MASC Evaluation Software V0.1 User Guide

Ralf Bennartz

20 April 2015

1. Overview 2

1.1. General strategy 2

1.2. Disclaimer: Pointers and objects 2

2. Integration of new MASC observations 3

3. Data analysis 4

3.1. MASCOBS\_CONTAINER 4

3.1.1. Data evaluation and access 4

3.1.2. Data subsetting, merging, and saving 5

3.1.3. Time-related container functions 5

3.2. Visualization 6

4. MASCOBS 6

5. FLAKEROI 6

6. Using this 8

7. Writing your own evaluation routines 8

# Overview

This document provides a brief overview on the MASC object-oriented evaluation software.

**Do NOT modify any of the key components of the software stored in directories FLAKEROI or MASC3DGEO**

## General strategy

The software distinguishes between two phases:

* Integration of new raw MASC observations: This will only have to be done once but requires a few additional steps to ensure that snowflakes are matched correctly. The matching is done stereoscopically and it needs to be ensure the imaging parameters for the cameras are represented correctly in the software. This will typically have to be done once for each experimental campaign.
* Data evaluation: Once the data is integrated all evaluation can be performed on the resulting objects, which hold all necessary information for all snowflakes detected in an experiment. Data between different experiments can be combined or sub selected at any stage.

## Disclaimer: Pointers and objects

The software is written using objects and pointers. This means all potential pitfalls associated with pointers apply. In particular, the following sequence will lead to undefined results:

IDL> a=PTR\_NEW(‘asd’) ; creates a pointer

IDL> PRINT, \*a ; print whatever the pointer points at

asd

IDL> b=a

IDL> PRINT, \*b ; print whatever the pointer points at

asd

IDL> PTR\_FREE,a ; free pointer a

IDL> PRINT,\*b ; leads to b being undefined

% Invalid pointer: B.

% Execution halted at: $MAIN$

The same happens with objects. Make sure you understand the implications. If you overwrite a previously defined object, you delete all content. Any other objects pointing to that same content will afterwards give undefined results.

# Integration of new MASC observations

Data integration is tricky and not well documented at this point. The following three points are important when you try to integrate new MASC observations:

* The software is only as good as the description and quality of the MASC images are. The MASC is assumed to be in focus at the center of the decagon. This needs to be ensured by the MASC user at the beginning of the experiment.
* The software assumes by default that 12.5 mm lenses are used. This can be changed via keywords. Ask Ralf for detail if you ever want to change this.
* The software assumes certain empirical vertical and horizontal offsets between the three different images. These can also be changed via keywords. It needs to be ensured these offsets are correct for the experiment to be integrated. Ask Ralf.

# Data analysis

There are three different object types you need to be familiar with. For simple data analysis you will only need the first two. If you want to write your own data analysis routines you will also need the third object.

1. MASCOBS\_CONTAINER: provides a container class that holds a number of MASC observations
2. MASCOBS: One MASC observation consisting of fallspeed, three images stored as FLAKEROI objects, and a Julian date.
3. FLAKROI: One image plus a variable number of auxiliary pieces of information about the image

## MASCOBS\_CONTAINER

This is the central class for working with masc data fields. It is based off the IDL LIST data type and provides top-level access to all data. A mascobs container provides functionality that can loosely be split into:

* Data evaluation and access functionality
* Data subsetting, merging, and saving functionality
* Time-related functionality.
* Visualization

### Data evaluation and access

There are two generalized data evaluation and access functionalities:

#### Evaluate

The evaluate functionality provides an interface to apply user-defined functions to the entire dataset, i.e. to each MASC image stored in the object. The calling sequence is as follows:

IDL> result = o-> evaluate(‘do\_what\_i\_want\_you\_to\_do’)

When first called with a new evaluation call (do\_what\_i\_want\_you\_to\_do ), evaluate expects to find a routine

flakeroi\_\_do\_what\_i\_want\_you\_to\_do.pro

which tells it what exactly to do with the function call. For more details, see the definition of flakroi. There are various evaluation routines already predefined and described under flakeroi.

#### ToStruct

ToStruct converts an object container to an array of structures.

IDL> result->o->toStruct(dereference=dereference)

The optional keyword dereference copies all pointers in the object into new fields, so that even after you destroy the initial object, the pointers in result will be valid. This option is memory intensive because each flake image will be copied.

### Data subsetting, merging, and saving

#### Subset

IDL> onew = o->subset( indices, dereference=dereference)

The above call creates a new container object only holding the elements specified by indices.

The optional keyword dereference copies all values and fields from obj2 into obj1 (instead of just copying the pointers).

This option is memory intensive and slow. It is implemented following Fanning’s suggestion here ( <https://www.idlcoyote.com/tips/copy_objects.html> , see clone objects at very bottom.) All warnings expressed by Fanning apply here as well.

#### Cat

Cat concatenates two macsobs container objects.

IDL> obj1->cat, obj2, dereference=dereference

The above call adds all elements of obj2 to obj1.

The optional keyword dereference copies all values and fields from obj2 into obj1 (instead of just copying the pointers).

This option is memory intensive and slow. It is implemented following Fanning’s suggestion here ( <https://www.idlcoyote.com/tips/copy_objects.html> , see clone objects at very bottom.) All warnings expressed by Fanning apply here as well.

#### Saving and restoring data

Separate routines for saving and restoring data are not implemented. Use the IDL standard commands SAVE and RESTORE.

### Time-related container functions

Time is internally stored in IDL Julian day format. For convenience, there are three time-related access functionalities:

#### JulDay

IDL> jul = o->Julday()

Return Julian day for all observations in container. Same as:

IDL> jul = o->evaluate(‘julday’)

#### CalDat

IDL> o->caldat,m,d,y,hh,mm,ss,doy

Return numeric values for month, day, year, hour, minute, second, and day of year for all observations in container.

#### SCalDat

IDL> o->Scaldat,m,d,y,hh,mm,ss,doy

Return strings for month, day, year, hour, minute, second, and day of year for all observations in container. Format is I2.2, I2.2, I4.4, I2.2, I2.2, I2.2, I3.3.

## Visualization

#### Quicklook

The quicklook functionality allows the user to screen quickly through large numbers of snowflakes in the container. The following call will display all flakes sequentially in a regular IDL graphics window:

IDL> o->quicklook,wait=<seconds>

The optional wait keyword specifies the display time in seconds. A good value is 1.0. Default is 0.1.

# MASCOBS

The MASCOBS object constitutes one MASC observation. The container object discussed above holds many MASCOBS objects.

# FLAKEROI

This holds one image plus a variable set of auxiliary information. The following functions are coded up for flakeroi and can be used either on individual flakeroi images or, using the **evaluate** function, on mascobs\_container objects).

The mascobs\_container call is in all cases as follows:

Result = o->evaluate(‘NAME’)

Where NAME stands for the function names listed in the following table.

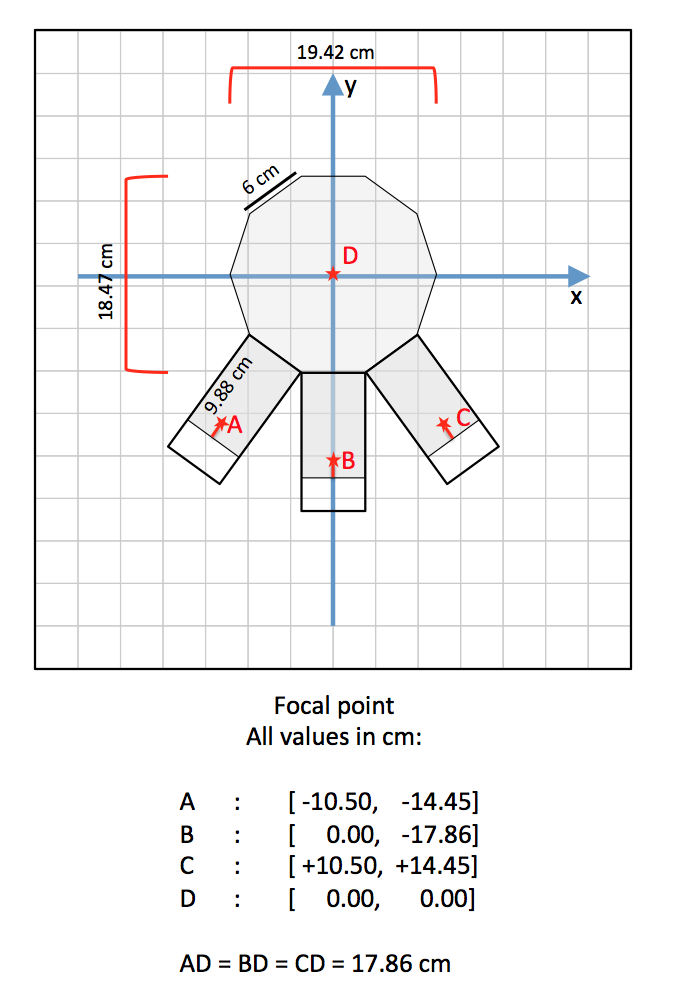
|  |  |  |
| --- | --- | --- |
| Name | Functionality | Example |
| MEAN  (flakeroi\_\_mean.pro) | Returns mean value of grey values of the snowflake. For a container it returns 3 x the number of observation triplet in the container holding the mean value for all three cameras for each observations | m |
| MASK  (flakeroi\_\_mask.pro) | Returns a pointer to the mask for each image. For a container it returns 3 x the number of observation triplet in the container holding the pointers to all masks. | m=o->evaluate('MASK')  img=IMAGE(\*m[1,987])  will show the mask for image 987, middle camera, e.g.: |
| IMG  (flakeroi\_\_img.pro) | Returns a pointer to the snowflake image. For a container it returns 3 x the number of observation triplet in the container holding the pointers to all masks. | m=o->evaluate(‘IMG')  img=IMAGE(hist\_equal(\*m[1,987]))  will show the image for image 987, middle camera, e.g.: |
| ELLIPSE  (flakeroi\_\_ellipse.pro) | Returns a structure or array of structures with fitted ellipse parameters. X0,Y0 position in image, Major,Minor axes length in pixel. Angle of rotation of major axis off positive x-axis, mathematically positive. | e=o->evaluate('ellipse')  help,e[0]  \*\* Structure <102b1348>, 5 tags, length=20, data length=20, refs=3:  X0 LONG 219  Y0 LONG 196  ANGLE FLOAT -34.0980  MAJOR FLOAT 467.348  MINOR FLOAT 310.243 |
| TEXTURE  (flakeroi\_\_texture.pro) | Returns mean greylevel co-occurrence matrix value for shift of 1. |  |
| FRACDIM  (flakeroi\_\_fracdim.pro) | Returns fractal dimension of snowflake outline. (E.g. coast of England has a fractal dimension of 1.25) |  |
| PERIMETER  (flakeroi\_\_perimeter.pro) | Returns perimeter of snowflake in [cm] |  |
| AREA  (flakeroi\_\_area.pro) | Returns area of snowflake in [cm2] |  |
| STDV, LAPALCE2, CENTROID, SOBEL | As of now undocumented but you can probably guess what they return. |  |

There are a few other parameters you can return using

o->evaluate(‘NAME’)

Those do not have a corresponding function flakeroi\_\_NAME.pro because they are initialized in the beginning when the container is populated.

* **X**: Best estimate of position of snowflake in x-direction [cm], see figure below for axis definition
* **Y**: Best estimate of position of snowflake in y-direction [cm], see figure below for axis definition
* **Z:** Best estimate of position of snowflake in z-direction [cm], i.e. vertical
* **MULTIFLAKE**: Returns the number of flakes identified in original image (out of which the best match was picked).



# Using this

Plot fallspeed against maximum diameter of particle:

IDL> f = o->evaluate(‘fallspeed’)  
IDL> e = o->evaluate(‘ellipse’)

IDL> dmax = MAX(e.major,DIM=1)

IDL> PLOT,dmax,f,PSYM=1

# Writing your own evaluation routines

* See flakeroi\_\_mean.pro as an example….
* Pick a unique name, such as flakeroi\_\_myfunct.pro
* Make sure it returns something that’s a scalar. If you need to return more than one value, return a structure.
* You can then use it like o->evaluate(‘myfunct’) and it will return an array of whatever want it to return.