axis during calibration intervals

Parameters

- **data_df** (*pandas.DataFrame*) – Pandas dataframe with lleo data
- **cal_dict** (*dict*) – Calibration dictionary

Returns

- **lower** (*pandas dataframe*) – slice of lleo datafram containing points at -1g calibration position
- **upper** (*pandas dataframe*) – slice of lleo datafram containing points at -1g calibration position

See also:

lleoio.read_data() creates pandas dataframe *data_df*

read_cal() creates *cal_dict* and describes fields

pylleo.lleocal.**read_cal**(*cal_yaml_path*)
Load calibration file if exists, else create

Parameters **cal_yaml_path** (*str*) – Path to calibration YAML file

Returns **cal_dict** – Key value pairs of calibration meta data

Return type dict

pylleo.lleocal.**update**(*data_df*, *cal_dict*, *param*, *bound*, *start*, *end*)
Update calibration times for give parameter and boundary

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index