Semi-Automated Cloud Imager System - Software Guide

John Mrziglod (john.mrziglod@mail.de)

November 2017

1 Introduction

The purpose of the Semi-Automated Cloud Imager System (SACIS) is to measure the cloud coverage and further cloud statistics along a ship track over the sea. It consists of three (optional four) components:

- **Pinocchio:** A former surveillance dual-camera system made by MOBOTIX consisting of a visible and an infrared camera. While looking at the zenith, it creates images of the sky and clouds.
- **Ceilometer:** An instrument mady by Jenoptik measuring the cloud base height by using a laser system at near-infrared wavelength.
- **DShip:** Additional atmospheric data (such as air temperature, pressure, etc.) is so far coming from a weather station of the research vessel where SACIS is deployed.
- **Dumbo (optional):** A dual-camera system similar to Pinocchio but with a more advanced infrared camera. Since it needs more maintenance than Pinocchio and cannot be driven full-automatically, it is only used as calibration reference for Pinocchio.

The CLOUD toolbox (`https://github.com/JohnMrziglod/cloud`) contains python scripts and modules to process and analyse the data coming from SACIS. This document explains how to install, to use and to develop this toolbox. If you just want to use the CLOUD toolbox without dealing with the underlying python code, you should read the sections 2, 3 and 4. If you want to develop it further, the section 7 gives more information about the internal structure to you.

2 Setup the Cloud toolbox

2.1 Install dependencies

CLOUD requires python 3.6 or higher and the following packages:

1. numpy
2. scipy
3. pandas
4. matplotlib >= 2.0
5. netCDF4
6. Pillow >= 2.0
7. typhon

To install the first six packages type this into your console:[1]

```
$ pip3 install --user numpy scipy pandas matplotlib netcdf4 Pillow
```

typhon is a python package developed by the Radiation and Remote Sensing group of the Meteorological Institute of the University Hamburg. Since the required features for Cloud are only in the developer version at the moment, it has to be installed differently. Go to your favourite directory (wherever you want to put the typhon module) and type those lines into your console:

```
$ git clone https://github.com/atmtools/typhon.git
$ cd typhon
$ pip3 install --user --editable .
```

2.2 Get the Cloud toolbox

You can get the Cloud toolbox via this command (run it where you want to have Cloud toolbox):

```
$ git clone https://github.com/JohnMrziglod/cloud.git
```

Just type `$ cd cloud` and you are in the toolbox folder. And it is done! Send an e-mail to john.mrziglod@mail.de if you run into problems.

3 Scripts & User Files

3.1 Scripts

The Cloud toolbox contains scripts that are designed to be used from the command line without requiring special knowledge about Python:

Table 1: Scripts of the Cloud toolbox.

Script	Description
`processor.py`	• Converts camera files from raw files to netCDF format and applies calibration. • Masking of images. • Calculates cloud statistics of camera images.
`monitor.py`	• Produces overview and comparison plots for all components.
`pinocchio_calibration.py`	Generates a calibration file from a performed Pinocchio calibration (see section 6). Note: this script is deprecated and should be rewritten before use

The scripts' behaviour can be adjusted via command line options or the configuration file. Each script has its command line documentation, simply run `$ script_name.py -h`.

3.2 Configuration file

To avoid long command line arguments there is a file (per default called `config.ini`) that contains the paths to all datasets, mask & calibration files and further configurations. All scripts load the configurations from this file automatically. If you want to use a configuration file with a different name, you can call the script with `--config filename.ini`.

The configuration file syntax is similar to Windows' INI format. See table 2 for a description of all configuration sections. A detailed documentation of the configuration keys can be found in the example file in the Cloud toolbox.

[1] The $ sign does only indicate the start of a command line - do not enter it into your console.

Table 2: Sections of the configuration file.

Section	Description
General	Keys for number of common basedir, parallel processes, etc.
Dumbo	Paths to the raw, netCDF, cloud parameters and mask files of Dumbo.
Pinocchio	Paths to the raw, netCDF, cloud parameters, calibration and mask files of Pinocchio.
Ceilometer	Path to the netCDF files of the Ceilometer.
DShip	Path to the ASCII files of the DShip metadata.
Plots	Output paths of the generated plots, window size of average, etc.

You can define via paths which files should be processed by the scripts and where they should be created. The paths can contain placeholders (e.g. {`year`}, {`month`}, etc.) that are used to retrieve temporal information from the path names.

This manual and in the command line documentation of the scripts will refer to the configuration keys in the file via `[Section][key]`.

3.3 Calibration file

Pinocchio instruments need calibration files to convert the pixel brightness into a temperature. Those calibration files are in CSV format and must contain two columns: one with pixel brightnesses (between 0-255) and one with their corresponding temperatures (in degree Celsius). You can find current calibration files in the directory `calibration`. You can create your own calibration files after performing a calibration with `pinocchio_calibration.py` (see section 6). Set the path to the calibration file that you want to use in the config file via the key `[Pinocchio][calibration]`.

3.4 Mask file

When installing the cameras on a ship or close to buildings, parts of the image should be masked to avoid incorrect cloud statistics (e.g. if a pole reaches into the camera's viewing area). Simply create a matching mask by using the templates in the `masks` directory (e.g. with gimp) and set the path to the mask file via configuration key `[instrument][mask]`.
It is important that the mask matches the size of the raw images (Dumbo: 384x288, Pinocchio: 336x252). The mask should be a gray scale PNG image. White pixels are interpreted as transparent, black pixels as opaque.

3.5 Logbook file

If one works on the instruments during measuring, some images might be falsified. To exclude those images from processing, you can write a logbook file. This should be a ASCII file with at least two columns separated by tabulators. The first column contains the starting time of the falsification of the measurements in UTC, the second column contains the ending time. All images that are measured between those two timestamps will be excluded from processing. You can specify one logbook file for each camera system (Dumbo and Pinocchio). Set the path to the logbook file via the configuration key `[instrument][logbook]`.

4 Usage & Workflow

Before running any scripts you should check whether all configuration keys in `config.ini` are correctly set and whether the paths point to existing files (unless they are for files that are going to be created). Make sure that the scripts that you want to run have executive permissions.[2] Afterwards you can follow this workflow:

[2] If not, you may run `$ chmod 744 script_name.py` first.

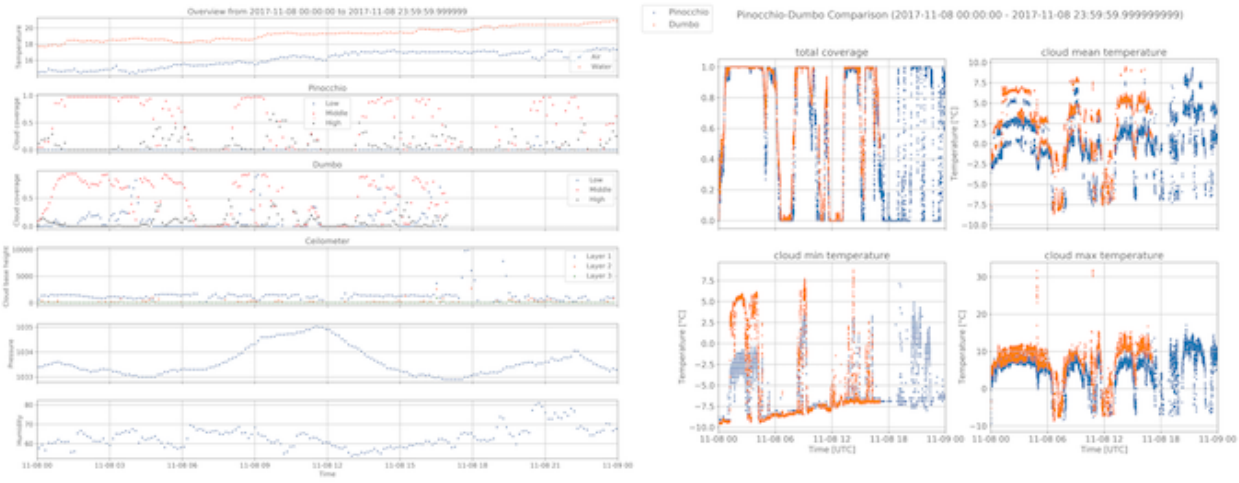

Figure 1: Overview and comparison plot.

1. **Copy the new data to their locations:** After recording, copy the data from the cameras and other instruments to the locations that you have set in the configuration file. Pinocchio images should be saved in tarball archives as JPG files, Dumbo images as ASCII files in unzipped folders. The Ceilometer data should be in netCDF format and the DShip data in CSV files.

2. **Process the images:** Convert the raw files into netCDF files (which are easier to handle) and calculate cloud parameters. Those cloud parameters are going to be saved in additional netCDF files. You can use the `processor.py` script for this. When having Pinocchio files, it also can extract them from their daily tarball archives. Here are the most common calls of `processor.py`:

   - Extract and convert Pinocchio images and then calculate their cloud parameters for one day:
     `$ ./processor.py -xcs "2017-11-03" "2017-11-04"`
   - Same as above but with a shorter time period:
     `$ ./processor.py -xcs "2017-11-03 12:00:00" "2017-11-03 16:00:00"`
   - Dumbo images can be processed by using the `instrument` (`-i`) option. Note that the `extract` (`-x`) option is unnecessary for Dumbo files and can be left out.
     `$ ./processor.py -csi Dumbo "2017-11-03" "2017-11-04"`
   - If you want to calculate cloud parameters from already existing netCDF files of Pinocchio:
     `$ ./processor.py -s "2017-11-03" "2017-11-04"`

3. **Create plots:** You can plot an overview of all instruments with `monitor.py`. It can also create comparison plots between Dumbo and Pinocchio (see figure 1 for an example).The most common calls of `monitor.py` are:

   - Plot an overview of all instruments from 2017-11-02 to 2017-11-10:
     `$ ./monitor.py -o "2017-11-03" "2017-11-10"`
   - Same as above but instead of creating one plot for the full time period, we create one plot for every three hours. For this, we are using the `frequency` (`-f`) option.
     `$ ./monitor.py -of 3H "2017-11-03" "2017-11-10"`
   - Comparison plots of Pinocchio and Dumbo can be created with this command:
     `$ ./monitor.py -c "2017-11-03" "2017-11-04"`

5 Cloud parameters

The main goal of SACIS is to calculate parameters from cloud images and retrieve further climate statistics. With `processor.py` one can calculate basic cloud parameters which are saved in the

statistics files (where they should be saved can be set via `[instrument][stats]`). All cloud parameters are available for three different height levels:

- Level 1 - Low clouds: All clouds with a base height up to 2 km.
- Level 2 - Middle high clouds: All clouds with a base height from to 2 to 6 km.
- Level 3 - High clouds: All clouds with a base height higher than 6 km.

The cloud base height is retrieved by using a temperature gradient. At the moment, the temperature gradient is simply a lapse rate which provides a linear temperature descent (no inversions included). You can set the lapse rate via the config key `[General][lapse_rate]`.

5.1 cloud_coverage

The cloud coverage CC is the ratio between the number of cloud pixels and the total number of all unmasked pixel of the image:

$$CC = \frac{n_{\text{cloud}}}{n_{\text{all}}} \qquad [0-1] \tag{1}$$

5.2 cloud_inhomogeneity

The cloud inhomogeneity CI is a number to quantity the jaggedness of the clouds. It is defined by the perimeter of the clouds divided by their area:

$$CI = \frac{p_{\text{cloud}}}{A_{\text{cloud}}} \qquad [0-\infty] \tag{2}$$

Please note that this parameter is very experimental and may be influenced by image size and the used mask.

5.3 cloud_mean_temperature

The cloud mean temperature CT_{mean} is the mean temperature of all cloud pixels.

5.4 cloud_max_temperature

The cloud max temperature CT_{max} is the maximum temperature of all cloud pixels.

5.5 cloud_min_temperature

The cloud min temperature CT_{min} is the minimum temperature of all cloud pixels.

6 Calibration of Pinocchio

FIXME: Insert instructions how to do the calibration of a Pinocchio camera and how to generate a calibration file.

7 Development of Cloud

7.1 Developer documentation

There is a HTML documentation of all used modules and classes which can be generated by Sphinx. Call these commands in the directory of your Cloud toolbox:

```
$ cd docs
$ make html
```

7.2 Code style and design

I tried following the Coding Style Guide by Google which is based on modern Python coding practices. The code is object oriented to make its structure better maintainable and extendable. If you are not familiar with object oriented programming, here is a good youtube video as an introduction.

The files for each instrument are represented by `typhon.spareice.Dataset` objects (link to further documentation and tutorial). I developed those objects to make handling and processing of datasets with many files easier and faster. `typhon.spareice.Dataset` is still under development and may change its API and name in future. When those major changes come, I will contact you so that the Cloud toolbox can be developed and still benefit from typhon updates.