

CUAV Field Tests

Data Reader Project

High-Level Documentation

Generated: 2025-11-20 19:23:16

Table of Contents

1. Project Overview
2. System Architecture
3. Data Flow
4. Core Methods
5. Data Processing Pipeline
6. Database Storage
7. Analysis and Visualization
8. Configuration
9. Usage Examples

1. Project Overview

The CUAV Field Tests Data Reader is a Python package designed for processing, analyzing, and visualizing atmospheric measurement data from RaymetRICS CUAV (Coherent Doppler Lidar) field tests. The system handles data from multiple sources including processed measurement files, raw spectra files, and log files containing pointing angles and timestamps.

Key Features:

- Automated matching of processed and raw data files across mirrored directory structures
- Timestamp-based filtering and synchronization with log files
- Data aggregation from multiple sources (peak, spectrum, wind, raw spectra)
- SQLite database for persistent storage and efficient querying
- Range-resolved heatmap visualization for spatial analysis
- Single-profile mode for SNR and wind profile analysis with Doppler lidar processing
- Configurable processing parameters via simple text file

Main Use Cases:

- Match processed timestamps with corresponding raw spectra files
- Filter data to include only measurements present in log files
- Aggregate data from multiple file types into unified structures
- Store aggregated data in a database for persistent access
- Generate heatmaps showing parameter distributions across azimuth/elevation angles
- Analyze range-resolved profiles at specific altitudes
- Process power density spectra to compute SNR and wind profiles using Doppler lidar equations
- Visualize temporal evolution of SNR and wind profiles across all timestamps

2. System Architecture

The system is organized into modular packages, each responsible for a specific aspect of data processing:

The **parsing** module extracts metadata from filenames and log files. Key functions include `timestamp_from_spectra_filename()` for parsing datetime information from spectra filenames and `read_log_files()` for loading log files containing azimuth, elevation, and timestamp data.

The **matching** module aligns processed and raw data by traversing directory trees. It provides `match_processed_and_raw()` to pair processed timestamps with raw spectra files and `filter_matches_by_log_timestamps()` to keep only matches present in the log file.

The **reading** module handles loading data files into NumPy arrays. It includes `read_processed_data_file()` for loading processed files such as `_Peak.txt`, `_Spectrum.txt`, and `_Wind.txt`, as well as `read_raw_spectra_file()` for loading raw spectra files.

The **processing** module aggregates, filters, and integrates data from multiple sources. It provides `build_timestamp_data_dict()` to aggregate data from all sources for each timestamp and `build_and_save_to_database()` to build and save aggregated data to the database in one step.

The **storage** module manages SQLite database for persistent data storage. It includes the `DataDatabase` class for advanced operations, `query_timestamp()` for querying a single timestamp, and `query_timestamp_range()` for querying a range of timestamps.

The **analysis** module generates visualizations and extracts range-resolved data. It provides `create_heatmaps()` to generate heatmaps for parameters at specific ranges, `extract_range_values()` to extract values from range-resolved profiles, `process_single_profiles()` to compute SNR and wind profiles from power density spectra, and `create_profile_visualizations()` to generate profile plots.

Module Dependencies:

The architecture follows a layered dependency structure: `parsing` → `matching` → `reading` → `processing` → `storage` → `analysis`. Lower-level modules (`parsing`, `matching`, `reading`) provide foundational data access, while higher-level modules (`processing`, `storage`, `analysis`) build upon these foundations to provide advanced functionality.

3. Data Flow

The system processes data through several stages, from raw file discovery to final visualization. The data flow begins with processed files containing measurement data (_Peak.txt, _Spectrum.txt, _Wind.txt) and a log file (output.txt) containing pointing angles and timestamps. Timestamps are extracted from both processed files and raw spectra filenames. Processed timestamps are then matched with raw spectra files (spectra_*.txt) based on index order. The matches are filtered to keep only those present in the log file. For each filtered timestamp, data is aggregated from all sources including peak, spectrum, wind, azimuth, elevation, and raw spectra. The aggregated data is stored in a SQLite database with timestamp as the primary key. Finally, heatmaps are generated by querying the database, extracting range-resolved values at specific ranges, and aggregating data by azimuth and elevation angles.

Data Flow Stages:

1. File Discovery:

Traverse processed and raw directory trees. Processed trees contain files like _Peak.txt, _Spectrum.txt, _Wind.txt. Raw trees contain spectra_*.txt files.

2. Timestamp Extraction:

Extract timestamps from processed files (third column) and from raw filenames (parsed from datetime format: spectra_YYYY-MM-DD_HH-MM-SS.ff.txt).

3. Matching:

Pair processed timestamps with raw files based on index order within each directory pair. Each processed timestamp corresponds to one raw file.

4. Filtering:

Keep only matches where the processed timestamp exists in the log file (within tolerance). This ensures data consistency.

5. Aggregation:

For each filtered timestamp, collect data from: processed files (peak, spectrum, wind), log file (azimuth, elevation), and raw spectra files (power density spectrum).

6. Storage:

Save aggregated data to SQLite database with timestamp as primary key. Arrays are stored as JSON strings.

7. Visualization:

Query database, extract range-resolved values at specific ranges, aggregate by azimuth/elevation, and generate heatmaps. Output files are organized into parameter-specific subfolders (wind_heatmaps/, snr_heatmaps/, etc.) within visualization_output//.

8. Single-Profile Processing (Mode 3):

Process range-resolved power density spectra to compute SNR and wind profiles. For each timestamp, compute frequencies from FFT size and sampling rate, find maximum SNR frequency for each range, compute wind speed using Doppler lidar equation, and store profiles in database. Output saved to single_profile/ subfolder.

9. Differences Processing (Mode 4):

Compute sequential differences in SNR and wind between adjacent range pairs. Output saved to snr_difference/ and wind_difference/ subfolders.

10. FWHM Processing (Mode 5):

Compute Full Width at Half Maximum of dominant frequency peaks. Output saved to fwhm/ subfolder.

4. Core Methods

Parsing Methods: The `timestamp_from_spectra_filename()` function extracts datetime information from spectra filenames following the format `spectra_YYYY-MM-DD_HH-MM-SS.ff.txt`. The `read_log_files()` function loads log files containing columns for azimuth, elevation, and timestamps, transposing the data for easier access.

Matching Methods: The `match_processed_and_raw()` function walks directory trees and pairs processed timestamps with raw spectra files based on index order. The `filter_matches_by_log_timestamps()` function keeps only matches whose processed timestamps exist in the log file, ensuring data consistency.

Reading Methods: The `read_processed_data_file()` function loads processed files such as `_Peak.txt`, `_Spectrum.txt`, and `_Wind.txt` into NumPy arrays. The `read_raw_spectra_file()` function loads raw spectra files, automatically skipping the first 13 header lines for ASCII format files.

Processing Methods: The `build_timestamp_data_dict()` function aggregates data from all sources (peak, spectrum, wind, azimuth, elevation, raw spectra) for each timestamp into a nested dictionary structure. The `build_and_save_to_database()` function builds and saves aggregated data to the database in one step.

Storage Methods: The `query_timestamp()` function queries a single timestamp from the database, returning all associated data. The `query_timestamp_range()` function queries a range of timestamps from the database, useful for time-series analysis.

Analysis Methods: The `extract_range_values()` function extracts values from range-resolved profiles at specific requested ranges. The `create_heatmaps()` function generates heatmaps for parameters at specific ranges, visualizing parameter distributions across azimuth and elevation angles. The `process_single_profiles()` function processes range-resolved power density spectra to compute SNR and wind profiles using Doppler lidar equations. The `create_profile_visualizations()` function generates plots of all SNR and wind profiles for temporal analysis.

5. Data Processing Pipeline

5.1 Matching Process

The matching process pairs processed data files with raw spectra files by traversing mirrored directory structures. Both trees follow the pattern: *Wind/YYYY-MM-DD/MM-DD_Hhh/MM-DD_HH-##/*

Within each matched directory pair, files are sorted by timestamp and paired by index. Each processed timestamp (from *_Peak.txt*, third column) is matched with the corresponding raw file (*spectra_*.txt*) at the same index position.

5.2 Timestamp Filtering

Filtering ensures data consistency by keeping only timestamps that exist in the log file. The process uses tolerance-based matching (default: 0.0001 seconds) to account for floating-point precision differences. Timestamps are normalized to 6 decimal places before comparison.

5.3 Data Aggregation

For each filtered timestamp, the system aggregates data from multiple sources:

- **Azimuth & Elevation:** Retrieved from log file by matching timestamp
- **Peak Data:** Read from *_Peak.txt* file, starting from column 4 (range-resolved SNR)
- **Spectrum Data:** Read from *_Spectrum.txt* file, starting from column 4 (range-resolved spectrum)
- **Wind Data:** Read from *_Wind.txt* file, starting from column 4 (range-resolved wind velocity)
- **Power Density Spectrum:** Read from raw spectra file (ASCII .txt), skipping first 13 header lines

The aggregated data is organized into a nested dictionary structure, with timestamps as keys and dictionaries containing all parameters as values.

6. Database Storage

The system uses SQLite for persistent storage, enabling efficient querying and long-term data retention. The database schema is normalized with separate tables for different data types.

6.1 Database Schema

The database uses a normalized schema with five tables. The **timestamps** table is the main table storing timestamp metadata. It contains columns for timestamp (primary key), azimuth, elevation, source_processed_dir, source_raw_dir, source_log_file, imported_at, and updated_at. This table serves as the central reference for all timestamp-related data.

Six additional tables store array data, each with a timestamp foreign key and a data column storing JSON-formatted arrays. The **peak_data** table stores range-resolved peak/SNR data. The **spectrum_data** table stores range-resolved spectrum data. The **wind_data** table stores range-resolved wind velocity data. The **power_density_spectrum** table stores raw power density spectrum data. The **snr_profile** table stores computed SNR profiles from Mode 3 processing. The **wind_profile** table stores computed wind profiles from Mode 3 processing. All array data tables use ON DELETE CASCADE to maintain referential integrity with the timestamps table.

6.2 Key Features

- Fast timestamp-based lookups using primary key indexing
- Range queries for time-series analysis
- Automatic database creation and schema initialization
- Support for updating existing entries without duplicates
- Foreign key constraints ensure data integrity
- Array data stored as JSON for efficient storage and retrieval
- Metadata tracking (source files, import/update timestamps)

7. Analysis and Visualization

7.1 Range-Resolved Profile Analysis

The system processes range-resolved profiles, where each parameter (wind, peak, spectrum) is a 1D array representing measurements at different altitudes. The range resolution is defined by:

- **Range Step:** Spacing between range bins (default: 48 m)
- **Starting Range:** Range corresponding to first bin (default: -1400 m)
- **Requested Ranges:** Specific ranges to extract (e.g., [100, 200, 300] m)

7.2 Heatmap Generation

Heatmaps visualize parameter distributions across azimuth and elevation angles at specific ranges. The generation process:

1. Query all data from database for the specified parameter
2. Extract values at the requested range for each timestamp
3. Group data by azimuth and elevation angles
4. Create a 2D grid by binning azimuth/elevation values
5. Compute mean values for each grid cell
6. Generate heatmap visualization with color-coded values
7. Save plot as image file (PNG, PDF, SVG, etc.)
8. Display plot interactively

7.3 Supported Parameters

The system supports three main parameters for visualization. The **wind** parameter provides wind velocity profiles from `_Wind.txt` files, representing range-resolved wind velocity measurements. The **peak** parameter (also known as **SNR**) provides signal-to-noise ratio profiles from `_Peak.txt` files, representing range-resolved SNR measurements. The **spectrum** parameter provides spectral power profiles from `_Spectrum.txt` files, representing range-resolved spectral power measurements. All three parameters are range-resolved profiles, meaning each measurement is associated with a specific altitude or range value.

7.4 Single-Profile Mode (Mode 3)

Mode 3 processes range-resolved power density spectra to compute SNR and wind profiles. The processing pipeline:

1. For each timestamp, load range-resolved power density spectra (one spectrum per range)
2. Compute frequency array from FFT size and sampling rate (frequencies from 0 to fs/2)
3. For each range, find frequency bin with maximum SNR within allowable frequency interval
4. Store maximum SNR value as SNR profile value for that range
5. Compute wind speed using coherent Doppler lidar equation: $v = \text{laser_wavelength} * (\text{dominant_frequency} - \text{frequency_shift}) / 2$ (result in m/s)
6. Store computed wind speed as wind profile value for that range
7. Save SNR and wind profiles to database (only updates profile tables, preserves other data)
8. Generate two visualization plots: one with all SNR profiles, one with all wind profiles

The Doppler lidar equation relates wind speed (v) to dominant frequency (f), frequency shift (f_0), and laser wavelength (λ): $v = \text{laser_wavelength} * (\text{dominant_frequency} - \text{frequency_shift}) / 2$, where `dominant_frequency` is the frequency at maximum SNR, `frequency_shift` is typically 0 Hz, and `laser_wavelength` is typically 1.55 μm. The result is in m/s units.

Output files are organized into subfolders within `visualization_output//`:

- `wind_heatmaps/` - Wind speed heatmaps
- `snr_heatmaps/` - SNR (peak) heatmaps
- `spectrum_heatmaps/` - Spectrum heatmaps
- `single_profile/` - SNR, wind, and dominant frequency profile plots
- `snr_difference/` - SNR difference heatmaps between sequential ranges
- `wind_difference/` - Wind difference heatmaps between sequential ranges
- `fwhm/` - FWHM heatmaps of dominant frequency peaks

Range Notation Support: The `requested_range_indices` parameter supports flexible range notation. Single values can be specified as comma-separated lists (e.g., 1,2,3). Ranges can be specified using dash notation (e.g., 1-50 expands to 1,2,3,...,50). Mixed formats are also supported (e.g., 1,4-10,8 expands to 1,4,5,6,7,8,9,10,8).

8. Configuration

The system uses a simple text-based configuration file (config.txt) with key=value pairs. This allows easy customization without modifying code.

8.1 Key Configuration Parameters

Directory Paths:

- processed_root: Root directory for processed data files
- raw_root: Root directory for raw spectra files
- log_file: Path to log file (output.txt)
- database_path: SQLite database file path
- visualization_output_dir: Output directory for heatmaps

Processing Parameters:

- timestamp_tolerance: Tolerance for timestamp matching (default: 0.0001 s)
- timestamp_precision: Decimal places for normalization (default: 6)
- processed_suffix: Suffix for processed files (default: _Peak.txt)
- raw_file_pattern: Pattern for raw files (default: spectra_*.txt)
- raw_spectra_skip_rows: Header lines to skip (default: 13)

Visualization Parameters:

- range_step: Spacing between range bins (default: 48.0 m)
- starting_range_index: Index of range bin corresponding to 0 m (default: 0)
- requested_range_indices: Range notation supported (e.g., 1-50, 1,4-10,8)
- heatmap_colormap: Matplotlib colormap name (default: viridis)
- heatmap_format: Image format (default: png)

Execution Mode:

- run_mode: Main script mode (test, heatmaps, or profiles)
- heatmap_parameters: Comma-separated parameters (e.g., wind,snr)

Single-Profile Mode Parameters:

- profile_fft_size: FFT size for frequency computation (default: 128)
- profile_sampling_rate: Sampling rate in Hz (default: 100000.0)
- profile_frequency_interval: Allowable frequency range "min,max" in Hz (default: "0,50000")
- profile_frequency_shift: Frequency shift for Doppler equation in Hz (default: 0.0)
- profile_laser_wavelength: Laser wavelength in meters (default: 1.55e-6)

Debug Mode:

- debug_mode: Enable verbose debug output (true/false, default: false)
- When true: prints detailed database content, matching details, array structures
- When false: prints only essential information (normal mode)

9. Usage Examples

9.1 Running Tests

To run the complete test suite (includes database creation):

```
python main.py --test
```

9.2 Generating Heatmaps

Generate heatmaps for wind and SNR at specific ranges:

```
python main.py --heatmaps --parameters wind snr --ranges 100 200 300
```

Use default parameters from config.txt:

```
python main.py --heatmaps
```

9.3 Generating Single Profiles

Generate single-profile visualizations (Mode 3):

```
python main.py --profiles
```

This mode processes range-resolved power density spectra, computes SNR and wind profiles using Doppler lidar equations, stores profiles in database, and generates visualization plots.

9.4 Programmatic Usage

Example Python code for matching and filtering:

```
from data_reader import match_processed_and_raw, filter_matches_by_log_timestamps
matches = match_processed_and_raw(processed_root, raw_root) filtered =
filter_matches_by_log_timestamps(matches, log_file) print(f'Filtered matches:
{len(filtered)}')
```

Example for generating heatmaps:

```
from data_reader import create_heatmaps results = create_heatmaps(
db_path='data/cuav_data.db', range_step=48.0, starting_range=-1400.0,
requested_ranges=[100, 200, 300], parameters=['wind', 'peak'],
output_dir='visualization_output', colormap='viridis', save_format='png' )
```

Example for single-profile processing (Mode 3):

```
from data_reader.analysis.visualization import ( process_single_profiles,
create_profile_visualizations ) # Process profiles stats = process_single_profiles(
db_path='data/cuav_data.db', range_step=48.0, starting_range=-1400.0, fft_size=128,
sampling_rate=100000.0, frequency_interval=(0, 50000), frequency_shift=0.0,
laser_wavelength=1.55e-6 ) # Generate visualizations results =
create_profile_visualizations( db_path='data/cuav_data.db', range_step=48.0,
starting_range=-1400.0, output_dir='visualization_output/logfile_name',
save_format='png' )
```

Summary

The CUAV Field Tests Data Reader provides a comprehensive solution for processing, storing, and analyzing atmospheric measurement data from lidar field tests. The system automates the complex task of aligning data from multiple sources, provides efficient persistent storage through SQLite, enables sophisticated spatial analysis through range-resolved heatmaps, and supports advanced Doppler lidar profile processing.

Key advantages of this system include:

- Automated data alignment across multiple file types and directory structures
- Robust timestamp matching with tolerance-based filtering
- Efficient database storage enabling fast queries and long-term data retention
- Flexible visualization capabilities for spatial analysis
- Single-profile mode for SNR and wind profile computation using Doppler lidar equations
- Simple configuration management through text-based config files
- Modular architecture enabling easy extension and maintenance