Chico Measurement Campaign
Data Organization & Real-Time Processing

Pierre Dérian, Chris Mauzey

as of November 19, 2014

This document aims at describing the organization of the REAL data and the motion estimates. The idea is to use an *MySQL* database to store information on and link scans, estimates and sequences altogether. A *Python* framework supervises the data flow, motion estimation, and provides tools for visualizations. Services are setup using *Upstart* to create an autonomous, integrated system that handles data reception, motion estimation and visualizations in real-time.

Contents

1 General concepts 3
1.1 Data flow . . . 3
1.2 Linking pieces together . . . 3

2 Description of MySQL tables 4
2.1 Locations . . . 4
2.2 Bscans . . . 4
2.3 Motions . . . 5
2.4 Sequences . . . 5
2.4.1 Cross-correlations . . . 6
2.4.2 Optical Flow . . . 6
2.5 House-Keeping . . . 7

3 Python framework 8
3.1 Incoming data: Bscans and HKs . . . 8
3.2 Motion estimators . . . 8
3.3 Visualizations . . . 8
3.3.1 Scans . . . 9
3.3.2 Motions . . . 10
3.3.3 Time-series . . . 10
3.3.4 Image loopers . . . 11
3.4 Structure of the Python toolbox . . . 11
3.4.1 CrossCorrEstim, BatchProcessing . . . 11
3.4.2 lidarIO . . . 11
3.4.3 lidarRunTime . . . 11
3.4.4 lidarAnalysis . . . 11
3.4.5 PyCudaTools . . . 11
3.4.6 Papers . . . 12
3.5 Required non-standard Python modules . . . 12

4 Upstart, towards an autonomous system 13

List of Figures

1 Detailed dataflow on **aeolus** . . . 14
2 MySQL tables connections . . . 15
3 Python file monitors . . . 15

1 General concepts

1.1 Data flow

Here is the basic flow of a scan file and the corresponding motion estimates:

(i) scan data is collected by **real2** and stored as a single bscan file.

(ii) this bscan is then transferred to **aeolus**.

(iii) there, motion estimation will be performed by both CC (cross-correlations) and OF (optic flow) *if a previous scan is available.*

(iv) results (NetCDF files) are finally written on **aeolus** file system.

(v) (optional) visualizations of scan and motion are created and made available, on **physics** web server.

Figure 1 gives a more detailed overview of the situation.

Remark 1. real2 must be able to transfer data to **aeolus**. Dedicated user account with limited permissions?

1.2 Linking pieces together

As the experiment lasts over several months, it is crucial to keep track of the recorded and estimated data in a simple, efficient and flexible way. First, let us roughly define four entities:

- each lidar *location* has its individual properties (latitude, longitude, etc.).
- each *scan* has its individual properties (type, location, date+time, duration, angles, etc.).
- each *motion* estimate, whether CC or OF, corresponds to a pair of scan.
- a *sequence* is a collection of motion estimates sharing identical estimation parameters (block size, regularization, ...).

Therefore, we propose to use an MySQL database of 7 tables to link scans, geographic location, motion estimates and their parameters (sequence) together:

- one table which contains the geographic information of the scanning origin (i.e. the location of the REAL);

- one table which contains the list of all scans;
- two tables, one for CC and one for OF which lists all estimates (of a given method) and links them to their respective scan pair and sequence;
- two tables, one for CC and one for OF which lists all sequences and the associated set of parameters.
- one table for the house-keeping files.

These tables and their connections are shown in Fig. 2.

2 Description of MySQL tables

[TODO] The whole database can be generated from the SQL file `real_lidar_db.sql` located in `MySQL/` directory of the SVN. The structure of each table is listed below, as: **column_name**; brief description; (`data type`).

Remark 2. Backscatter profiles are timed with sub-second precision. However, *the precision of the date/time objects used by MySQL is limited to the second.* This date/time information (in the database) is precise enough to sort scans, or to select the scans over a given time range. *It should not be used for precise timing*; instead the complete date/time information should be read directly from the bscan.

2.1 Locations

id unique ID of the scanning origin's location (`int`);

latitude latitdude of the scanning origin (`double`);

longitude longitude of the scanning origin (`double`);

altitude altitude (meters above sea level) of the scanning origin (`double`);

description a description of the REAL's location (`text`);

2.2 Bscans

id unique ID of the scan (`int`);

path path to the bscan file (`text`);

location_id ID of the scanning origin's location (`int`);

type scan type (PPI, RHI or STARE) (`enum`);

start_time date+time of the first record (`datetime`);

end_time date+time of the last record (`datetime`);

start_azimuth azimuth of the first record (degrees) (`float`);

end_azimuth azimuth of the last record (degrees) (`float`);

start_elevation elevation of the first record (degrees) (`float`);

end_elevation elevation of the last record (degrees) (`float`);

num_records number of records (`int`);

num_ranges number of range samples per record (`int`);

range_zero first range value (meters) (`float`);

range_step distance between samples (meters) (`float`);

2.3 Motions

The two tables (CC, OF) are identical:

id unique ID of the estimate (`int`);

path path to the NetCDF file (`text`);

bscan0_id id of scan at time t0 (`int`);

bscan1_id id of scan at time t1 (`int`);

sequence_id id of the sequence to which this scan belongs (`int`);

start_time date+time of the first record in scan t0 (`datetime`);

end_time date+time of the last record in scan t1 (`datetime`);

2.4 Sequences

Each estimator has its own set of parameters, so tables differ.

2.4.1 Cross-correlations

id the unique id of the sequence (`int`);

grid_cell_width length of sides of each square cell in the profile grid (meters) (`float`);

interrogation_block_width length of sides of each square tile extracted for cross correlation (meters) (`float`);

vector_spacing distance between vectors spaced out in a uniform rectangular grid (meters) (`float`);

max_scan_radius maximum distance from the lidar to the edge of the scan (meters) (`float`);

2.4.2 Optical Flow

id the unique id of the sequence (`int`);

num_moment number of vanishing moment of the wavelet (`int`);

num_decomposed number of scales considered (`int`);

num_estimated number of scales estimated (`int`);

num_initialized number of scales initialized, with the pyramid (`int`);

smooth_kernel sigma value for the gaussian smoothing kernel [pixel] (`float`);

num_pyramid_step number of pyramid steps (`int`);

scale_X scaling factor along X in the pyramid [%] (`float`);

scale_Y scaling factor along Y in the pyramid [%] (`float`);

regularization regularization scheme (`int`);

alpha_regularization regularization parameter alpha (`float`);

robust_function robust M-estimator (`int`);

sigma_robust robust parameter sigma (`float`);

dim_X dimension of region of interest (ROI) along X [pixel] (`int`);

dim_Y dimension of ROI along Y [pixel] (`int`);

offset_X horizontal offset of ROI from top-left border [pixel] (`int`);

offset_Y vertical offset of ROI from top-left border [pixel] (`int`);

cut_values cut values before estimation (`tinyint`);

val_min minimum value [signal unit] (`float`);

val_max maximum value [signal unit] (`float`);

histogram correct histograms before estimation (`int`);

normalize_values normalize values to [0;1] before estimation (`tinyint`);

2.5 House-Keeping

id ID of house keeping entry (`int`);

datestring Date and time of house keeping entry (`datetime`);

1_064_energy 1.064 energy (`float`);

1_544_energy 1.544 energy (`float`);

eff_percent EFF % (`float`);

shots Number of shots (`int`);

cell_temp Raman cell temperature (`float`);

cell_pres Raman cell pressure (`float`);

lsr_temp Laser temperature (`float`);

tbl_temp Table temperature (`float`);

rm_temp Room temperature (`float`);

fl_shots Flashlamp shots (`int`);

fl_volts Flashlamp voltage (`float`);

long_ang Long ang (`double`);

trans_ang Trans ang (`double`);

ups_power UPS power (`double`);

centx_loc CentX loc (`double`);

centy_loc CentY loc (`double`);

radius Radius (`double`);

process_length Process Length (`double`);

location_id ID of current location of REAL (`int`);

3 Python framework

3.1 Incoming data: Bscans and HKs

Incoming data consists of bscans and house-keeping files (HKs). These files are regularly transferred to **aeolus** by SFTP to a temporary directory `REAL_sftp`. Two python scripts (one for bscans, another one for HKs) monitor this directory. Their role is to record the new files into the appropriate tables of the database, and to move them from `REAL_sftp` to their destination in `REAL_bscans` or `REAL_house_keeping`. Figure 3 illustrates this process.

3.2 Motion estimators

3.3 Visualizations

The Python Toolbox provides a few high-level visualization tools wich can be used for both post-processing and real-time. Note that these tools cannot possibly handle every situation; for the most specific cases custom versions should be developped from the low-level modules – See Sect. 3.4.

Remark 3. For interactive, on-screen display to be available with `--show`, a valid `$DISPLAY` must be set – either a monitor or X-redirection through SSH. Otherwise, image-file export only is allowed. Note that using interactive display through network (SSH) can be fairly slow if the bandwith does not keep up and/or as the complexity of the plot increases.

Remark 4. In the following, date-time arguments are specified in UTC,[1] following the format "year-month-day hour:min:sec", e.g. `"2013-03-02 01:18:22"`.

Remark 5. When file transfer to distant directory is available, SSH authentication based on public-private keys[2] must be available for the given user. Password-based authentication is NOT possible.

[1] universal time coordinates - http://en.wikipedia.org/wiki/Coordinated_Universal_Time.
[2] See http://en.wikipedia.org/wiki/Secure_Shell#Definition.

3.3.1 Scans

Scans can be visualized, either on-screen (interactive, see Rem. 3) or by PNG images (static) using *viewScan.py* script. The main modes are:

- Single scan, with database: picks the first scan after given date (see Rem. 4), `-tmin DATETIME`.
- Sequence of scan, with database: scans comprised between two dates, `-tmin DATETIME_MIN -tmax DATETIME_MAX`.
- Real-time visualization, with database, `--standby`.
- Single scan, from bscan file, `-s PATH/TO/BSCAN`.

Scans of a given type only (PPI, RHI) can be selected with `-t SCANTYPE`. The local directory where images will be written is given with `-o PATH`. If omitted, no output. A *distant* directory can be specified with `-do LOGIN@HOST:DISTANT_PATH` – see Rem. 5. Various options modify scan aspect (e.g., domain, units, grid, colors, range of data). In particular, `--no-filter` disables median filtering. The resolution of output images is controled by `--style STYLE`. By default, a web-compatible image is generated. Other choices include '720p' for HD-ready, '1080p' for full-HD, 'print' for print-compatible resolution.

Typical work run: basic on-screen display of a single PPI scan at a given time, no file output.

```
python viewScan.py -t PPI --show -tmin "2013-10-23 23:30:31"
```

Typical post-processing run: a 1-hour, full-HD sequence of unfiltered RHIs, with constrainted domain, units in km and custom data range.

```
python viewScan.py -t RHI --no-filter --km --clim 33.0 45.0 \
    -xmin 0 -ymin 0 -xmax 5000 -ymax 2500  \
    -o /tmp --style 1080p \
    -tmin "2009-09-10 14:30:00" -tmax "2009-09-10 15:30:00"
```

Typical real-time run: PPIs only, with (image) SNR, local time, grid and rings, and transfer to remote server.

```
python viewScan.py --standby -t PPI --grid --ring --ptz \
    --snr --imgsnr -o /tmp -do LOGIN@physics.csuchico.edu:/tmp
```

3.3.2 Motions

3.3.3 Time-series

Time-series of wind speed and direction and their rolling means can be plotted using *liveSeries.py* script. The script can handle up to three series simultaneously among OF, CC and Doppler (when available). It works both for real-time analysis or post-processing. Note that on-screen display is not available at the moment, only PNG images output. The main parameters are:

- `--OFseqID ID`, `--CCseqID ID` the ID number, in the database, of the motion sequence for optic flow and/or cross-correlations, respectively.
- `-x0 X`, `-y0 Y` the coordinates (X, Y) for where OF, CC estimates are probed, in [meter]. Make sure these coordinates belong to the scan domain! By default, they point to the Doppler location $(382, 1475)$.
- `--doppler` enable display of Doppler data, if available.
- `-heigh HEIGHT` the height in [meter] of Doppler data.
- `--start DATETIME` the beginning of the time-series. *If omitted, starts at current time and enables real-time display.*
- `--span SPAN` the length of the time-series, in hours.
- `-localOut LOCAL_FILE.PNG`, `-distantOut DISTANT_DIR` the local file and optionally distant file (see Rem. 5).
- `--coherenceCheck` discards outliers based on temporal coherence. Very efficient with isolated outliers.
- `--style "stat"` makes a scatter plot of *instantaneous* speed and direction values instead of time-series. Note that it requires 2 series.

Other options include the length of time-averaging window, the radius of space averaging area, other plotting styles, etc.

Typical post-processing run: a 10-hour period of OF and Doppler, in UTC, with outliers removal:

```
python liveSeries.py --OFseqID 59 --doppler --coherenceCheck \
    --start "2013-09-17 16:00:00" --span 10 \
    -localOut /tmp/series_20130917-160000_OF-Doppler.png
```

Typical real-time run: display OF and CC estimates at $(x, y) = (1000, 1000)$ m for the last 12 hours, in local time (PTZ), transfer to remote server. In this mode, the script runs continuously and checks periodically for new data points. If any, the image is updated and saved. The script has to be stopped manually (CTRL-C, `kill`).

```
python liveSeries.py --OFseqID 10 --CCseqID 21 \
    -x0 1000 -y0 1000 --span 12 --ptz \
    -localOut /tmp/liveseries_OF-CC.png \
    -distantOut LOGIN@physics.csuchico.edu:/tmp
```

3.3.4 Image loopers

3.4 Structure of the Python toolbox

3.4.1 CrossCorrEstim, BatchProcessing

3.4.2 lidarIO

3.4.3 lidarRunTime

It also contains scripts used on aeolus for input data management during real-time operations (Sec. 3.1):

bscan_monitor.py handles bscan files from **real2**;

hk_monitor.py handles house-keeping files from **real2**;

csm_monitor.py handles CSM files from the Streamlines doppler lidar.

It also contains scripts used for visualizations, including during real-time operations:

viewScan.py visualize scans (Sec. 3.3.1);

viewMotion.py visualize wind fields (Sec. 3.3.2);

liveSeries.py time-series of wind speed and direction (Sec. 3.3.3);

pngMonitor.py scan visualizations for PPI, RHI loopers on **physics** (Sec. 3.3.4);

updateWebVisDB.py updates the CC data on **physics** for the Wind Visualization webapp;

updateOFWebVisDB.py updates the OF data on **physics** for the Wind Visualization webapp.

3.4.4 lidarAnalysis

3.4.5 PyCudaTools

This directory contains CUDA kernels and PyCuda interface modules:

CuMedianFilter low-pass and high-pass median filtering for scan pre-processing;

CuPolarToRect polar to cartesian interpolation on *regular grids*;

CuScatteredInterp nearest-neighbor interpolation for *scattered data*;

CuScanSNR image SNR computation.

Test/demo scripts are provided as well.

3.4.6 Papers

This directory contains scripts created (or adapted) specifically for a given paper/poster: AMS BLT 2014, JTECH on wavelets, cross-correlations, Chico experiment, ...

3.5 Required non-standard Python modules

The modules listed below are not part of the Python Standard Library, they must be installed in order to use the framework detailed above. Most of these modules are shipped with Python distributions such as ActivePython or Anaconda. Others can be installed by `pip`, or manually.

Matplotlib provides Matlab-like plotting and visualization capabilities.
http://matplotlib.org/

Mysql.connector for connections to MySQL database.
http://dev.mysql.com/downloads/connector/python/

NetCDF4 enables to handle NetCDF files.
https://code.google.com/p/netcdf4-python/

Numpy is the fundamental package for scientific computing.
http://www.numpy.org/

PyCuda is a CUDA wrapper for Python.
http://mathema.tician.de/software/pycuda/

PyEphem to compute sunset, sunrise times.
http://rhodesmill.org/pyephem/

Pytz to deal with time zones conversions.
http://pytz.sourceforge.net/

Scipy is a scientific library for advanced computations (e.g. filtering).
http://www.scipy.org/

4 Upstart, towards an autonomous system

Ideally, the *acquisition-processing-visualization system* deployed for real-time operations (currently on **aeolus**) must be as autonomous as possible. In particular, it should handle the following situations *automatically*, without human intervention:

- be available as the system starts (typically after a reboot);
- maintain the critical mission;
- recover from crashes.

The *critical mission* is the reception, registration in database and direction of incoming bscans (and optionally, HKs), since the rest of the framework relies on the information provided by the database (and the expected location of files). These tasks are supervised by *monitor scripts* (Sec. 3.1), they require the *network* and *mysql* services to be running. For now, the other tasks (motion estimation, all kinds of visualizations) are secondary. As long as the critical mission is maintained, they can be stopped and resumed anytime.

In the current deployment, points listed above are handled by creating services (a.k.a. jobs) using Upstart:[3]

> "Upstart [. . .] handles starting of tasks and services during boot, stopping them during shutdown and supervising them while the system is running."

Upstart relies on *Job Configuration Files* that describes, for each job, things such as what to do and under what conditions. Typically, in our context, these elements are:

- start when mysql, network are running;
- respwan (restart) after a crash (up to a maximum number of times);
- set-up the environment (output directories, logs, . . .);

[3] http://upstart.ubuntu.com/

- finally run the Python script with adequate parameters.

Examples of configuration files can be found in the `Upstart/` directory of the SVN repository for the Python scripts used during real-times operations. These files are to be placed in the `$HOME/.init` directory of the user in charge of running the services.

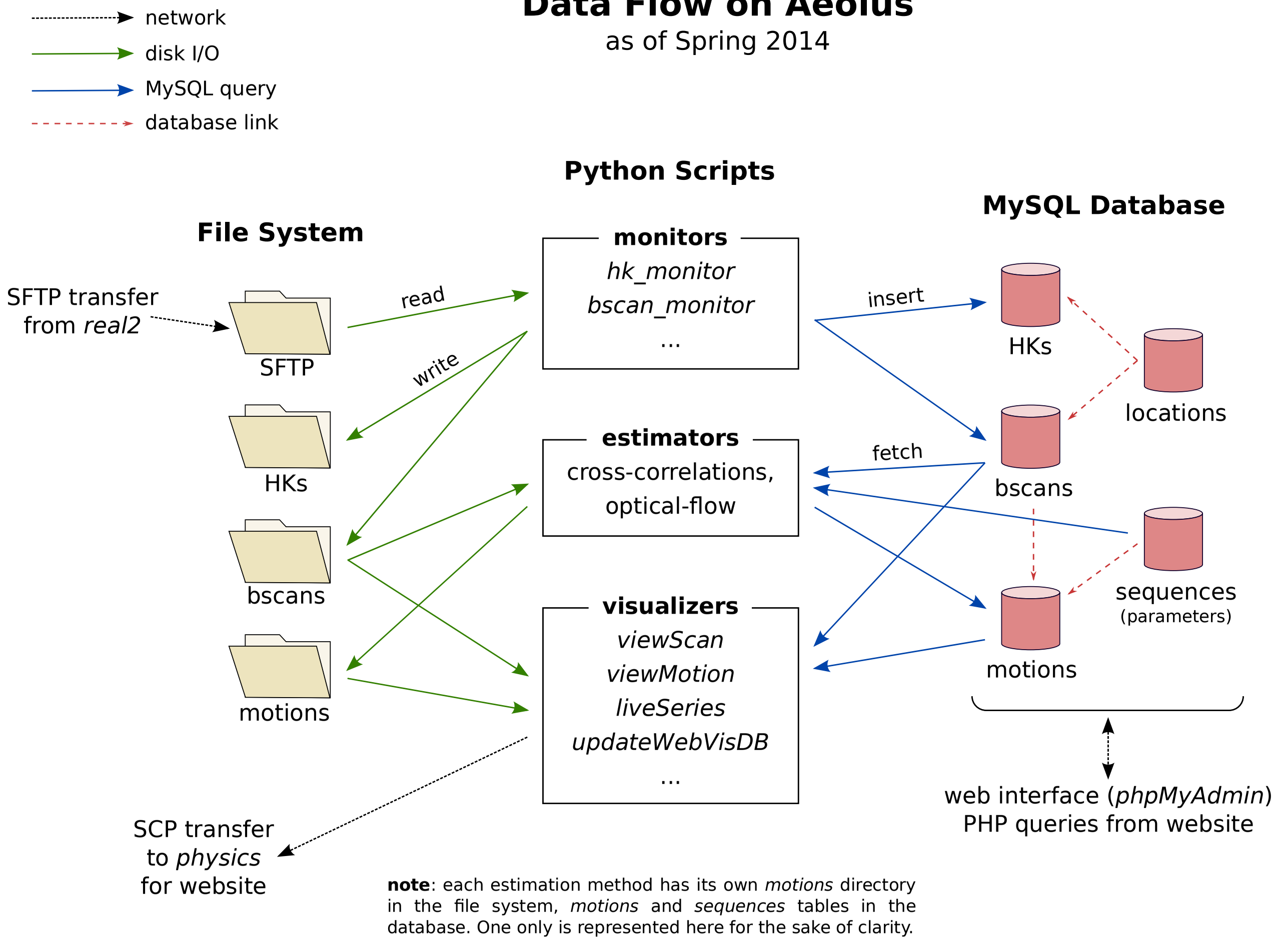

Figure 1: Dataflow on **aeolus**, showing the 3 components: file system, Python framework, MySQL database.

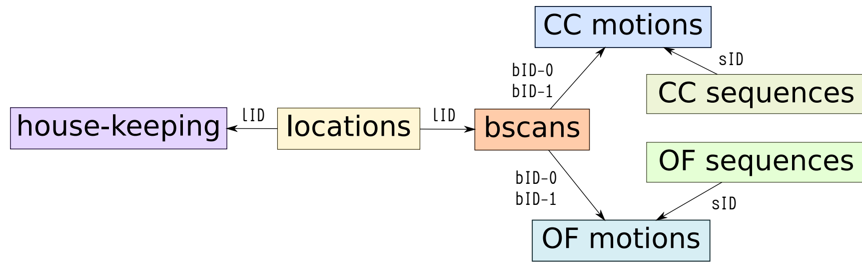

Figure 2: Schema of the 7 SQL tables. `bID` stands for bscan-ID, `LID` for location-ID, and `sID` for sequence-ID.

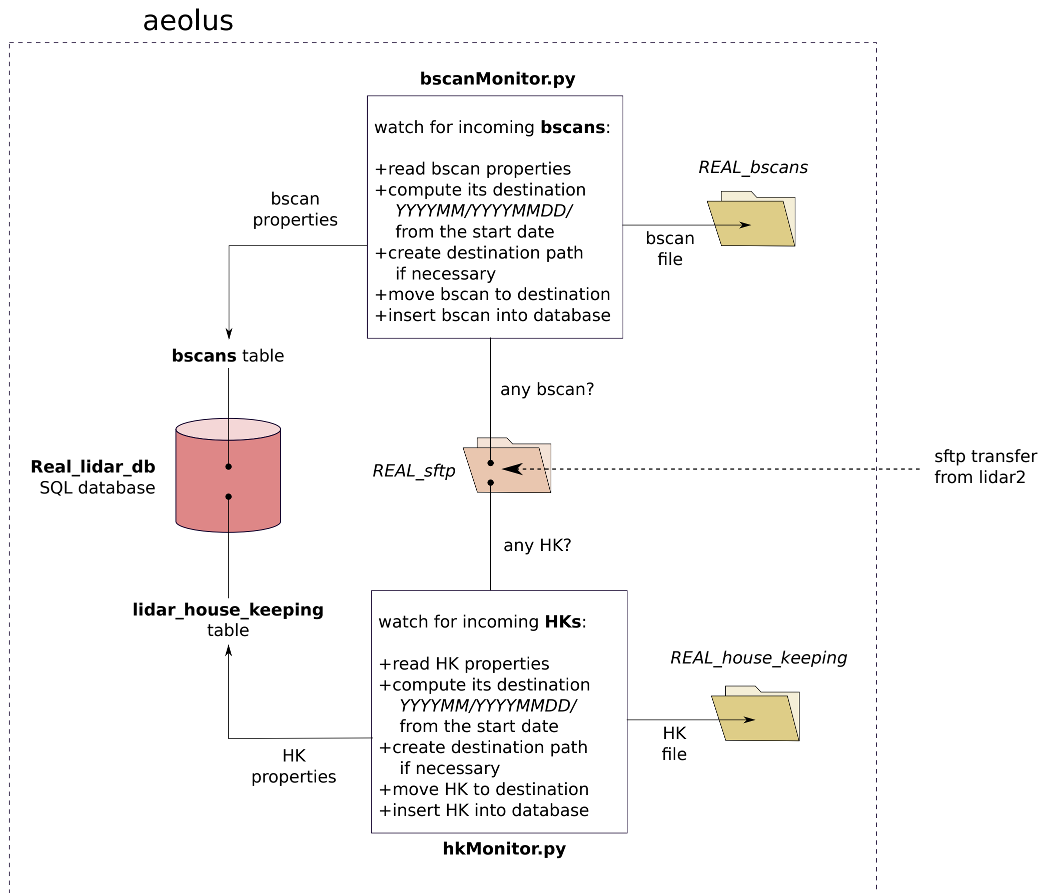

Figure 3: Schema of the input data process, showing the two monitoring scripts watching directory `REAL_sftp` and processing incoming bscan and HK files.