High-speed Event Retrieval and Management for Enhanced Spectral neutron imaging with TPX3Cams

HERMES Manual! Everybody cheer!

Version 8.7.2025

Contents

1. Overview

HERMES comprises a set of Python and C/C++ libraries (not a standalone program) designed to support the development of custom code for acquiring, processing, and analyzing data from the TPX3Cam manufactured by Amsterdam Scientific Instruments. HERMES' primary function is providing the foundational tools needed for users to create applications tailored to their specific requirements in energy-resolved neutron imaging with TPX3Cams, along with its subsequent analysis. With HERMES users have access to a flexible framework that simplifies the integration of TPX3Cam's capabilities into their projects, while also providing the needed diagnostics for development and trouble shooting.

HERMES has three primary functions:

1) Acquire

Easily connect to and collect data from TPX3Cams to simplify the data acquisition process.

2) Unpack

Unpack the .tpx3 binary files from acquisition and unpack into a .rawSignals file.

3) Analyze

Provide basic data analysis tools for users of TPX3 camera systems.

2. Installation

To install HERMES, you need to install both pixi and the HERMES directory from GitHub. Currently, HERMES is not supported by Windows, only Linux/MacOS systems. HERMES has been shown to work on WSL environments.

2.1 Linux/MacOS Instructions

Run the following code in order, in the directory that you would like HERMES to be installed:

```
curl -fsSL https://pixi.sh/install.sh | sh
git clone https://github.com/lanl/HERMES.git
```

Now, `cd` into your HERMES directory that was just created. If you type `pwd`, you should see something like this output:

```
/Users/my_username/Programs/hermes
```

If not, double check where HERMES has been installed. If everything is looking good so far, run:

```
pixi install
pixi shell
pixi run build-cpp
```

Pixi will automatically install required dependencies and manage the python version for HERMES. If everything worked correctly, you should see the text (hermes) in your terminal preceding the current address:

```
(hermes) user:~/HERMES
```

If the prefix is shown, HERMES has successfully been installed and is ready for use. There are many minor variations on the second part of this address depending on what terminal you are using (bash, zsh, etc.) but the `(hermes)` will always show. That is what matters. If this does not show, ensure that pixi was installed properly. To test getting back into this pixi environment, run `pixi shell` again.

2.2 Copying necessary files into workspace

Many necessary files are needed to acquire, unpack, and analyze data. Below is a line for each of these operations. If any of these commands fail, ensure you are in the home directory for HERMES (this contains the docs, src, examples, and workspace folders).

Acquiring data - This copies the acquisition python file and configuration file to the workspace.

```
cp examples/scripts/acquire_data/acquireTpx3.py workspace &&
cp examples/scripts/acquire_data/acquireTpx3.ini workspace
```

Unpacking data - This copies the unpacker binary and unpacker configuration file to the workspace. This will only work if you have ran `pixi run build-cpp`. You can alternatively `cd` into `src/chermes` and run `make`.

```
cp src/chermes/unpacker.config workspace/ && cp
src/chermes/bin/tpx3SpidrUnpacker workspace
```

Analyzing data - This copies the analysis example notebook to the workspace.

```
cp examples/notebooks/analysis_hermes.ipynb workspace
```

This provides a basic starting point for HERMES users. See respective sections in the rest of the manual to understand how to use these basic files.

3. Data Acquisition

3.1 System Overview and Prerequisites

The acquisition scripts interface with the TPX3Cam and SPIDR readout boards using the `tpx3serval` Python file. These scripts can configure the camera, setting up run directories, logging configuration files, and perform acquisition runs.

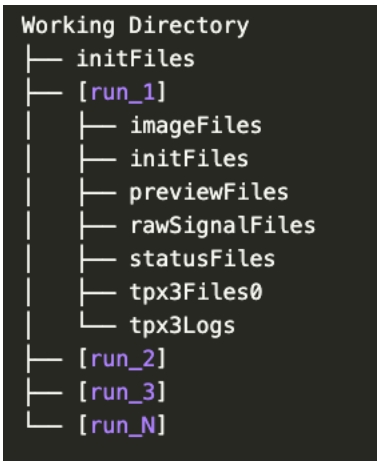
Currently, there are multiple processes that must be done manually before acquisition can occur. Your system must have Serval, created by ASI, to connect the SPIDR readout board to the data acquisition computer (daq) using a local server. Your system must have SoPhy, created by ASI, to perform calibration of the TPX3Cam. See Appendix A for gathering these calibration files. Lastly, your system must have Zaber controls to properly adjust bias voltage to an image intensifier. These processes will eventually be implemented directly into HERMES, yet they currently are not. sorry :(

To acquire data, the TPX3Cam must already be running and connected to Serval. To operate different camera procedures, such as starting or stopping measurement, an operator must direct the server to different addresses. This step is annoying to perform manually, so HERMES performs these operations instead. For more information on how Serval interacts with the TPX3Cam, refer to the Serval manual.

3.2 Directory Structure

HERMES has a structured directory layout. The working directory contains one folder for each run and several subfolders for specific data files. The working directory location is defined in the acquisition configuration file, initially copied over as `acquireTpx3.ini`.

Example Directory Layout:



The working directory is the location that you want all data to be saved. The only thing you must do before acquiring data inside of this directory is create the `initFiles` directory for the initial detector settings and the initial server destination.

The `acquireTpx3.py` script will automatically generate the run folders and their contents automatically, before starting to acquire .tpx3 files.

3.3 Command Line Interface (CLI)

3.3.1 Default Behavior

The CLI provides a flexible way to run acquisitions. Defaults are built into the script; a configuration file and/or CLI flags can override these defaults.

Usage:

In the same directory as the acquireTpx3.py script, run:

```
python acquireTpx3.py [options]
```

By default, no configuration file is required. If using defaults, file/folder locations will still need to be changed. Configuration options can be provided via:

1. Config file (`-c` or `--config`)
2. CLI flags (highest precedence)

Default Behavior (no config):

- Trigger period: 10 s
- Exposure time: 9 s
- Number of runs: 1

3.3.2 CLI Flags

General Options

- `-h, --help`: Information on available commands
- `-c, --config`: Path to config file
- `-W, --working-dir`: Working directory path
- `-r, --run-name`: Run name (folder name and filename prefix)
- `-N, --run-number`: Starting run number (integer, zero-padded as 0000)
- `-n, --num-runs`: Total number of runs
- `-t, --trigger-period`: Trigger period (s), must be >0.
- `-e, --exposure`: Exposure time (s)
- `-T, --num-triggers`: Number of triggers per run
- `-v, --verbose`: Verbosity (0, 1, or 2)
- `--dry-run`: Print effective configuration and exit. Does not acquire data.

If the help command is run, the following output will be placed in the terminal:

```
usage: acquireTpx3.py [-h] [-c CONFIG] [-W WORKING_DIR] [-r RUN_NAME] [-N RUN_NUMBER] [-n NUM_RUNS] [-t TRIGGER_PERIOD]
                      [-e EXPOSURE] [-T NUM_TRIGGERS] [--dry-run] [-v {0,1,2}]

TPX3 Data Acquisition

options:
  -h, --help            show this help message and exit
  -c, --config CONFIG   Optional config file to override defaults
  -W, --working-dir WORKING_DIR
                        Working directory path
  -r, --run-name RUN_NAME
                        Override run name (used in folder names)
  -N, --run-number RUN_NUMBER
                        Initial run number (int)
  -n, --num-runs NUM_RUNS
                        Number of runs
  -t, --trigger-period TRIGGER_PERIOD
                        Trigger period [s]
  -e, --exposure EXPOSURE
                        Exposure time [s]
  -T, --num-triggers NUM_TRIGGERS
                        Number of triggers per run
  --dry-run             Show the effective config and exit
  -v, --verbose {0,1,2}
                        Verbosity level (0=quiet, 2=very verbose)
```

3.4 Configuration File

The `acquire_config.ini` file defines all configurable parameters for acquisition.

Sections and Parameters:

- [WorkingDir]
  - `path_to_working_dir`: Full path to working directory (required)
  - `path_to_init_files`: Path for initialization files
  - `path_to_status_files`: Path for status files

- `path_to_log_files`: Path for log files
- `path_to_image_files`: Path for image files
- `path_to_preview_files`: Path for preview files
- `path_to_rawSignal_files`: Path for .rawSignals files
- `path_to_raw_files`: Path for raw .tpx3 files

- [ServerConfig]

  - `serverurl`: URL for TPX3Cam server (default: `http://localhost:8080`)
  - `path_to_server`: Path to the Serval directory
  - `path_to_server_config_files`: Path to camera settings directory relative to Serval directory
  - `bpc_file_name`: Pixel configuration filename (from SoPhy Calibration)
  - `dac_file_name`: DAC configuration filename (from SoPhy Calibration)
  - `destinations_file_name`: Server destinations file
  - `detector_config_file_name`: Detector configuration file

- [RunSettings]

  - `run_name`: Name for the run (used as folder name and in filenames)
  - `run_number`: Starting run number (default: 0000)
  - `trigger_period_in_seconds`: Camera trigger period
  - `exposure_time_in_seconds`: Exposure time (must be ≤ trigger period)
  - `trigger_delay_in_seconds`: Delay before triggers
  - `number_of_triggers`: Number of triggers per run
  - `number_of_runs`: Total number of runs to perform
  - `global_timestamp_interval_in_seconds`: Timestamp interval

3.5 Examples

Use defaults and specify working directory:

```
python acquireTpx3.py -W /data/acquisition_test
```

Load a config file and override exposure time:

```
python acquireTpx3.py -c acquire_config.ini -e 5
```

Specify working directory, run name, and number of runs:

```
python acquireTpx3.py -W /data/beam_test -r beam 2025 -n 10
```

Load config file and override multiple parameters:

```
python acquireTpx3.py -c acquire_config.ini -e 7 -t 12 -T 25
```

Full example with almost all parameters:

```
python acquireTpx3.py -c acquire_config.ini -W /data/full_test
-r complex_run -N 5 -n 8 -t 15 -e 12 -T 30 -v 2
```

3.6 Acquisition Process Flow

To acquire data, the TPX3Cam goes through 4 steps:

1. Configuration**:** Script merges defaults, config file, and CLI flags.
2. Directory Verification**:** Working directory and run folder are created or cleaned.
3. Camera Check**:** TPX3Cam connection is verified.
4. Run Execution**:**

    - Run number is incremented and formatted.
    - Configuration files and detector status are logged.
    - Exposure is started using the configured parameters.
    - Data is written into the appropriate subdirectories.

For details on TPX3Cam server and dashboard behavior, refer to the Serval Camera Manual.

4. Unpacking Data

Unpacking data in HERMES requires use of C++ files to turn .tpx3 files into .rawSignals files. The .tpx3 files are created from acquiring raw data from the TPX3Cam, and the .rawSignals is a usable form that can easily be loaded into a pandas DataFrame.

4.1 Create Unpacker

In a terminal, navigate to your HERMES directory. This should be the directory that contains folders such as `src`, `workspace`, and `examples`. In this directory, run:

```
cd src/chermes && make && cp unpacker.config ../../workspace/
&& cp bin/tpx3SpidrUnpacker ../../workspace/ && cd ../../
```

This will both create the binary file to run the unpacker and copy it into the workspace area, along with a default configuration file. If you try to run this command multiple times, you may get the error:

```
make: Nothing to be done for 'all'.
```

In this case, navigate to `/src/chermes/bin` and delete tpx3SpidrUnpacker. Re-run the code and a new version should be created and copied into the workspace. Make sure that you have created an up-to-date version of the unpacker.

4.2 Using the Unpacker

4.2.1 Using the CLI

To run and test the unpacker, navigate to `workspace`. The HERMES unpacker utilizes a command line interface format to unpack data files from a location. You can view the help menu by inputting one of the following into your terminal.

`tpx3SpidrUnpacker` OR `tpx3SpidrUnpacker -h` OR `tpx3SpidrUnpacker --help`

The following menu appears when running one of these commands.

```
(hermes) (base) ➜  workspace git:(v2.0) ✗
(hermes) (base) ➜  workspace git:(v2.0) ✗ tpx3SpidrUnpacker
Error: Please provide command-line arguments.
Input/Output Options:
  -i, --inputFile <file>     Input TPX3 file
  -I, --inputDir <dir>       Input directory (for batch mode)
  -b, --batch                Enable batch mode (requires -I <directory>)
  -o, --outputDir <dir>      Output directory
  -c, --configFile <file>    Configuration file

Processing Options:
  -s, --sort                 Enable signal sorting
  -w, --writeRawSignals      Enable writing raw signals
  -W, --no-writeRawSignals   Disable writing raw signals
  -C, --clusterPixels        Enable pixel clustering
  -p, --writeOutPhotons      Enable writing photon data
  -H, --fillHistograms       Enable histogram filling

Clustering Parameters:
  -S, --epsSpatial <n>       Spatial epsilon for clustering (pixels)
  -T, --epsTemporal <n>      Temporal epsilon for clustering (seconds)
  -P, --minPts <n>           Minimum points for clustering
  -q, --queryRegion <n>      Query region for clustering

Diagnostic Options:
  -m, --maxPackets <n>       Maximum packets to read (0=all)
  -v, --verbose <level>      Verbose level (0-3, default: 1)
  -h, --help                 Show this help message
(hermes) (base) ➜  workspace git:(v2.0) ✗
```

All these parameters are present in the configuration file, except for `-c`, which specifies if you would like to use a configuration file. See examples in section 4.2.4 to see `-c` being used properly. Other parameters will be thoroughly explained in section 4.2.2.

4.2.2 Configuration File

Like HERMES acquisition, all these parameters can be specified in a configuration file. The template for the unpacker configuration file is given in `unpacker.config`, which was automatically copied into the workspace alongside the binary file.

Below are the parameters present in the configuration file and their purpose:

Input/Output options:

- `rawTPX3Folder, -I, --inputDir` : Directory to location with desired .tpx3 files.
- `rawTPX3File, -i, --inputFile` : File name of single .tpx3 file to analyze. Set as 'ALL' to process all files in a folder.
- `outputFolder, -o, --outputDir` : Output directory that .rawSignals files go into. Defaults to `rawSignalFiles` folder in same directory as `rawTPX3Folder`.
- `-c, --configFile` : Path to configuration file if using CLI.

Processing options:

- sortSignals, -s, --sort : Sort signals as they unpack
- writeRawSignals, -w, --writeRawSignals : Create .rawSignals files from the .tpx3 files.
-

```
# Path to folder with .tpx3 files
rawTPX3Folder = PATH/TO/YOUR/FOLDER

# File to process (or use ALL for batch mode)
rawTPX3File = ALL

# Output directory
outputFolder = PATH/TO/YOUR/FOLDER

# Signal processing options
writeRawSignals = true
sortSignals = true
fillHistograms = false
clusterPixels = false
writeOutPhotons = true

# Verbosity level
verboseLevel = 2

# Packet limits, set to 0 to unpack all packets
maxPacketsToRead = 0

# Clustering parameters
queryRegion = 0
epsSpatial = 2
epsTemporal = 500.0
minPts = 3
```

A clustering function is built into the unpacker of HERMES; however, many users may desire to cluster using custom parameters or functions, thus allowing for `clusterPixels` to be enabled or disabled. Below are definitions for each customization option:

`rawTPX3Folder =` The path to your .tpx3 directory that you want to unpack.

`rawTPX3File =` The filename of a specific .tpx3 file you want to unpack. Set to 'ALL' for batch mode, unpacking every file in `rawTPX3Folder`.

`outputFolder =` The path to your .rawSignalFiles directory that you want unpacked files to be saved in.

`writeRawSignals =` Enable/Disable ability to write raw signals.

`sortSignals =` Enable/Disable ability to sort signals.

`clusterPixels =` Enable/Disable ability to cluster pixel hits.

`writeOutPhotons` = Enable/Disable ability to write photons.

`verboseLevel` = Gives user detailed terminal output depending on value.

0 = Silent mode, 1 = Basic Information, 2 = Detailed logs.

`maxPacketsToRead` = Maximum number of packets to read. 1 packet is 64 bits, or generally is one 'event', whether that be a TDC, pixel hit, GTS, or control signal. Set to 0 to unpack all packets.

4.2.3 .rawSignals Structure

The HERMES unpacker saves unpacked files as .rawSignals files, which contain the following structure:

```
// Represents the data for a single raw signal.
struct signalData {
    uint32_t bufferNumber;  // Buffer number from where signal was recorded
    uint8_t signalType;     // Type of the signal (TDC=1,Pixel=2,GTS=3)
    uint8_t xPixel;         // X-coordinate of the pixel
    uint8_t yPixel;         // Y-coordinate of the pixel
    double ToaFinal;        // Time of Arrival in seconds (final value)
    uint16_t TotFinal;      // Time over Threshold in nano seconds (final value)
    uint32_t groupID;       // Group ID for clustering.
};
```

This structure is further unpacked into an array with basic processing. See section 5.3 for HERMES functions to analyze .rawSignals files.

Generally, useful data will have a `signalType` of 1, 2, or 3. A `signalType` of 4 or 5 shows a SPIDR Control signal and TPX3 Control signal respectively, and any other `signalType` value is not useful and can be discarded. We are in the process of refining this unpacking to produce less useless information and retain a more memory-efficient system.

4.2.4 Examples

Unpack a single .tpx3 file with default settings:

```
tpx3SpidrUnpacker -i data/run0001.tpx3 -o output/
```

Unpack a single .tpx3 file with a configuration file:

```
tpx3SpidrUnpacker -i data/run0001.tpx3 -o output/ -c
unpacker.config
```

Process an entire directory in batch mode:

```
tpx3SpidrUnpacker -I data/ -b -o output/
```

Unpack and cluster pixels with custom parameters:

```
tpx3SpidrUnpacker -i data/run0002.tpx3 -o output/ -C -S 2 -T
5e-9 -P 3
```

Unpack using most available options:

```
tpx3SpidrUnpacker -I data/runs/ -b -o output/full_test/ -c -
unpacker.config -s -w -p -H -C -S 3 -T 1e-8 -P 5 -q 10 -m
50000 -v 2
```

5. Analyzing Data with HERMES packages

HERMES offers a handful of features to allow for data analysis of .rawSignals, .csv, or .pixelActivations files. These files are acquired from different unpacker sources:

Program	Executable	Output File Extension
HERMES	.tpx3	.rawSignals
EMPIR	.exportpixels	.pixelActivations

HERMES has several built-in packages that allow for quick and easy data analysis. These packages are in the `src/hermes/analysis` folder. To use these packages, include the following headers as part of your imports:

```
from hermes.analysis.loader import SignalsIO

from hermes.analysis.plotter import BufferPlotter,
HistogramPlotter, ToAImageSequenceGenerator

from hermes.analysis.analyzer import SignalAnalyzer
```

Note that any changes made to a package file will only be implemented if the kernel is restarted to load the package once again. Below are all the packages currently implemented into HERMES and the abilities their functions provide.

5.1 loader.py

This module provides functionality to load various file types (.rawSignals, .csv, .pixelActivations) and convert them into pandas DataFrames. The loader also can export these same pandas DataFrames into .csv and .parquet files. Together, the loader.py module provides three functions for users to utilize in their analysis.

5.1.1 Load Function

The load function is a powerful tool to easily load individual files or a folder of files into a pandas DataFrame. At its minimum, the load function only requires a path to either a single file or a folder. The base case looks like this:

```
df = loader.load_data("path/to/file-or-folder")
```

The loader can automatically detect the extension of the files you are using and load them with their according structure. Once loaded, all files are concatenated into a single DataFrame with the following structure:

bufferNumber	signalType	xPixel	yPixel	ToaFinal	TotFinal	groupId	signalTypeDescription
uint16	uint8	uint8	uint8	float64	float32	uint16	category

Each row, or 'packet', is equivalent to a single event taken on a TPX3 camera. There will be as may rows as there are packets.

The load function allows for many other parameters besides a path. Below are other parameters and a description:

- `format : str` - Format of the file to read. Required if multiple valid filetypes in a folder.
- `index : str` - Index of files to read in folder. Can include start, stop, and step separated by a colon.
- `time_adjust : Boolean` - Enable ToA continuity across files by adding offset to ToA based on index. Note: If using time_adjust and an index step, the time ToA data will have 'gaps' where a file was skipped over.
- `round_period_to : float` - Helps time_adjust round to a certain window. Default of 0.5 s.

- `file_duration : float` - If defined, time_adjust does not assume/round offset and uses a constant duration for offset instead.

See examples in 5.4 for proper usage/syntax.

5.1.2 Exporter Functions

HERMES has two basic functions to export a pandas DataFrame to either a .csv file or a .parquet file. Both functions are very simple and require a DataFrame and a string path to save the file. Here is an example of each of these exporter functions:

```
reader.export_to_csv(df, "path/to/save/data.csv")
reader.export_to_parquet(df, "path/to/save/data.parquet")
```

If only a filename is provided instead of a full path, the file will save in the same directory as the python file where the function was called. Generally, this would be the workspace.

5.2 plotter.py

This module has various plotting functions that allow users to quickly visualize their data without spending time messing with matplotlib's intricacies. There are three important classes, each with a handful of functions. These functions will not be heavily documented as they are changed often and are intended for quick data visualization, not high-quality figures.

5.2.1 BufferPlotter

This class will create different plots based on the `bufferNumber` in a data set. The functions are:

- `plot_3d_pixels_vs_toa()` - Plot pixels in a 3D space with x position, y position, and ToA as z position. Color codes clusters based on `bufferNumber`.
- `plot_tot_image()` - Plot an image of ToT frequencies for a given data set.

5.2.2 HistogramPlotter

This class will plot various types of 1D and 2D histograms. The functions are:

- `plot_packets_per_buffer()` - Plots the number of packets in a buffer.
- `plot_2D_histogram()` - Creates a heatmap of how many pixels are hitting a specific position.

5.2.3 ToAImageSequenceGenerator

This class creates animations and gifs based on the ToA of dataset. The functions are:

- `generate_images()` - Creates a specified number of images to later be turned into a gif.
- `compile_images_to_gif()` - Compiles folder of images into a gif.

5.3 analyze.py

This module contains basic functions for users to perform basic data analysis. This module is the simplest and least refined module of HERMES analysis tools because much analysis requires specialized tools or niche applications. These tools included are meant to be as useful as possible.

5.3.1 Summary/Diagnostic Functions

This module contains one summary function that prints basic information about the loaded DataFrame. To call this function, run:

```
analyzer.get_summary_stats(df, rows=#)
```

`rows` is how many rows you would like to preview. The default value for `rows` is 10. This function will print out how many signals were loaded, the columns of the DataFrame, the distribution of signal types, the time and pixel range, the number of buffers and groups, and a preview of the first few rows.

5.3.2 Filtering Functions

HERMES has built-in functions to filter by a specific signalType or a time range.

For `filter_by_signal_type`, you need a DataFrame and a string `signalType` of 'TDC', 'Pixel', 'GTS', 'TPX3_Control', or 'SPIDR_Control'.

For `filter_by_time_range`, you need a DataFrame, a float `startTime` and a float `endTime`. Both times must be in seconds. See examples for implementation of both.

With this filtering, you can then use some of the functions in plotter.py to visualize data.

5.4 Coding Examples

5.4.1 Loader Examples

Setting various paths to different data:

```
rawSignals_dir = "PATH/TO/YOUR/RAWSIGNALS/DATA"
csv_dir = "PATH/TO/YOUR/CSV/DATA"
pixelActivations_dir = "PATH/TO/YOUR/PIXELACTIVATIONS/DATA"
```

Load SignalsIO() class into object:

```
loader = SignalsIO()
```

Load a single .rawSignals, .csv, and .pixelActivations file:

```
df1 = loader.load_data("path/to/data.rawSignals")
df2 = loader.load_data("path/to/data.csv")
df3 = loader.load_data("path/to/data.pixelActivations")
```

Load the .rawSignals files form a folder that has many different file types:

```
df = loader.load_data("path/to/mixed/data",
format="rawSignals")
```

Load the .csv files from a folder with many file types and only load files 5 through 9:

```
df = loader.load_data("path/to/mixed/data", format="csv",
index="5:10")
```

Load the .pixelActivations files from directory, load files 10 through 19 with a step of 2, and enable ToA continuity:

```
df = loader.load_data(pixelActivations_dir, index="10:20:2",
time_adjust=True)
```

Load .rawSignals files from directory, load files 10 through 100 with a step of 5, enable ToA continuity with rounding estimation:

```
df = loader.load_data(rawSignals_dir, index="10:101:5",
time_adjust=True, round_period_to=0.25)
```

Load .rawSignals files from directory, load files 10 through 100 with a step of 5, enable ToA continuity with defined file duration:

```
df = loader.load_data(rawSignals_dir, index="10:101:5",
time_adjust=True, file_duration=2.5)
```

5.4.2 Plotter Examples

Filter DataFrame to only be pixels, then plot a heatmap of pixel hits.

```
pixel_df = analyzer.filter_by_signal_type(df, "Pixel")
HistogramPlotter.plot_2D_histogram(pixel_df)
```

5.4.3 Analyzer Examples

Load SignalAnalyzer() class into object:

```
analyzer = SignalAnalyzer()
```

Get summary of data that has been loaded:

```
analyzer.get_summary_stats(df)
```

Filter DataFrame to only show pixels:

```
pixel_df = analyzer.filter_by_signal_type(df, "Pixel")
```

Filter DataFrame to be between time window of 1.5 and 2 seconds:

```
time_df = analyzer.filter_by_time_range(df, 1.5, 2.0)
```

5.5 Example Notebook Files

HERMES contains an example notebook with many of the function shown above in the file `analysis_hermes.ipynb`. This file is laid out in a way that is intended to be as easy as possible to get started with. It is recommended to use the extension 'Data Wrangler' to easily visualize data in a Jupyter notebook.

Appendix!!!!!!!

Appendix A: Serval Usage

will write later.

Appendix B: SoPhy Calibration

In SoPhy, you must calibrate the camera to properly detect when pixels should activate and to mask out any pixels that are not behaving as they should. This will create a `.bpc` and a `.dac` file, both of which are important for data acquisition.

In the SoPhy side menu, navigate to the Setup tab on the bottom, denoted by a wrench and screwdriver icon. In the Devices folder, navigate to your specific device, then `HW Info 0`. Within this folder, ensure that `BiasSupplyEnable=true` and `BiasAdjust=40`. If not, adjust them.

Navigate to the Exposure/Record tab on the bottom, denoted by a camera icon. Set the measurement type to equalization. Click the gear icon next to this drop-down menu, and set the pre-set parameters to `PRECISE`.

Navigate back to the first window, and in the top bar click on preview. Open a window for equalization. Now, click on 'start', just below where you pick the correct device. Equalization will begin and continue for several minutes.

ADD SCREENSHOTS

When finished, a plot will pop up showing nosy pixels. Ensure that it seems reasonable. With all of this, there are now several files that need to be saved. This can be done by clicking "File -> Medipix/Timepix control -> Export pixel config". Save this inside the CameraSettings folder in your Serval directory. By default, it is simplest to just label the file as 'settings' and not provide an extension. This will export 2 files, a `.bpc` and a `.bpc.dac`. Both are essential. It is also common to include another file specifically for the camera ID and date, so for example the directory might be:

```
Programs/TPX3CAM/Serval/Serval_2.1.6/CameraSetting/230010078/20250806
```

Where `'230010078'` is the specific camera ID and `'20250806'` is the date (August 6, 2025). This will make it easier to set up the acquisition configuration file.