Allsky cloud-mask Algorithm (ASCA)

Marcus Klingebiel[1] **and Tobias Machnitzki**[2,*]

[1]Max-Planck-Institut for Meteorology, D-20146, Hamburg, marcus.klingbiel@mpimet.mpg.de
[2]Max-Planck-Institut for Meteorology, D-20146, Hamburg, tobias-machnitzki@web.de

ABSTRACT

The All-Sky Cloud mask Algorithm (ASCA) is a program for automatic detection of clouds in a picture of the sky. So far it just works with images made by the cloud cam on the Barbados-site of the Max-Planck-Institut f”ur Meteorologie Hamburg.
ASCA does not just detect the clouds, but also calculates the cloudiness in percent and in eighth, as well as saves the image rescaled for saving memory. It works live, meaning for the calculation of one image it needs between 4 and 8 seconds, while the camera takes a picture only every minute.
There are two versions working for two different tasks. First is for using it for the website *barbados.zmaw.de*. It is called *ASCA_BCO.py* and its implemented as module in the *create_BCO_animation.py* script.
The second version is for running the script on the *.avi* quicklook files. The intention therefore is to analyse the data observed for the last five years. This script is called *ASCA_climate.py*.
The whole code is written in python 2.7.

1 Getting started

1.1 ASCA_BCO.py

This script defines the function `cloudiness(InputFilePath)`, returning the cloudiness and a timestamp. For input it needs the Path of the folder, where the `.jpg` images are in. For example:

```
cloudiness_list, time_list = cloudiness('/data/.../allsky/cc1605/01/')
```

It returns the values in lists, containing the results for the whole calculated period.

1.2 ASCA_climate.py

This script defines a similar function called `cloudiness(InputFilePath)`. It is called by the script `ASCA_climate_starter.py`, which defines the period and saves the values to NETCDF files. For the calculation of the cloudiness it just uses the `.avi` video files, which makes this script much faster, therefore a little less precise. For running it needs the c-shell script aviconverter.tcsh in the same folder. This shell script is for splitting the videofile in to images and renaming them properly.

2 structure

The structure of the ASCA code is as followed:

1. Settings
2. Calculating the SI-parameter
3. Reading the inputfiles
4. Getting day and time
5. Calculating SZA
6. Open csv-file
7. Read and rescaling the image
8. Converting the image to an array and removing unnecessary parts around the true allsky image
9. Calculating position of the sun in the picture
10. Excluding sun
11. Calculating SI for outer area
12. Calculating SI for inner area
13. Calculating cloudiness
14. Mirror image
15. Adding text
16. Saving values to csv files
17. Saving picture

Not in every version all parts are used and implemented.
You will find the same titles as comments in the code.

2.1 Settings

All variables set here are to be changed by the user for his/her individual needs.
See the comments in the code for more information.

2.2 Calculating the SI-parameter

This part calculates a parameter dependent from its distance to the sun. Most cloud-detection-scripts use a non variable parameter:
A pixel contains the colors red, blue and green. Each color-part can have a number between 0 and 255. So we define an array for the picture in which each pixel-position gets 3 values, for example `image_array[1500,1300,:] = [255,0,0]`. This means that the pixel at the position(1500,1300) has the color (255,0,0), which is pure red. To calculate the Sun Index (SI) you just do:

$$SI = \frac{blue - red}{blue + red}$$

(Based on Letu et al. , 2014)
Or as seen in the code:

$$SI = \frac{array[:,:,2] - array[:,:,0]}{array[:,:,2] + array[:,:,0]}$$

Which leaves us with an array of the same size as the picture, where for each position now has a SI-value. We just need to compare this index with a reference value (usually 0.1), and afterwards can say whether this pixel belongs to a cloud or not (See section 'Calculating SI for outer area' for more information).
In this cloud-detection-script, the reference value is not static but gets calculated depending on its distance to the sun (see figure 1). Therefore the missdetection of clouds which are actually just bright sky is much less.

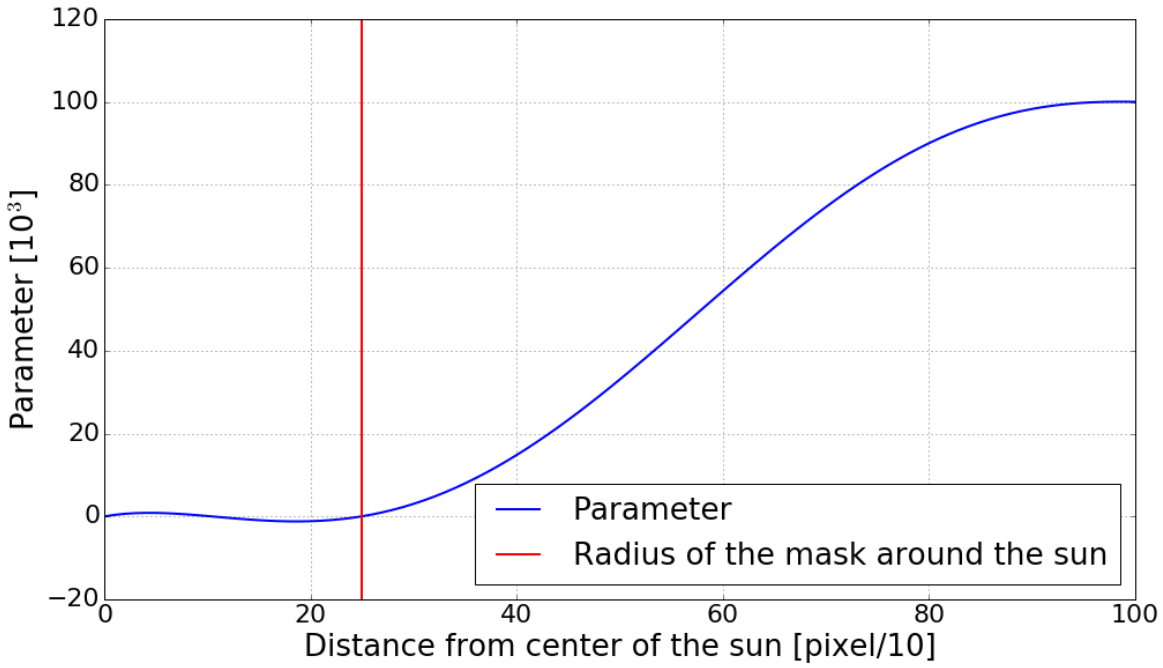


Figure 1. Function of the SI criterion distribution dependent on the distance to the sun.

2.3 Reading the inputfiles

The variable OutputPath just gets important, if you want to save your result as an image. See section 2.17 for more information.

2.4 Getting day and time

The name of the inputfile should be ccYYMMDDHHMMSS.jpg. For example the file from the 5th of June 2016, 12:00 and 36 seconds would be called '`cc160605120036.jpg`'. Out of this name the script regenerates the day and time.

2.5 Calculating SZA

SZA stands for `Sun Zenith Azimuth`.
In the variable `tus` the geographical position of the camera is saved. In `times` the day and time when the image was taken is stored. `times_loc` combines the time and location information of the image and out of these generates the timezone. In it is saved the time and timezone of when the image was taken.
With the function `pvlib.position.get_solarposition()` stored in `pos` we now get the Azimuth and the zenith of the sun. We just need the Zenith first and store it in the variable `sza`.
The script only continues if `sza <= 85`, setting this as the criterion for being daytime.

2.6 Open csv-file

If the option `TXTFile` is set `True`, a textfile will be generated. In this section the header (1st line) is created. The file will be stored in the output path and will include the full date and time in its name and ends with `'..._ASCA.csv'`.

2.7 Read and rescaling the image

In this part of the code, the image is opened and gets resized. The rescale factor is set in the section 'Settings'. The highest it can be is 100, the lowest is 1. If set to 100, the original imagesize will be used. If set 1, the image gets rescaled to 1% of its original size. The smaller the size gets, the faster the script does run, but the quality suffers.
The image used for calculation of the cloud coverage is the same in figure 2 as in figure 3, but in 2 the the original image size of 2592x1944 is used, while in 3 the quality was scaled down with the `scale_factor = 30`. Therefore the cloudiness of the better-quality-image gets calculated as 35.6%, while with the lower-quality-image the cloudiness gets calculated as 44.6%.

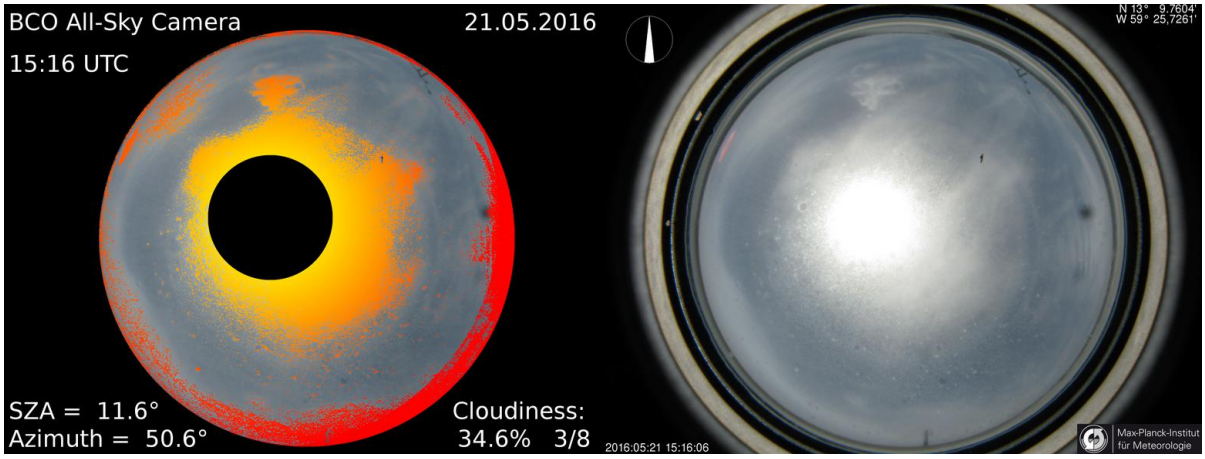


Figure 2. Cloud coverage calculated with a full scaled image (scale_factor = 100) of the size 2592x1944.

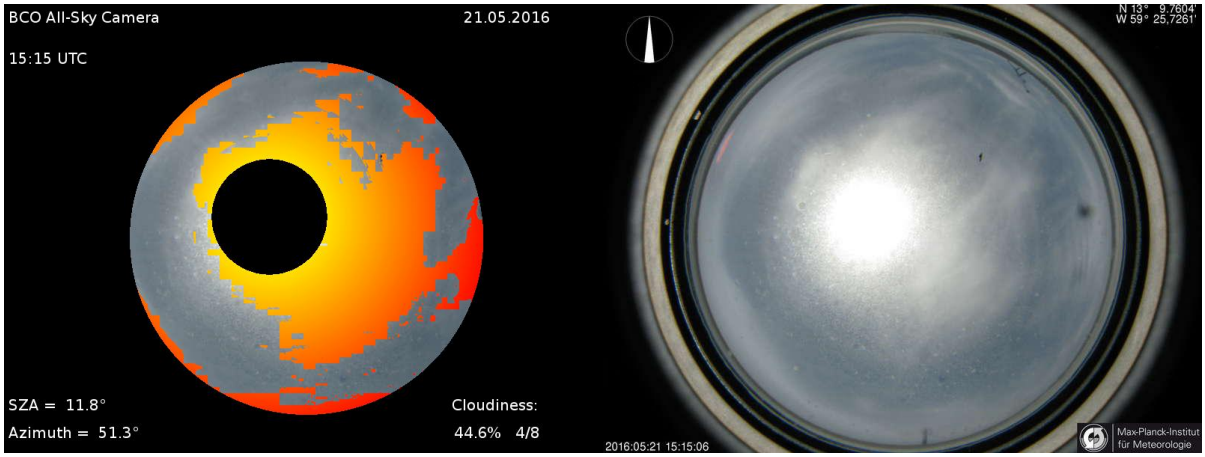


Figure 3. Cloud coverage calculated for the same image as in 2, but with a scale_factor of 30.

We mirror the image here, to make the calculation of the position of the sun later easier. Usually you are looking from above to the ground and then want to calculate the position of the sun. But for we are on the ground and looking up in the sky the calculation needs to be the other way round. To be able to use mathematical formulas already approved, we make it easy for us and just mirror the image.

2.8 Converting the image to an array and removing unnecessary parts around the true allsky image

First we create a True-False-Mask, which is true for where the actual image of the sky is, and false for the black area and the rings of the camera-case.
Second we generate an numpy.ndarray from the picture making further calculations easier. After doing so we use the just generated mask and put it on the array, colouring everything black, that is not part of the image itself (See image 4).
If `Radius_synop == True` then the size of the black area gets bigger, leaving a smaller area of an actual image of the sky. The area of the image than just covers an angle from 60 ° from the center of the image (the zenith).

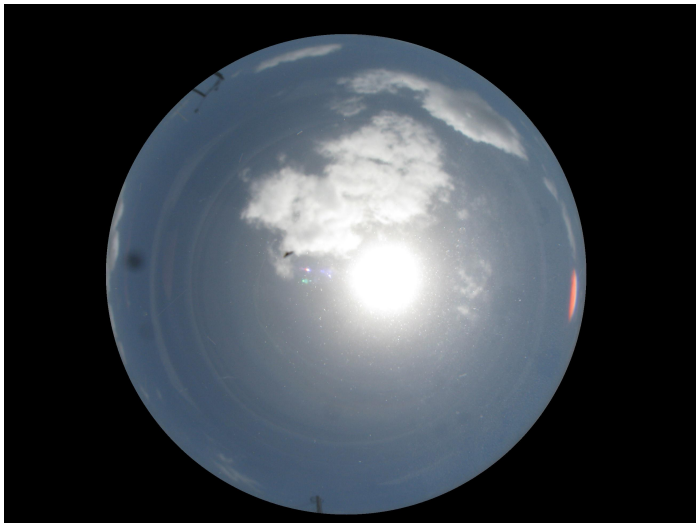


Figure 4. Image how ASCA sees it after removing the outer space around the original image.

2.9 Calculating the position of the sun in the picture

`sza = sza -90`, because we want the angle measured from the zenith, not from the ground. Same for the azimuth.

2.10 Excluding sun

After finding the x and y position of the center of the sun in the picture, we create a True-False-Mask, which is true for a disc with the radius of the sun in the picture and false outside. After using this mask on our image-array the result looks like in picture 5

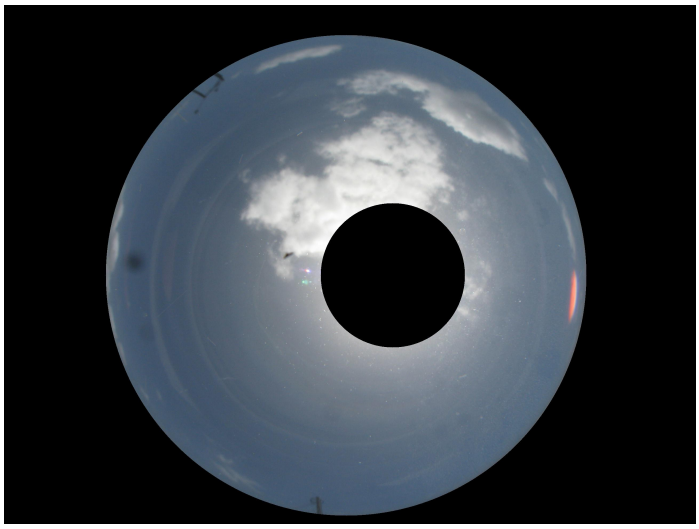


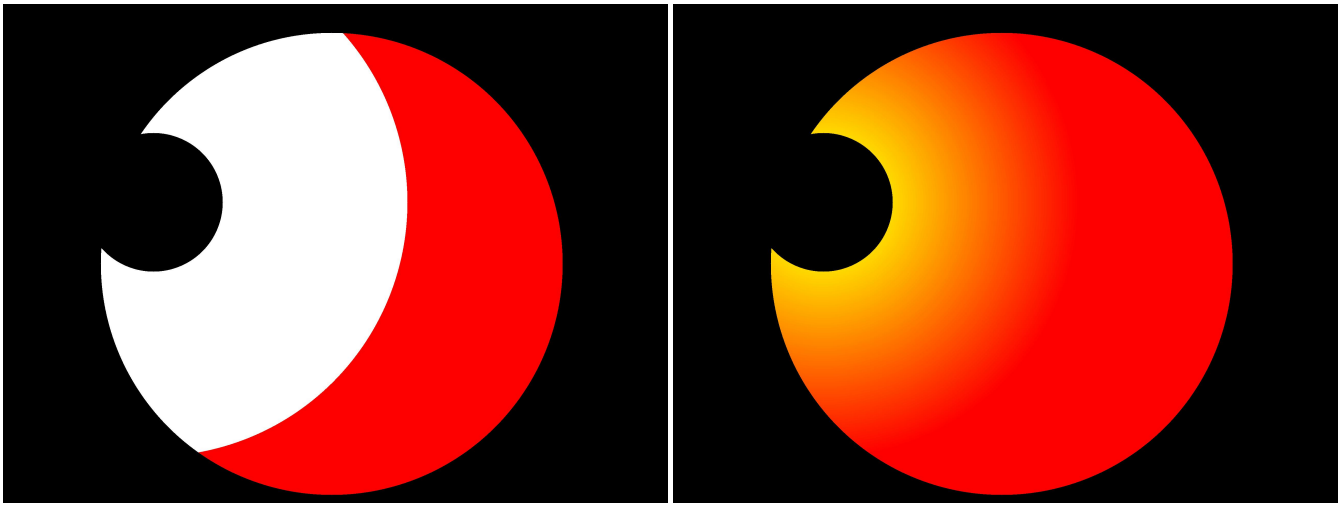
Figure 5. The image as seen by the algorithm after removing the outer area of the picture and the sun.

2.11 Calculating SI for outer area

In section 2.2 the actual calculation is explained.
In this part, the calculated factor for each pixel gets compared, with a reference value. This value is set by default to 0.1. If the Sun Index of a pixel is less the 0.1, it is declared as a cloud, therefore being coloured red (`image_array[x,y,:] = [255,0,0]`).
By creating the temporary True-False-Mask `sol_mask_double`, it is guaranteed, that not the sun itself or the area around it is coloured red, but just the outer area (see figure 6.a).

2.12 Calculating SI for inner area

We duplicate the `image_array` and call the new one `image_array_c`. The reason for this is to have one array for calculating the cloudiness later, where everything that got recognized as a cloud is coloured red. The other array is for saving the array to a picture, where everything that got recognized gets coloured depending on its SI-criterion seen in figure 1.
Now we part the before called 'inner area' in 100 rings and for each ring we compare, if the Sun Index is smaller than our SI-criterion from figure 1 (see figure 6.b).
In our array for counting `image_array_c` we paint the clouds red. In our array for saving as an image, we paint clouds depending on what the SI-criterion was used to detect that part as a cloud (see picture 7). Because the SI-criterion is a function only of the distance to the sun, the colour can be seen as an indicator of what distance to the sun in the picture the cloud has, as well.



(a) Just the red painted area gets calculated by the part of the script 'Calculating SI for outer area'. The blank area gets calculated by the part 'Calculating SI for inner area'.

(b) The colour might seem, as if there was a fluent passage from yellow to red, but in fact, there are 100 discrete rings, going from the rim of the sun to the part where the outer area starts (see figure 6.a).

Figure 6. Outer and inner area for the SI-criterion.

2.13 Calculating cloudiness

For calculating the cloudiness, we just count the red pixels (clouds) and all pixels that are not black (complete area) in our counting array `image_array_c`.
To get the cloudiness in percent we just need to find the ratio:

$$cloudiness = \frac{100}{completearea} \cdot clouds$$

Afterwards we convert our `image_array` into a picture.

2.14 Mirror image

We mirror the image here a second time to get the intentional orientation back (see section 2.7).

2.15 Adding text

All our results for the calculated picture are being drawn on the picture itself (see picture 8).

2.16 Saving values to csv file

Earlier the csv file was created and the header is already written.
Now the same information as on the picture itself are written into this textfile. Each line represents one picture. The delimiter to separate columns is the comma (,).

2.17 saving picture

The picture is submitted to the converter module, where the original image and the coloured one are glued together and scaled down for easier storage. If the converter module is commented out the result will look like in picture 8. Else you will get a picture like in figure 9.

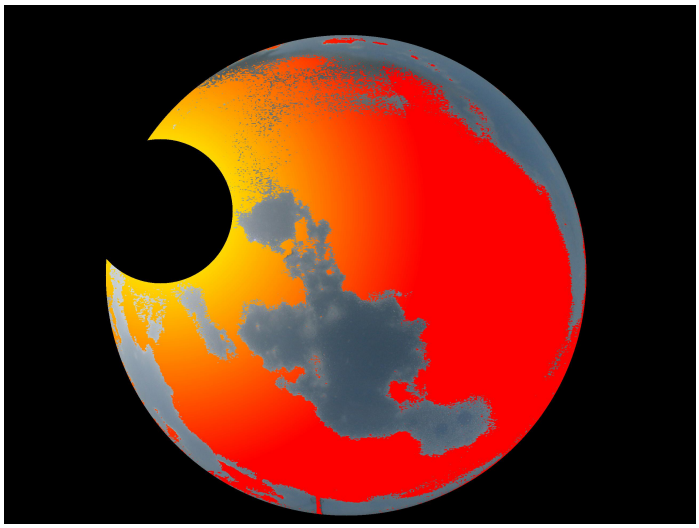


Figure 7. All detected clouds are being coloured, depending on what SI-criterion was used to detect them.

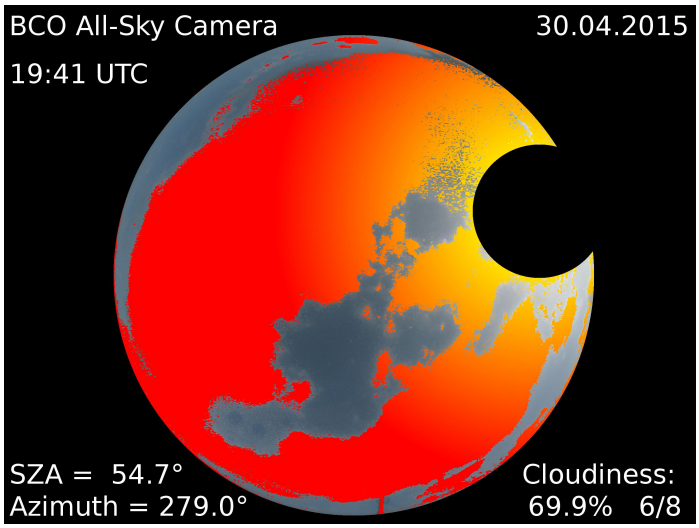


Figure 8. The final image ASCA creates, if the converter is not being used. All results are written in the corners.

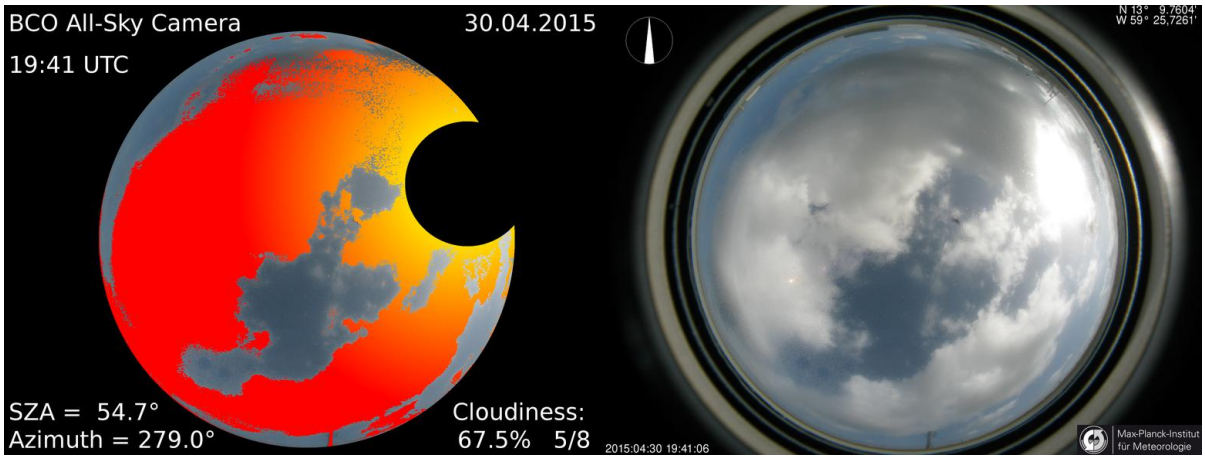


Figure 9. The final image ASCA creates, if the converter is being used. On the left side the clouds are coloured and in the corners are the results. On the right side you can see the original image.