

Internship report  
Precipitable water comparison and All-Sky  
16.03.2020 - 31.07.2020

Joanna Szulc, joanna.szulc.pl@gmail.com

October 15, 2020



# Contents

<b>1</b>	<b>Project I: Precipitable water comparison</b>	<b>5</b>
1.1	Ground-based GPS calculations . . . . .	5
1.2	Radiosondes launched at Andøya Space Center . . . . .	7
1.3	AERONET data . . . . .	8
1.4	Comparison between radiosonde and CIMEL data . . . . .	9
1.5	What remains to be done . . . . .	13
<b>2</b>	<b>Project II: All-Sky</b>	<b>15</b>
2.1	Introduction . . . . .	15
2.2	Hardware . . . . .	16
2.3	System schematics . . . . .	17
2.4	Default settings . . . . .	20
2.5	Mounting the system 2020 . . . . .	20
	2.5.1 Issues solved during mounting . . . . .	20
	2.5.2 Unresolved issues . . . . .	21
	2.5.3 Set-up . . . . .	22
2.6	Review of cloud detection options . . . . .	25
	2.6.1 Popular methods of cloud detection . . . . .	25
	2.6.2 Using machine learning for cloud detection . . . . .	33
	2.6.3 Post-processing RAW images with C++ and Python . . . . .	36
2.7	Image distortion and projection onto a map . . . . .	38
2.8	Software . . . . .	40
	2.8.1 uioCam v2.1 . . . . .	40
	2.8.2 ICE . . . . .	43
	2.8.3 Sun blockade . . . . .	47
	2.8.4 Post-processing software . . . . .	47
	2.8.5 For users . . . . .	47
	2.8.6 For developers . . . . .	50
	2.8.7 End note for future developers . . . . .	52
2.9	Appendix I: Codes, older documents and scripts . . . . .	53



# Chapter 1

## Project I: Precipitable water comparison

Goal: Development of automatic routines allowing for real-time calculation of integrated water vapour column from high precision GPS available at the Alomar Observatory. Comparison of the data with the water vapour profile from the radiosondes launched from Andøya Space Center and CIMEL photometer (part of AERONET) at the Alomar Observatory.

### 1.1 Ground-based GPS calculations

Calculations based on the time delays of the radio signal from GPS system can provide precise, independent of weather conditions, almost real-time data on the amount of water vapour (WV) in the atmosphere. Technique is well known and developed since the '90s and can be implemented in various ways. Usually radiosondes on meteorological balloons or mid-IR radiometers are used for creating WV profiles. As WV is an important greenhouse gas and contributes to almost every aspect of the weather and climate the additional measurements utilizing a GPS signal can provide validation of other methods. WV estimations using ground-based GPS receivers allow only to measure the vertically integrated amount of WP known in meteorology as precipitable water or integrated water vapour (IWV). Precipitable water is defined as total atmospheric water vapour contained in a vertical column between any two specified levels (in this case between the ground-based GPS receiver and the satellite). Commonly it is measured in terms of the height (in mm) to which that water vapour would stand if condensed in the same column. In the case of the GPS based calculation this water column is in fact a cone that starts at the ground-based receiver (at Alomar) with a surface of less than  $1\text{m}^2$  and reaches to around  $100\text{km}^2$  at the level of the satellite. GPS radio signal is delayed while going through the atmosphere by these main contributions:

1. Ionosphere – this contribution may be corrected by acquiring the signal from two different satellites with different signal frequencies.
2. Neutral atmosphere (Total Delay, or TD) that can be further separated into Zenith Hydrostatic Delay (ZHD) and Zenith Wet Delay (ZWD)

ZHD – contributes 90% of the delay and can be modelled and eliminated with surface pressure measurements. If possible, the measurements should be taken at the same time and place as the GPS signal with precision of at least 0.5 hPa. Especially important for observatories situated at mountain peaks (like Alomar) and in environments with dynamical weather conditions.

ZWD – contributes around 10% of TD and is proportional to the amount of water vapour in the column. The calculations require accurate surface temperatures.

Depending on the amount of ground-based stations and the satellites used there is few different measurements that can be taken with these methods:

1. From a single observation point

Measurements of a single water column: allows to calculate IWV between the ground station and the satellite, the ground station connects only to one satellite at the time. Without other adjustments this would provide slant water, which is the WV column along the beam of the GPS signal. Modifying the calculations allows to scale it to zenith and estimate the vertical IWV.

Three-dimensional water vapour topography: connecting with many different satellites from one ground station and measuring the slant water in different direction allows to create an estimated 3D map of WV above the station. If the measurements and the calculations are done in real-time it provides the dynamics of the WV in time as well.

2. Network of ground-based GSP: International GPS Service, German Research Centre for Geosciences (GFZ) and others use wide networks of GPS ground stations in comparison with data from World Meteorological Organization (WMO) and European Centre for Medium-Range Weather Forecasts (ECMWF) to create world-wide models of WV and its atmospheric dynamics.
3. Occultation: this is an entirely different method of calculating WV using two satellites and no ground stations at all. It allows to estimate IWV in the column alongside the signal beam between the two satellites.

There are multiple models and solutions for calculating IWV from ZWD delay ([1], [2]), as well as calculating the ZWD delay from the raw GPS data.

In our case the Norwegian Mapping Authority will provide the raw GPS data to Deutsches GeoForschungsZentrum which calculates in near real-time the value of the IWV.

Summary based on: [1], [2], [3], [4], [5], [http://glossary.ametsoc.org/wiki/Main\\_Page](http://glossary.ametsoc.org/wiki/Main_Page).

## 1.2 Radiosondes launched at Andøya Space Center

Data from radiosonde launches is provided by Norwegian Meteorological Institute (<https://thredds.met.no/thredds/catalog.html>).

Radiosondes are launched from Andøya Space Center twice a day, usually around 12 am and 12 pm with slight alterations due to weather and flight plans. The measurements include geopotential height, pressure, temperature, relative humidity and dew point temperature. From those values IWV can be estimated for each launch in few steps:

First saturated water pressure (SWP) needs to be calculated. SWP is defined as the pressure above a surface of water, in a closed container in a state of equilibrium. It can be understood as the maximum capacity of the air to “hold” certain amount of water in given temperature. This also means that when partial pressure of water vapour is equal to SWP the relative humidity is 100%. There are multiple equations allowing to calculate it, we use the Buck’s equation over water:

$$P = 0.61121 \exp\left((18.678 - \frac{T}{234.5})\left(\frac{T}{257.14 + T}\right)\right) \quad (1.1)$$

P being pressure in hPa and T temperature in K.

Then the partial pressure of water (PP) can be calculated by substituting the temperature in 1.1 with the dew point temperature:

$$P_{PP} = 0.61121 \exp\left((18.678 - \frac{T_{dewpoint}}{234.5})\left(\frac{T_{dewpoint}}{257.14 + T_{dewpoint}}\right)\right) \quad (1.2)$$

$T_{dewpoint}$  is dew point temperature in K.

Using ideal gas law we can then estimate the number of moles of air and water vapour in the volume of air:

$$n = \frac{PV}{RT} \quad (1.3)$$

All variables are in SI unit:  $n$  is amount of substance in  $mol$ ,  $P$  pressure in  $Pa$ ,  $V$  volume in  $m^3$  and  $T$  the temperature in  $K$  and  $R$  is the universal gas constant in  $J/Kmol$ . Knowing the molecular mass of dry air ( $m_{mol_{air}} = 28.9628 \text{ g/mol}$ ) and water vapour ( $m_{mol_{wv}} = 18.0152833 \text{ g/mol}$ ) we can then calculate the mass of air or water vapour in the considered volume:

$$m_{air} = n_{air} * m_{mol_{air}} \quad (1.4)$$

$$m_{wv} = n_{wv} * m_{mol_{wv}} \quad (1.5)$$

We are trying to estimate water column having dimensions of 1 cm x 1 cm x 20 km to calculate the IWV in  $\frac{cm^2}{kg}$ . The mass of dry air or water vapour is calculated for every point recorded by the radiosonde with the height of the volume being the difference between the altitude at which the record was made and the next one. This way the whole length of the column is covered, but we are assuming that the temperature and pressure between those altitudes are uniform, which is far from being exact. The higher the height resolution of the data the better such integration will be.

Integration of the water column is made between 376 m and 30 000 m. Starting point is the position of Alomar Observatory above the sea level to make the comparison with the CIMEL photometer stationed there as exact as possible. The upper border was set at 30 000 m both because the humidity sensors above those altitudes tend to be imprecise and because the majority of WV can be found in the lower levels of the atmosphere.

### 1.3 AERONET data

Data from CIMEL photometer was acquired through AERONET network:

[https://aeronet.gsfc.nasa.gov/new\\_web/index.html](https://aeronet.gsfc.nasa.gov/new_web/index.html)

The AERONET (AErosol RObotic NETwork) project is a federation of ground-based remote sensing aerosol networks established by NASA and PHOTONS (PHOTométrie pour le Traitement Opérationnel de Normalisation Satellites; Univ. of Lille 1, CNES, and CNRS-INSU). The project provides long-term, continuous and readily accessible public domain database of aerosol optical, microphysical and radiative properties for aerosol research and characterization, validation of satellite retrievals, and synergism with other databases.

AERONET collaboration provides globally distributed observations of spectral aerosol optical depth (AOD), inversion products, and precipitable water in diverse aerosol regimes.

To gather the data CIMEL Electronique CE318 multiband sun photometer was used. CIMEL performs measurements of spectral sun irradiance and sky radiances. The latest CE318-T model also performs night time measurements of the spectral lunar irradiance.

The AERONET data, through its web download tool, provides already calculated precipitable water in cm. For comparison with radiosonde data the daily averages of the IWV were used. CIMEL takes measurements only when Sun or Moon are visible and unobstructed by clouds. As Alomar Observatory sees both the polar nights and severe weather conditions there are significant gaps in the data during the year in comparison to more reliable radiosonde data. At the same time while the conditions are good CIMEL takes the measurement every 15 min providing much better time resolution than radiosonde launches.



## 1.4 Comparison between radiosonde and CIMEL data

Data from radiosonde provides us with IWV in  $\frac{kg}{cm^2}$  and AERONET is precipitable water in cm. Simple conversion as seen in 1.6 allows us to compare the values.

$$PW_{cm} = IWV \frac{kg}{cm^2} * 1000 \quad (1.6)$$

Fig.1.1 shows the comparison of the measurements for the whole year of 2019. The points were plotted for daily averages taken from all the measurements available that day. The days when both AERONET and radiosonde data was usable were plotted in Fig. 1.2.

Fig. 1.3 details the differences between the AERONET and radiosonde measurements.

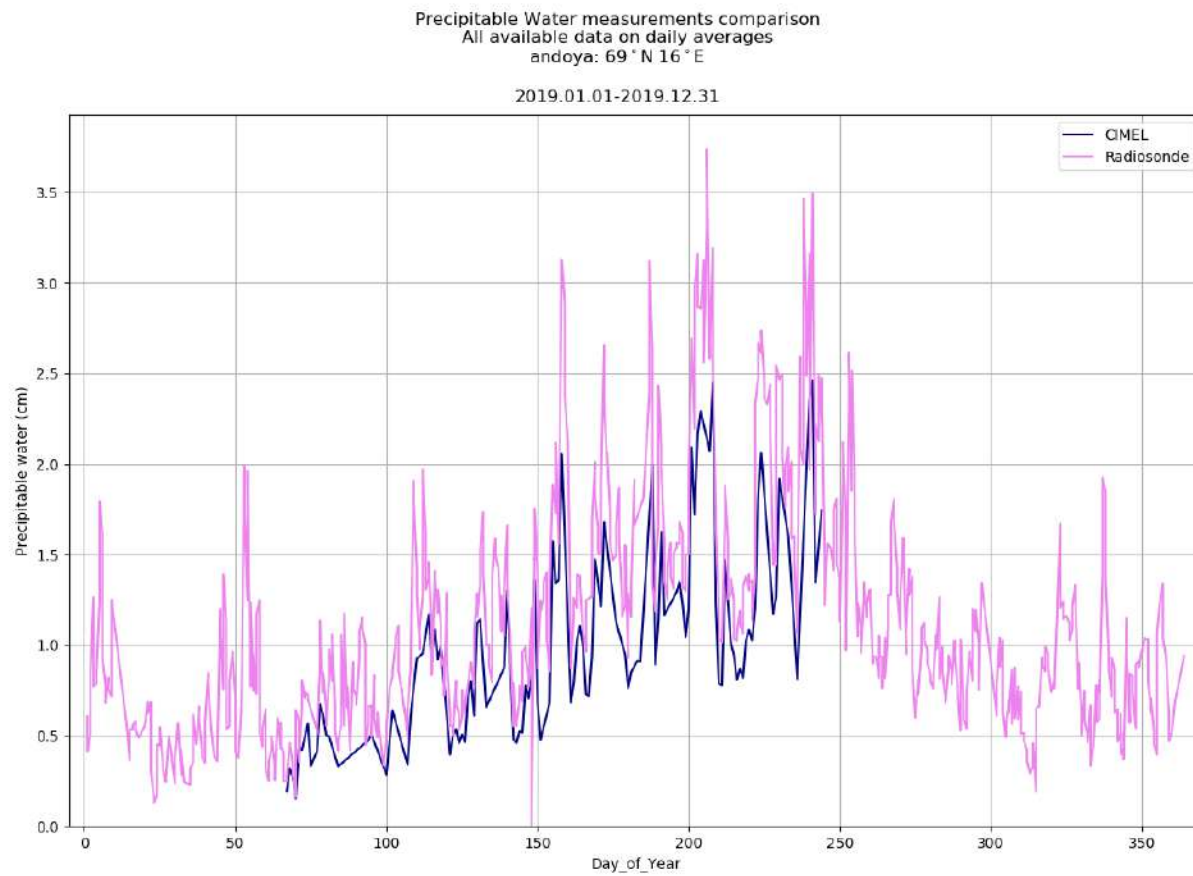


Figure 1.1: Comparison of precipitable water (PW) in cm between radiosonde and CIMEL measurements from 2019. Data is daily average from all available measurements.

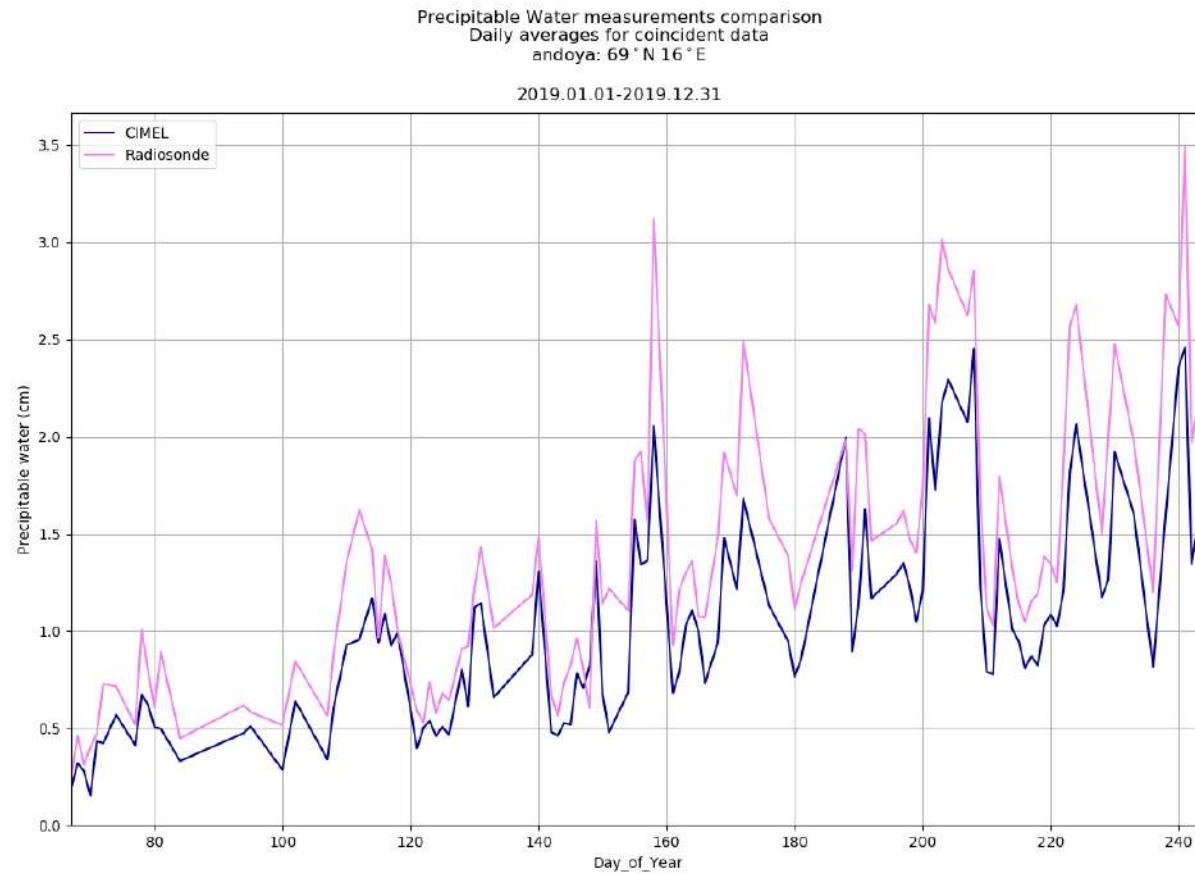


Figure 1.2: Comparison of precipitable water (PW) in cm between radiosonde and CIMEL measurements from 2019. Data is daily average from all available measurements. Only the days on which both CIMEL and radiosonde data was available is shown.

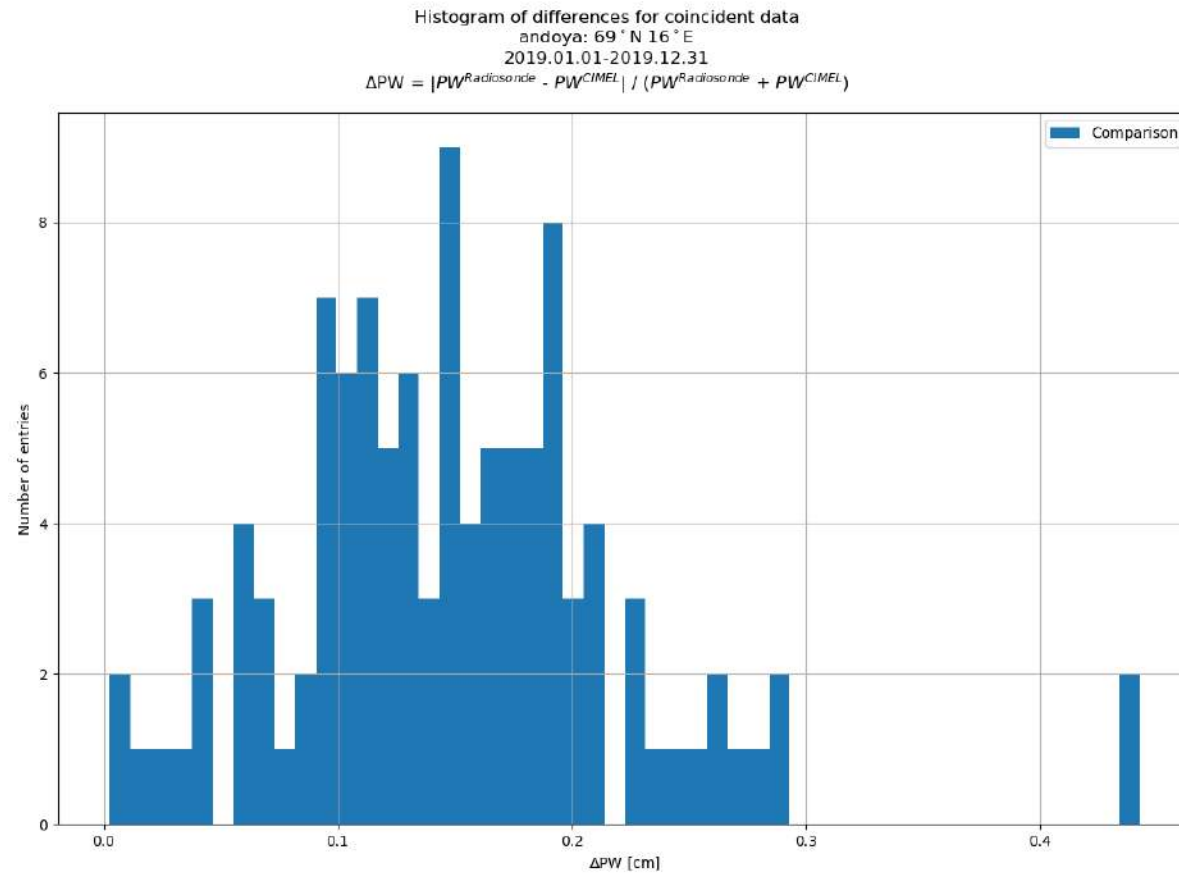


Figure 1.3: Histogram of normalized differences between AERONET and radiosonde measurements of precipitable water for the whole year 2019.

## 1.5 What remains to be done

While scripts for automatic download and comparison of radiosonde and AERONET data are finished, tested and ready to use the project still waits for the response from Norwegian Mapping Authority on the subject of allowing access to the GPS data to GFZ.

Contact person at GFZ is dr. Galina Dick ([dick@gfz-potsdam.de](mailto:dick@gfz-potsdam.de)), at ASC dr. Michael Gausa ([michael@andoyaspace.no](mailto:michael@andoyaspace.no)) and questions about the scripts should be directed to Joanna Szulc ([joanna.szulc.pl@gmail.com](mailto:joanna.szulc.pl@gmail.com)).

Script repository can be found at:

<https://github.com/Joanna-Szulc/ASC-precipitable-water-comparison>

If Norwegian Mapping Authority will allow access to the raw GPS data the GFZ will provide IWV calculated from the signal delays. Once that will be done the data can be integrated into comparisons between radiosonde and CIMEL data on WV and used to both observe the WV dynamics throughout the year and cross-reference the three different measurements.

There is also room for improvement in comparison between radiosonde and CIMEL data. Currently only daily averages of both datasets are taken into account, but with improvement to the script it should be possible to compare only those measurements that were taken, for example, in the same hour.

For observations of WV dynamics detailed analysis of measurement errors would be in order as well. The details on the devices used to gather the data are available respectively: for AERONET data on the website of the network, and for radiosonde in the netCDF files downloaded for the comparison (accessible through netCDF4 python library).



## Chapter 2

# Project II: All-Sky

**Goal:** Improvement and maintenance of the All-Sky system at Alomar Observatory for cloud detection and observation. Research of available image processing techniques available for cloud detection.

### 2.1 Introduction

The existing system is a conglomerate of various initiatives and projects, changed, modified and improved since 1995. Because of that information about the system are distributed over different manuals, documents and reports with varied degree of outdated information. This document is intended as a report from an internship project and will mainly deal with the newest addition to the systems, solutions that were considered for improvement (and if so, why they were dismissed) and the results achieved.

An actual manual of the system, describing both the hardware and software used in the current version (but including all the necessary information from the older documents) can be found in "All-Sky\_Manual\_2020".

Chapter	Part	More
Hardware	Camera	
	Fish-eye lens	
	Other mechanical parts	
	Additional optics	
	Controllers	
	Graphic schematic	
Software	Sony Remote	
	uioCam v2.1	
	ICE script	

## 2.2 Hardware

The tables beneath describe the All-Sky system as it was in August 2020. To see the information from the original manufacturers look toward older documents.

Table 2.1: List of hardware in the current set-up.

Function	Details
Fish-eye lens	Mamiya Sekor ULD fish-eye 24mm F4
Mechanical shutter	
Additional optics	
Filter wheel	5-positions, 3-inch filters
Filter wheel motor	SmartMotor Animatics
Camera lens	Canon FD 85 mm F1.2
Mount adapter	Sony NEX/E-mount to Canon FD
Camera	Sony A7 iii (ILCE-7M3)

Table 2.2: List of cables used in the current set-up.

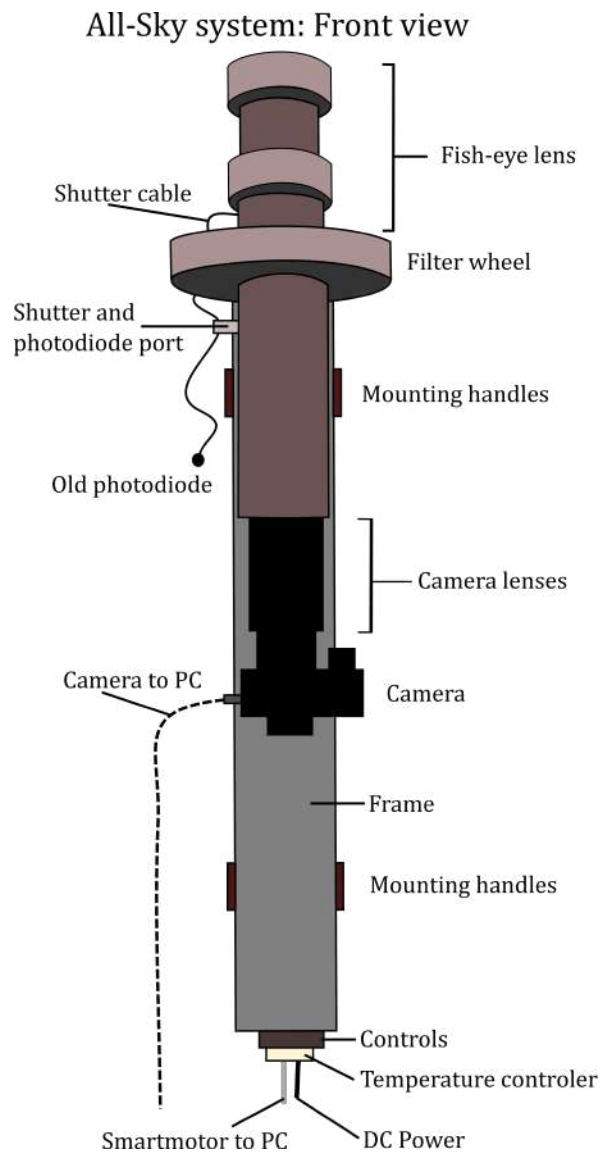
Function	Details
All-Sky controls to PC	RS-232 serial port, protocol details in the manual
Sony Camera to PC	Micro-USB to USB cable
All-Sky power	3-pin power cable

Table 2.3: List of control boards.

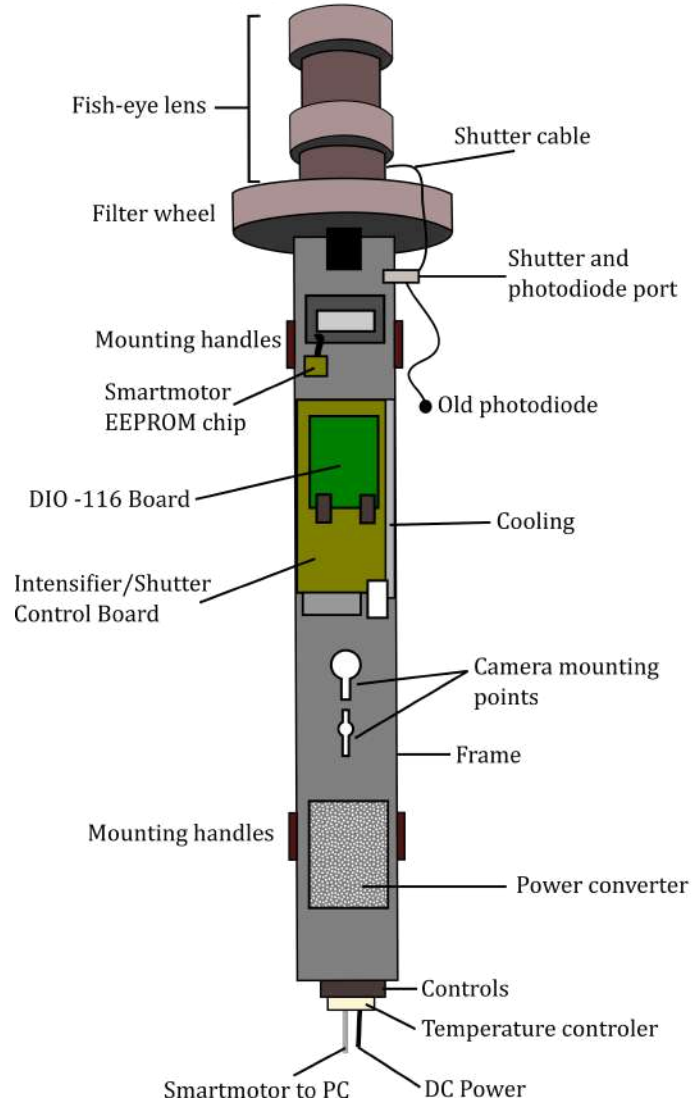
Function	Details
Power converter	-
Shutter/Intensifier Control Board	KEO Consultants, details in the manual
DIO-116 Board	$I^2C$ parallel I/O board for SmartMotor, details in the manual

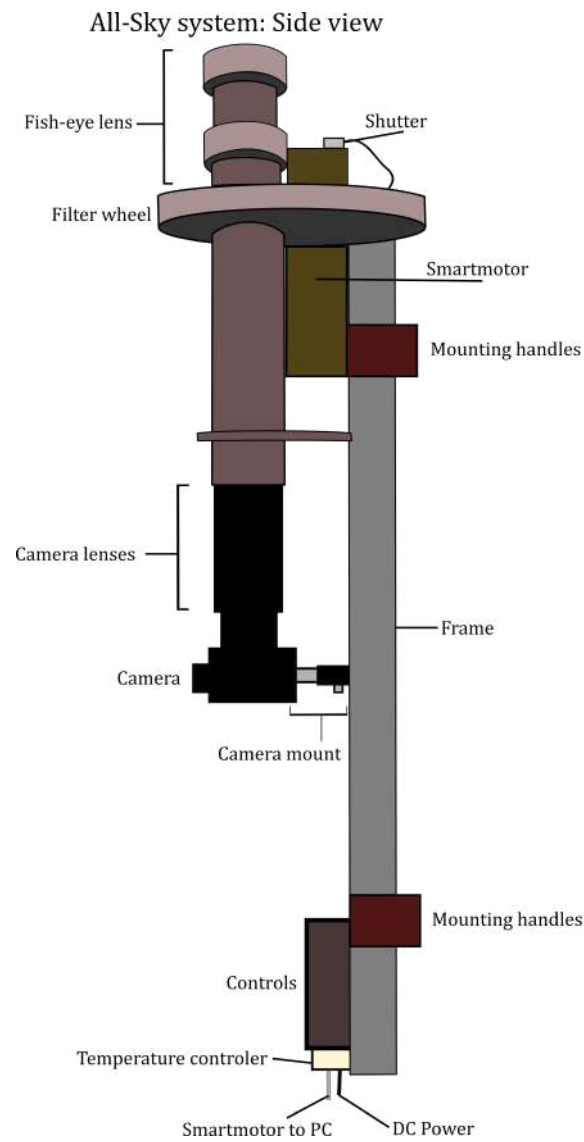


## 2.3 System schematics



## All-Sky system: Back view





## 2.4 Default settings

Settings like aperture number on fish-eye lens and camera lens, focus and photographic mode have to be manually set before pulling the system up to the dome and cannot be changed through Remote or uioCam software. Additionally some default settings like path for saving files and white balance settings need to be manually set in Remote before the automatic collection of the data can start. UioCam can send only the simplest commands to the camera through Remote. Making pictures with different settings is certainly possible, but it may interfere with the automatic program in uioCam. More on that in Sec. ??.

The largest changes in default settings is the aperture number both on fish-eye and the camera lens. The aperture was set to have the highest number possible, which corresponds to the smallest amount of light getting through to the camera matrix. Previously the lowest number possible was recommended because the system was used to observe the northern lights and the higher sensitivity was advantageous. High ISO sensitivity of the Sony camera and the nature of cloud observation allows for higher aperture setting which diminishes the effects of overexposure around the Sun and Moon.

Table 2.4: Default settings of the fish-eye lens and the camera.

Part	Setting	Value
Camera	Photographic mode	MANUAL
Camera	Switch button	ON
Camera lens	Aperture	f.16
Camera lens	Focus	Adjust manually, see the manual
Fish-eye lens	Aperture	f.22
Fish-eye lens	Focus	$\infty$
Fish-eye lens	Mode	SL-1B

## 2.5 Mounting the system 2020

In the beginning of 2020 All-Sky system was transported from Alomar Observatory in Andenes to Germany and back for the calibration of lenses. Afterwards it needed to be remounted on the platform that allows to move it up and down to the observational dome. This was used as an occasion to fix a few issues.

### 2.5.1 Issues solved during mounting

1. **Unresponsive Smartmotor** The shutter and filter wheel were powered up, but unresponsive after the test of the equipment. The full report from the troubleshooting can be found in a separate document, but the issue turned out to be a broken off EEPROM microchip. The EEPROM microchip stores all of the subroutines that drive the operation of the shutter

and filter wheel via uioCam software. After resoldering and reattaching the microchip the system started to function normally.

2. **Image not fitting on the camera chip** Previously the picture of the whole sky did not fit onto the camera chip and the circular images were cropped at one side as seen on the left in Fig. 2.5.1. To improve that the camera needed to be attached at a lower position and additional rings were mounted after the lens just to make sure that the whole picture was sharp and fitted in. The results can be seen on the right in Fig. 2.5.1.
3. **Maintenance** Thorough cleaning of all the lenses and filters to remove dust and other small obstacles from the photos was made. TIP: If a filter ring will be stuck - use a rubbery wire to hold the ring and turn then.
4. **Wire labelling** To avoid confusion during future maintenance work all wires and plug-ins were labelled for quick and easy assembly.
5. **Aperture settings** To minimize the overexposure of the sky in the vicinity of the Sun (or Moon) the aperture on fish-eye lens was changed to f.22. Previously the aperture was set to the lowest setting possible to allow as much light as possible, because the system was set to observe northern lights. The aperture on the camera is easily adjustable.



Figure 2.1: Picture of the sky above Alomar made with the All-Sky in 2018.

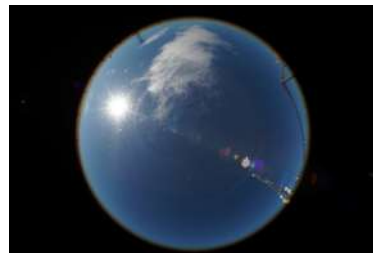


Figure 2.2: Picture of the sky above Alomar made with the All-Sky in 2020.

### 2.5.2 Unresolved issues

1. **Image sharpness** Depending on the aperture settings the depth of focus of the camera changes. Because of that it is possible to achieve sharp centre of the image, or sharp edge, but making an entirely sharp photo turned out to be difficult. During projecting the photo from the fish-eye distorted view to rectangular image the edges of the fish-eye image will be stretched so it is more important to have the best quality of the image on the edge of the circle. Still this issue remains open, because it should be possible to make entirely sharp image. The problem can be seen in Fig. 2.3.

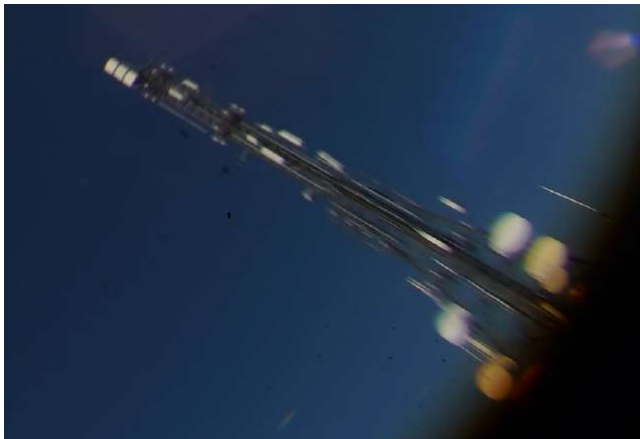


Figure 2.3: Close up of the edge of the photo made with aperture f.22 on the fish-eye lens and f.16 on the camera lens. It was the sharpest picture, but the blur is still significant close to the edge of the circle.

### 2.5.3 Set-up

#### Set-up checklist

1. Make sure all the control boards and internal wires are connected by labels on plugs and the frame. Fig.2.3 shows all the plugs that should be connected.
2. If needed clean the lenses and filters. Careful, it may lead to even more dust or scratches.
3. Mount the frame to the elevator.
4. Mount the camera as described in 2.5.3.
5. Connect the power, RS-232 and USB cables.
6. Adjust the manual settings on the camera and fish-eye lens.
7. Switch on the camera and power on the control board at the bottom of the frame.
8. If using a new computer follow the instructions in 2.8.6 to set-up the software.
9. Switch on the PC, launch uioCam and switch to manual mode.
10. Manually open the SmartMotor connection and Remote in uioCam.
11. Adjust the focus as described in 2.5.3.

## Camera mount

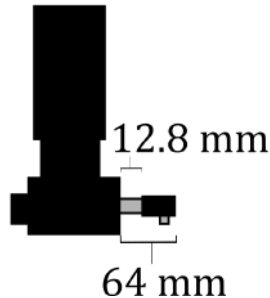


Figure 2.4: Good way to mount the camera.

12. Pull the elevator up to the dome, be careful not to cross the black marker on the steel wire.
13. All-Sky system should be ready to use in automatic mode. If you'll take a picture with default setting it will probably be underexposed/overexposed as the automatic mode in uioCam takes care of adjusting ISO and shutter speed on it's own, deleting the ill-exposed pictures.

### Mounting and maintaining

There are two slots for mounting the camera to the frame, the exact position depends on the aperture settings and focus, but the default position allowing for optimal focus is shown on the frame. As seen in Fig.2.4 the bolt holding the once set position should face downward. The mounting part is intentionally tilted in one direction by around 1 degree. The mount is adjustable, but to make sure that the camera lens is centred along the same axis as fish-eye lens the total height should be 64 mm and the shorter part should take 12.8 mm.

The easiest way to unmount and mount the camera quickly is to the screw on the other side of the frame and leaving the mounting part attached to the camera. This way we avoid adjusting it multiple times to 64 mm.

### Adjusting focus

Focus on the camera needs to be adjusted manually after mounting. Aperture number on the camera and the fish eye lens can influence the focus depth of the images (the bigger the aperture number the smaller focus depth) as well as the type of lens in the camera, mounting position and distance between the lenses. Because of that it is recommended to adjust the focus at the end of mounting, jut before pulling the system up to the dome.

Following is the mostly reliable method of adjusting the focus without help of a ladder. This assumes that all the previous step of mounting and set-up described in previous paragraphs are already done. We aim to focus the camera

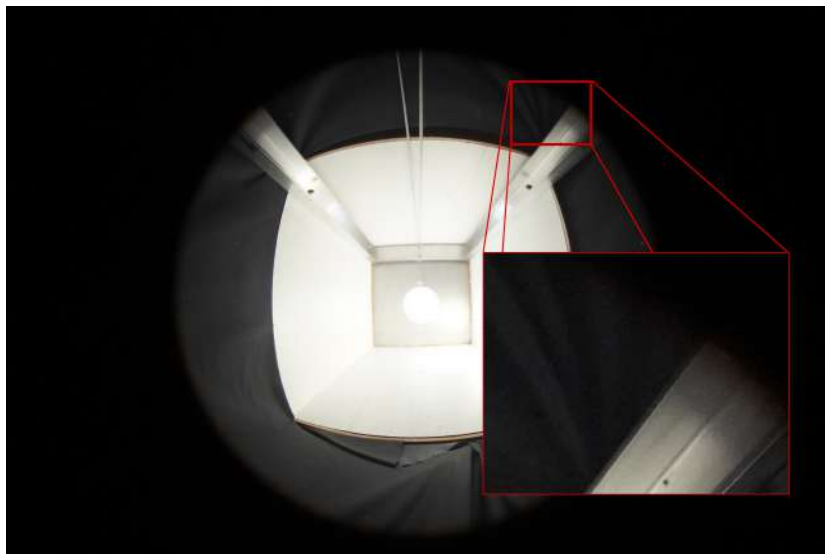


Figure 2.5: Example of a photo allowing to adjust the focus. Zoom up on a border fragment of the frame. Making sure that the steel frame is sharp all the way to the border of the photo is a quick test of the adjustment.

on the border of the all-sky photo, because during the procedure dewarping the image from the fish-eye view to rectangular the resolution of the border will decrease further while the centre of the photo will remain mostly the same:

1. Switch on the uioCam software in manual mode, open the connection to smartmotor and to Remote software.
2. Open the shutter and switch on the live view in Remote.
3. Set ISO high enough to see clearly the border of the image.
4. Close up on the border, preferably looking for distinctive features like straight lines, cables etc.
5. Using live view on Remote or camera's screen adjust the focus ring for maximum sharpness.
6. Pull up the camera to the dome, set ISO and shutter speed and the image should be at least as sharp as seen in Fig. 2.5.3



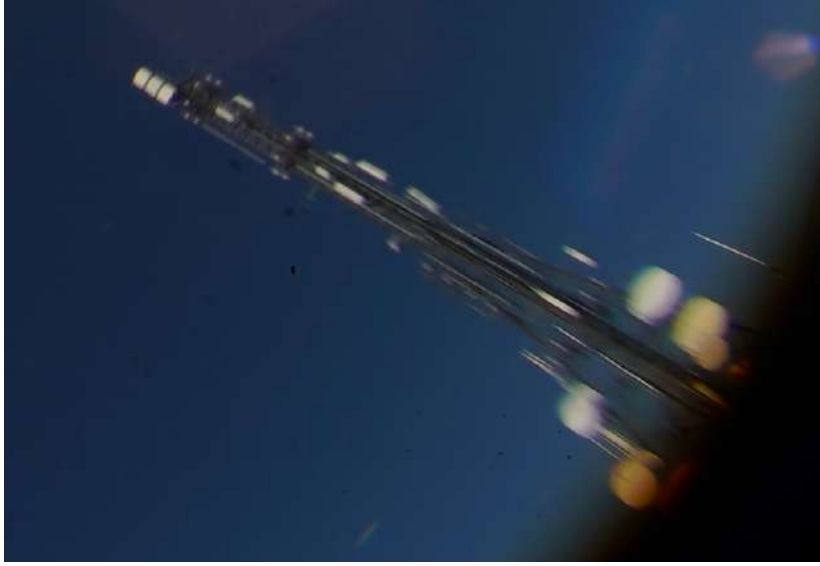


Figure 2.6: Zoom up on a mast at the border of all-sky image. Camera settings: f.16, ISO: 100, shutter speed: 1/160 s. Orange glow on the border is vignetting typical for large f. numbers.

## 2.6 Review of cloud detection options

### 2.6.1 Popular methods of cloud detection

The following is a quote from internship report written in 2018 by Elise Wright Knutsen, Marie Henriksen and Even Nordhagen:

**Theoretical background** As rays of light from the Sun moves towards us through the atmosphere, it does not travel in straight lines. They bounce off molecules, and depending on the incident angle of the rays and the size of the molecules, the light may change direction and polarization. There are two main models describing this process, is quickly summarized below.

1. **Rayleigh scattering** - The color of the sky is caused by light from the Sun being scattered off molecules in the atmosphere. This type of scattering is called Rayleigh scattering, and its intensity is wavelength dependent, see 2.1. This causes the scattering to be more effective for shorter wavelengths, hence the color we see is blue during the day and redder in the evening when the light is scattered through more particles.

$$I \propto \frac{1}{\lambda^4} \quad (2.1)$$

Rayleigh scattering can be considered elastic scattering, since the scattered

photon does not lose or gain any energy compared to the incident photon during the interaction with the molecule. Due to the small particle sizes compared to the wavelength, the scatter is close to uniformly distributed in all directions.

2. **Mie scattering** - Mie scattering differs from Rayleigh scattering in that it is almost wavelength independent. It produces the bright glare around the Sun, and the almost white light coming off mist, fog and clouds. The particles that produce Mie scatter are generally larger than the wavelength of the incident photon, which results in forward scatter dominating.

**Polarization** As white light from the Sun travels through the atmosphere to our eyes, one or many collisions may cause it to become polarized. White light can be polarized, and colored light need not display polarization at all, there is no straight forward connection between color and polarization. Only linear polarized light is considered here, as circularly polarized light is rarely seen in nature. The polarization of light around us is affected by how the light is transmitted to us, and also how things are lit. When light comes from a well defines source (as the Sun), in general it becomes more polarized than from a diffuse light source. For this reason light from an overcast sky is less polarized than light from a clear sky.

Still, different clouds yield different polarization effects as it is dependent on illumination, composition (ice/water), density and particle size. Incident polarized light on a thin water-cloud is barely changed at all, while ice-clouds have varying effect on the polarized light. Clouds with aerosols, dust-clouds and sand-clouds behave similarly to ice-clouds, though the variations tend to be less extreme.

The Sun's position in the sky is also important, as the highest degree of polarization is usually found around  $90^\circ$  from it. During and shortly after sunset the highest degree of polarization is seen around zenith.

**Ice-clouds and water-clouds** It is thought that by using a polarization filter, and rotating it while photographing the same clouds, one can differentiate between clouds of different phase (water, ice or mixed). As discussed above, ice- and water particles are thought to polarize light differently. Ice particles may come in many different shapes and sizes, and thus are harder to fit into a model. The water clouds however fit Mie scattering theory quite well, assuming spherical droplets. If the results are positive it is then possible to not only deduce cloud composition but also altitude from cloud images [6].

There is one striking difference between sunlit water-clouds and sunlit ice-clouds however. At about  $145^\circ$  from the Sun water-clouds show a sharp rise in polarization, even surpassing its value at  $90^\circ$  from the Sun. The same is not observed for any other type of cloud [7].

**Cloud detection** To distinguish clouds from clear skies, several methods have been proposed through the years. From the simplest looking only at the intensities of the red channel, evolving into fixed threshold methods for the red-blue

ratio. Later came the adaptive threshold method along with background subtraction. All these methods have both strong sides and problematic areas. Here follows our trial and review of several of them.

**Threshold methods** Threshold methods are generally divided into two categories: fixed and adaptive. The fixed methods set a limit for all images, sorting the pixels into "cloudy" or "clear". The adaptive methods change their threshold for each picture, trying to compensate for varying lighting conditions etc.

**Red channel intensity** A simple fixed threshold method for cloud detection is to exclusively look at the red color channel, where the sky would appear dark and the clouds more illuminated, and then look at the change in intensity across rows and columns in the image.

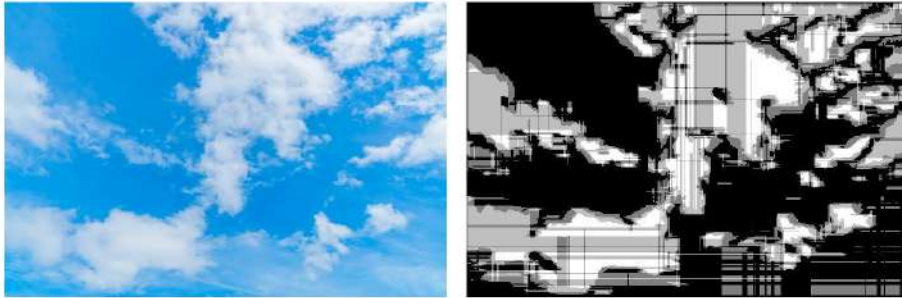


Figure 2.7: Image of the left shows the test image of a cloudy sky. On the right is the result image, using both horizontal and vertical, back and forth checks for intensity changes greater than 3.

This worked quite well for some regions, but showed serious problems in the horizon and around the Sun, as well as producing "cloud lines" in between separate clouds which were hard to eliminate, see Fig. 2.7.

**Red-blue ratio** A fixed threshold method for RBR was tried, but the results were highly variable for different images as well as having trouble near the horizon, detection the entire region as cloudy. We tried using several thresholds for different cloud opacities, see Fig. 2.8, but still the horizon remained a tricky region. Different from the clear blue of the region around zenith, the sky dome is divided into these three regions to obtain more accurate thresholds. The Sun's position is given in azimuth and elevation by UioCam every minute.

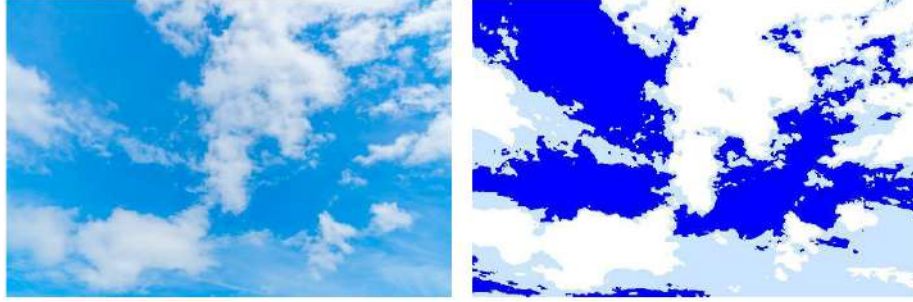


Figure 2.8: Image of the left shows the test image of a cloudy sky. On the right is the result image, using both horizontal and vertical, back and forth checks for intensity changes greater than 3.

As the color scheme of the horizon and circumsolar regions are quite different, separate regions were defined with independent thresholds for each region, see Fig. 2.9.

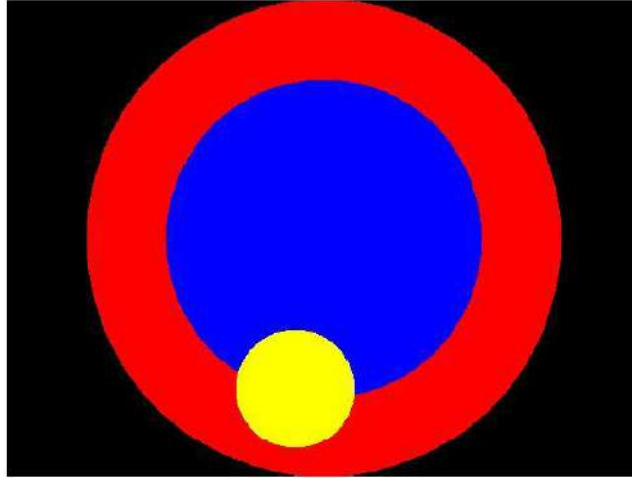


Figure 2.9: Model of how the three regions were divided. Red region is the horizon, yellow is the glare area around the Sun and blue is the large area around zenith.

While this seemed to fix some of the issues at least in the circumsolar region, it created more problems as a clear line between the regions became visible.

**Adaptive threshold** We start out with the original 24 megapixel image, the RBR is then calculated for each pixel, and a grey-scale image is produced. The standard deviation of this image is found, and compared to a threshold  $T_s$ . If  $\sigma < T_s$  the image is classified as bimodal, otherwise the image is classified as unimodal.

A histogram of the grey-scale image is produced. Unimodal images usually produce histograms with only one clearly defined peak, as all the pixels are roughly the same color. This applies for completely overcast skies as well as totally clear skies.

The location of the single peak of the unimodal image reveals if the image is of a clear sky or of overcast conditions. A threshold of  $Tf = 40$  is chosen after rigorous testing. If the peak is located below this value, the image is classified as "clear", otherwise it is set to "cloudy".

Bimodal images contain blue skies as well as clouds and therefore have several distinct colors. The histogram therefore has two or more peaks, but often for sky images there is one clearly defined peak for the blue, and an elevated elongated area for all the color tones of the clouds in the image. The standard deviation test is therefore better at checking for uni/bimodal images than looking for the number of distinct peaks [8].

A threshold for each individual image is chosen by Otsu's method, which aims to minimize the spread of pixels on each side (<http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>).



Figure 2.10: Image of the left shows the test image taken by our Sony camera. The image on the right is the result image using a standard deviation test as well as Otsu's thresholding technique.

The problem of cloud detection in the circumsolar region and close to the horizon was found now as well, as Fig. 2.10 shows. Again we tried to divide the image into the same three regions. Two different RBR's are used,  $\lambda = \frac{B}{R}$  is the regular ratio while  $\lambda_N = \frac{B-R}{B+R}$  is the normalized ratio.  $\lambda$  is less sensitive to thin clouds, but as a consequence makes fewer mistakes around the Sun and the horizon and is therefore used in these regions. The normalized ratio is used in the zenith region.

This did not seem to help much, and the borders between regions were easily spotted.

**Clear sky backgrounds** **Virtual** Instead of dividing the image further into many more regions to avoid the boundary problem of the fixed threshold method, a model of the clear sky was attempted. In a clear sky, the RBR is largest near the sun and decreases with increasing sun-pixel-angle (SPA) in the image dome.

The RBR also increases near the horizon, (large PZA) due to increased optical path length and larger aerosol concentrations near the surface.

To better distinguish clouds from the sky near the horizon, a background intensity image is generated based on the Eq. 2.2:

$$L(PZA, SPA) = (1 - \exp(\frac{-0.32}{\cos(PZA)})) \times (0.91 + 10 \cdot \exp(3SPA) + 0.45 \cdot \cos^2(SPA)) \quad (2.2)$$

To correct for increased RBR close to the Sun, an exponential decrease in intensity is set as a function of SPA [9]. The Sun's position is given by UiOCam.

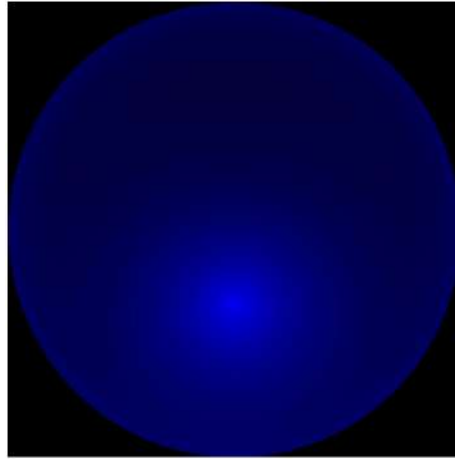


Figure 2.11: Virtual clear sky background.

The RBR for this clear-sky background (CSB) is calculated and subtracted from the original RBR of the real-sky image. A threshold is then applied to this difference:

$$\Delta_{RBR} = RBR - RBR_{CS} \quad (2.3)$$

The threshold is set at 0.15. If  $\Delta_{RBR}$  is above this value, the pixel is cloudy.

Method failed due to not being able to find a wavelength/color dependence. The equation above is supposed to be for the three RGB channels individually, but we do not know the RGB distribution across the sky.

**Real** Using a real clear sky image as base, the intensity gradient of the red channel as a function of distance to the Sun and azimuth angle is found. It is assumed that when the Sun is at zenith, the intensity gradient surrounding it is perfectly symmetrical, and as the Sun moves toward the horizon the intensity of a pixel between the Sun and zenith, at a given distance from the Sun, will remain the same. This might not always be the case, but the approximation seems fairly good.

The clear sky background image that was used for testing, Fig. 2.12, is not completely free of clouds, but has the closes solar elevation angle that was



Figure 2.12: Image used as clear sky background.

available to us. An FFT algorithm is applied to both images, and the frequency distribution in them is compared. The contrast between sky and cloud is enhanced and hopefully removing the Sun and any glare effects from the lens.

We found that even when the camera was mounted on the instrument, some movement was detected in the images which resulted in the glare and static structures in the FOV not being eliminated. If the camera is kept completely still this should be easily avoided. Another good method is to simply subtract the CSB from the cloudy image. To subtract all the color channels worked extremely poorly, so as suggested by Yang et al [10], we looked only at the green channel, and got promising results. Fig. 2.13 compares the real cloudy images to both the FFT and subtract methods. The FFT is the second row of images while the subtract method is depicted in the third row. It is clear that the FFT is superior in removing the Sun from the images, but has other obvious problems as seen in the second column. The subtracting method works quite well, but does not quite manage to remove the Sun. The static tower (at the bottom om the images) is almost completely removed however, so it is believed that if a more stable and robust camera system is set up the tower could be eliminated completely. Ultimately one should have a clear sky background (CSB) for all solar elevation angles, something we did not have time to do. Azimuth does not have to be considered thanks to symmetry. The background image can simply be rotated to fit with the original.

**Cloud types and coverage** Clouds are usually categorized by their cloud base height (CBH), the lower part of their structure. They are divided into three main regions; Low level (2 km) , Mid level (6 km) and High level clouds (13 km). In each region we find puffy white clouds (cumulus) as well as darker more even clouds (stratus) that produce precipitation. In meteorology, the amount of cloud coverage at any given location is measured in okta. Sky conditions are estimated in terms of how many eights of the sky is obscured by clouds. 0 on

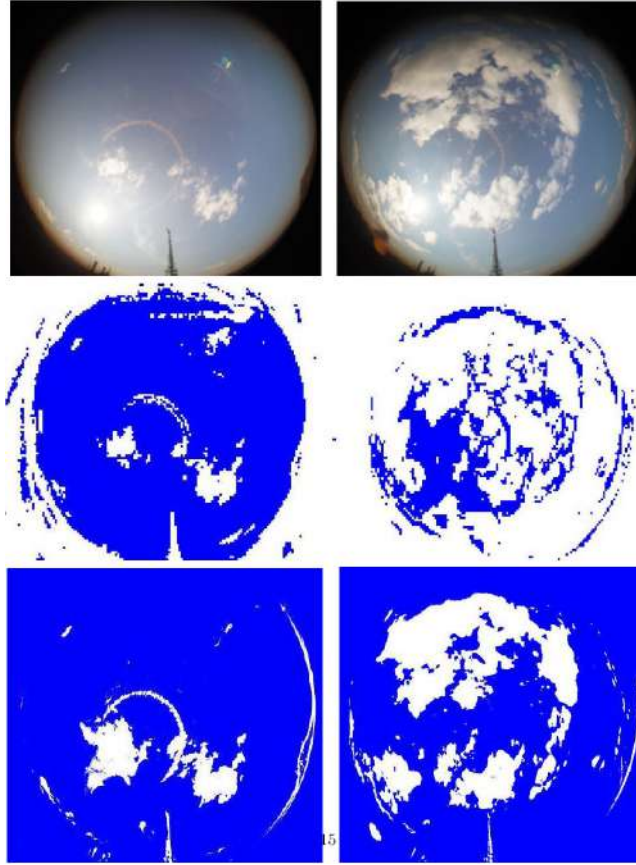


Figure 2.13: The top row are the original images taken on july 30th. On the second row the FFT algorithm has been applied to both images, and on the third row the subtraction method has been used.

the scale corresponds to completely clear skies, while 8 is completely overcast. This means that an uncertainty of at least 0.125 is to be expected, even from human observations. The scale does not account for cloud types or thickness, only extent. The number of cloudy pixels was divided by the total number of pixels and then rounded to the closest eighth to calculate coverage.

**End of quote.**

The full text of the internship report from 2018 can be found at the project github in the folder "documents". Most of the mentioned techniques for cloud detection were implemented in various python scripts that were not integrated into uioCam software. The scripts can be found on github as well. They require a bit of an update to work again properly, but may be a source of hints and ideas for future developers of the system.



### 2.6.2 Using machine learning for cloud detection

In certain cases machine learning (ML), and specifically neural networks (NN) and convolutional neural networks (CNN) may be the perfect tool to automatically classify or segment images. From the practical point of view the simplified process goes through these steps:

1. Certain dataset is chosen, for example the sky photos from All-Sky system. For successful application of ML techniques it is important that the quality, size and origin of the photos will be as similar as possible. For this the photos from All-Sky are perfect as each photo is taken with the same camera, from the same point and the content of the photo is mainly the same. Additionally it is very easy to obtain any amount of photos and collect the dataset.
2. Creating training dataset. For the machine learning algorithm to work it needs to be trained first. In order to recognize cloud coverage first we need to have a certain amount of photos that are already labelled by human with the right cloud coverage. The training dataset (labelled) is usually further separated into training, validation and test sets. The bigger the labelled training set the better the algorithm at the end. But of course that creates one of largest issues in our case as the pictures taken by All-Sky are unlabelled and would require lot of work to create the training data set.
3. Constructing the neural network. No matter the language of implementation the neural networks can have different characteristics that influence the quality of the finished algorithm. To learn more look at the references mentioned at the end of this subsection.
4. Training the neural network. Depending on the architecture of the network, size of the training data set and the difference between classifying/segmenting algorithms this step can be done during couple of hours on a typical PC or requires a large amount of CPU. Supercomputers for training the neural networks are easily available commercially (Amazon cloud, Azure, and countless others) or at universities. The training script can be tested on smaller pictures, or smaller datasets and then sent for the actual training. This step has to be done (at least in principle) only once and can be easily outsourced.
5. Using the algorithm. After training, validating and testing the algorithm we are left with a neural network with set, constant values that can quickly (in comparison to the training process) process new pictures and label/recognize/segment them.

First we need to state the difference between both classifying and segmenting as well as between NN and CNN techniques.

#### **Classifying and segmenting images**

In the case of classifying images the ML algorithm is feed certain set of images (dataset) and is trained to classify them into categories. The most common example is the MNIST database of handwritten digits. It's a huge collection of images of handwritten digits that is labelled by humans, which naturally recognize the handwritten numbers. This database is widely used as a training exercise for ML developers. Well written and trained neural network will easily classify any image of similar dimensions and quality as "1", "2", "3"... etc.

Classifying algorithms are currently used to recognize objects on pictures ("horse", "cat"...etc.), recognize faces and handwritten notes. The important detail is that classification will give for every image only a certain probability of it belonging to certain category.

In our case classifying could be used in couple of different ways:

- To classify sky images into 8 categories based on the *okta* unit used in meteorology. It would give us simple information like "cloud coverage is approximately 0%" without any details about the cloud position.
- To support thresholds methods described earlier by, for example, recognizing the right value of RBR threshold. Then the RBR analysis could sort out the pixels into "cloud" and "sky" pixels.
- To classify and vaguely (using a rectangle shape for example) point out certain predefined objects like "Sun", "mast", "obstacle"... etc.

Segmentation works differently. When the segmentation algorithm is fed specific images it can pinpoint, pixel-by-pixel, or shape-by-shape (depending on implementation) what probability each pixel has of belonging to certain category.

In our case each pixel could be attributed a certain probability of being a cloud which could provide as not only with exact position of the clouds, but also probable thickness of the cloud.

Unfortunately segmentation techniques require much larger training datasets and computation power available for training as every pixel is treated separately. This could be partially alleviated by 1) scaling down the pictures from original size 2) using convolutional neural networks instead of conventional ones 3) slicing the round image of the sky into four symmetric parts and treating each 1/4 of the picture as a separate picture.

Segmentation could provide us with much more information about the clouds - it's position on the picture, thickness and more precise estimation of cloud coverage. But at the same time labelling training datasets for segmentation requires manual painting over clouds and sky with software like Adobe Photoshop to create black and white masks labelling exactly every pixel. Fig. 2.14 shows how an All-Sky image and its mask could look like. In tries made during the internship creation of one mask took the average of 7 minutes. Considering that a reasonable amount of labelled pictures for training starts at couple of thousands it would require huge workforce to create such a dataset.

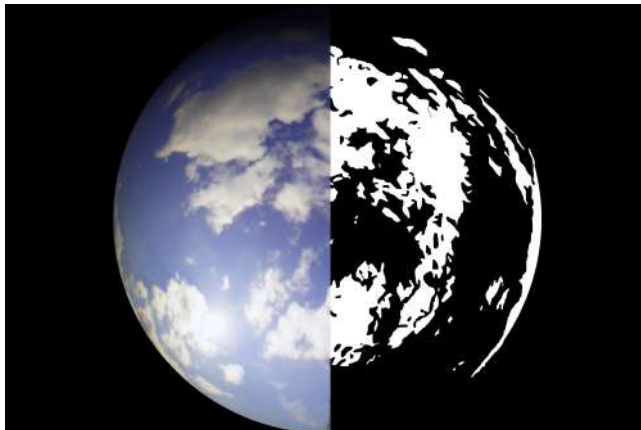


Figure 2.14: The left half of the picture is original All-Sky photo. The right side is a black and white mask made in Adobe Photoshop, labelling pixel by pixel clouds. It takes around 2 to 20 minutes to create one such mask by hand.

There are couple of possible solutions to that problem:

1. Outsourcing the labelling to professional company.
2. Automating the labelling in some way, for example by writing an application that allows to quickly input the right thresholds and then creates the black and white mask based on the threshold.
3. Finding an already ready labelled dataset like the group behind Sentinel Hub did and using this to train or retrain the algorithm.
4. Putting enough workforce and time to create a high quality data set of labelled cloud pictures and sell it later as high-quality training data or usable automatic cloud detection tool.

#### **Neural networks and convolutional neural networks**

In very short words the NN and CNN are two (out of dozens) different techniques of machine learning. Without getting into detail of how neural network work it is enough to say that CNN are adding an additional step to the process which is convolution of the images, before training. Convolution is a technique well known and used in image processing and recognition allowing to scale down the amount of data required for processing without losing information about the picture.

#### **Resources**

This was barely a few comments on practical use of machine learning for cloud detection with the All-Sky system. To learn more about machine learning, neural networks, convolution and other teams attempting to use ML to detect clouds refer to those pages:

<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>  
<https://towardsdatascience.com/a-keras-pipeline-for-image-segmentation-part-1-6515a421>  
<https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python>  
<https://machinelearningmastery.com/save-load-keras-deep-learning-models/>  
<https://medium.com/sentinel-hub/sentinel-hub-cloud-detector-s2cloudless-a67d263d3025>  
<https://www.fast.ai/2017/11/13/validation-sets/> <https://developers.google.com/machine-learning/data-prep/construct/collect/data-size-quality>  
<https://towardsdatascience.com/how-to-build-a-data-set-for-your-machine-learning-proj>

### 2.6.3 Post-processing RAW images with C++ and Python

Sony camera in All-Sky system allows us to save pictures in two possible formats: JPEG and RAW, or both at the same time.

RAW pictures, in this case having the extension of .arw, are the digital analog of unprocessed film. It is uncompressed data from the camera matrix and as such retains the most complete data from each photo. But this format is impossible to view or process with typical image packages and software before it goes through the process of demosaicking.

The issue with RAW files is their significant size (around 40 Mb per photo) and long processing time. To lessen this problem in every case were possible we used JPEG photos for some of the processing and intended to in the end save the accepted pictures in a lossless, compressed format of either PNG or loseless JPEG.

More on RAW pictures and the process of demosaicking can be found here:

<https://www.odelama.com/photo/Developing-a-RAW-Photo-by-hand/> <https://www.odelama.com/photo/Developing-a-RAW-Photo-by-hand/Developing-a-RAW-Photo-by-hand-Part-2/>

Various libraries in C++ and Python allow bigger amount of control on the demosaicking process, in our case we used rawpy, which is a python plug-in for a popular C++ library LibRaw.

<https://www.libraw.org/docs> <https://pypi.org/project/rawpy/>

RawDigger software allowed us to look into the RAW pictures before developing them: <https://www.rawdigger.com/download>.

### Extracting the data from underexposed photos

While implementing the ICE to work on RAW files instead of compressed JPG's it turned out that the post-processing of 14 bit RAW files makes it possible to retrieve images of good quality from highly underexposed photos. Using the brightness adjustment from *postprocess* function of the pyRaw library it was possible to adjust photo in Fig.2.6.3 to look like in Fig.2.6.3.

That presented new possibilities going far beyond using ICE to check the exposure of the last image. It occurred to us that if separate analysis of the Sun region and the rest of the image would be possible we could have well exposed photos of the sky, with Sun taking less space than in Fig.2.6.3. Couple of approaches were tried to achieve that:



Figure 2.15: Picture of the sky made with greyscale filter. ISO: 50, shutter speed: 1/500 s.



Figure 2.16: The same picture after post-processing with brightness adjustment.

- **Treating Sun as bad pixels.** RawPy library has in-built function for detecting and enhancing bad pixels before the post-processing. This function allows to either detect the bad pixels based on the list of photos or user defined coordinates. Using the coordinates for the Sun position from uioCam we tried to minimize the effect of the Sun on the later post-processing. This failed because bad pixels by definition (and the way they are treated in the function) are no larger than two pixels next to each other.
- **Masking the Sun.** Trying to change the values of pixels around the Sun to influence the post-processing brought no effect. The idea was that after removing the brightest pixels the picture will be adjusted with different contrast showing the sky below the overexposed glow around the Sun. If the mask was large enough to mask the whole area that Sun was overexposing the brightness adjustment flipped to whole picture into black. Trying to change the colour of mask did not change the results. The tries can be seen in Fig.??.
- **Log function of the RAW values.** In hope to lower the brightness of the overexposed parts and lighten up the dark part of the picture we tried to treat the RAW values with different log functions (natural log, log with base 2 and 10). After the log function the pixel values were "stretched" once again to the whole range of 14 bits (0 to 16 383). Both changing the values of the whole picture, and only of a square surrounding the Sun were attempted. The effect, not bringing much use, can be seen in Fig.2.6.3.
- **Subtracting the brightest values.** Another attempt was to take the RAW values around the Sun and subtract a constant value from them and later stretch the values back to the full range in hopes of magnifying the small differences in the intensity. This as well failed to improve the picture as seen in Fig.2.6.3.
- **Post-processing the Sun and the sky separately.** If the image could be separated into two arrays and treated with brightness adjustment on

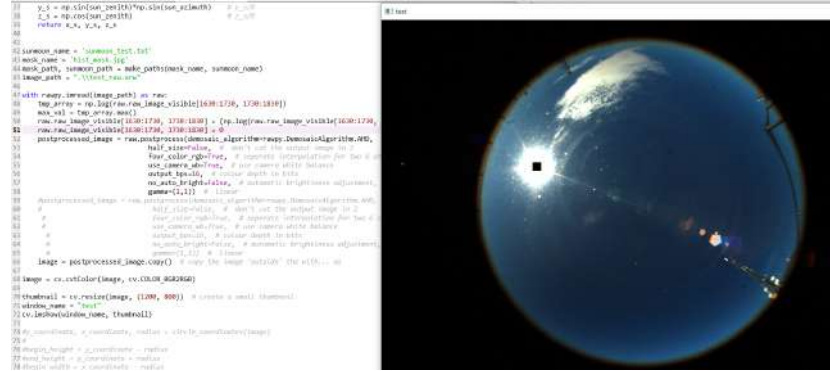


Figure 2.17: Attempt to mask the Sun in the RAW file before post-processing. The mask does not affect how the overexposed area is processed.

it's own possibly we could see a bit more of the sky behind the Sun. Unfortunately the implementation of the rawPy library makes it impossible to cut the RAW image into two objects. It allows to see and manipulate the values inside the RAW object, but once it is copied into the numpy array it can not be post-processed any more with the *postprocess* function. That could possibly be solved with a different library or altogether a different approach to handling RAW files. Considering other available methods it was considered too time consuming to follow up on it at the time.

For now none of those approaches give us the results we wanted. One problem maybe that even if the RAW picture contains a wider range of intensity than visible on the screen the relative difference between the overexposed area and the sky may be simply too big and any try to brighten the sky around the Sun will result in overexposed white. As such a mechanical blockade of the Sun seems to be the only reasonable solution. In any case, even without the blockade the area that we loose to the overexposure from the Sun is around 500x500 pixels (in the worst case scenario). That accounts for no more than 2% of the image. Moreover it is relevant only for the days when the Sun is not covered by clouds.

## 2.7 Image distortion and projection onto a map

In 2020 no important improvement or ideas regarding un-distorting the photos or projecting them onto a map were made, information on attempts to correct the lens distortion from 2018 can be found in internship report written by Elise Wright Knutsen, Marie Henriksen and Even Nordhagen.



Figure 2.18: After treating the RAW values with logarithmic function and stretching it back to the full range of intensity the picture not only takes on the pinkish hue, but the contrasts between the sky, clouds, Sun and border are too small to be useful in cloud detection.

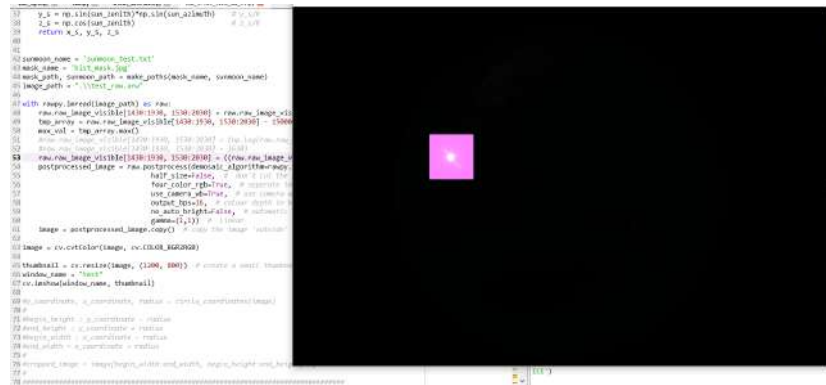


Figure 2.19: Covering the whole overexposed area with mask or using it to subtract the values leads to the post-processing function flattening down the rest of the picture to black. Even if that was to be avoided the area around the Sun still does not picture well contrasted Sun and the sky.

## 2.8 Software

### 2.8.1 uioCam v2.1

The original uioCam software for controlling the All-Sky system was adjusted from northern light observation to cloud detection during an internship in 2018. The description of changes can be found in 'And\_ya\_Space\_Center\_Journal' as well as some suggestion towards further improvements.

uioCam v2.1 implements some of those proposed changes and this is the list of:

#### New functionality

- Accessing information on the last photo. This allows to quickly check the manual settings even if the camera is up in the dome.
- ICE: automatic intensity control. This feature automatically adjusts shutter speed and ISO to current lighting without being as unpredictable as AUTO program. It was tested only over couple of hours and includes some known bugs that are described in more detail in Sec. 2.8.2.

#### Still left to improve

- Safeguards! The modes and intensity control should be tested properly and catch exceptions for funny scenarios so the program doesn't stop running if anything happens in the middle of an experiment.
- Proper testing of the whole system.
- Post processing scripts needs writing, testing and implementing in uioCam. It could involve improving image quality, detecting clouds, dewarping the fish-eye distortion, projecting on a map, compressing the images for archive, sending them to a cloud etc.

#### DONE

- When in manual the uioCam will from now on open the shutter when you press "get picture".
- Added new .cpp: photosIo.cpp that will include all functions dealing with accessing the already made images, for now it will be picking up metadata from the last picture.
- Found a working way to compress the .arw file without loss: lossless JPEG that actually works through Magick++ and gives me around 50% of the initial file size - but it is not currently used, as all image processing is done through ICE.



- Detect the circle of the photo automatically based on the image, crop the image to square.
- Old functions and dependencies responsible for compression of photos to PNG were deleted from uioCam v2.1. That included "pngIo.cpp", and all zlib files. Post-processing and compression will be handled in not yet existing post processing script.
- All references to keeping the last image in a bufor in uioCam were deleted. Image bufor had reason to exist only before Remote was used, and thanks to that uioCam should take around 50 mb less of computer resources during running.
- Polarisation and Night mode were commented in uioCam. They weren't finished two years ago, and consisted mostly of dummy functions. Instead there will only be Normal and Cloud Detection mode.

#### **NOT DONE for a reason**

- Safeguard if camera is switched on - the Remote won't open if the camera isn't connected/switched on and gives off a clear error message, connecting it through uioCam would take a lot of time and gave us nothing new.
- Add changing the white balance - Remote doesn't have any shortcuts controlling the white balance which makes it difficult to adjust it, but the more I read about WB the more important it seems to have it under control. We want to look at the ratio between the colours and that's exactly what WB is changing. RAW pictures allow us to change the WB later on, but I think that while shooting we should shoot it either in one set (clear skies for example) or in our own default without any correction at all, but not the AUTO WB.

#### **New setup of uioCam**

Considering the changes made to uioCam functionality proper changes in the window layout were made as well. The new setup can be seen in Fig.2.8.1. Radio buttons responsible for Polarisation and Night mode were removed, as those modes are no longer in use. Additional button for accessing the metadata on last photo was created. It is important to add that the metadata allows us to see if the camera was set in the MANUAL or AUTO mode. Although those modes will always have to be changed manually on the camera body it allows the remote user to at least identify the problem and ask someone at Alomar for help. Additional radio button for post-processing options was created as well. It allows to control if the photos should be automatically post-processed by PINE script if and when such a script will be written.

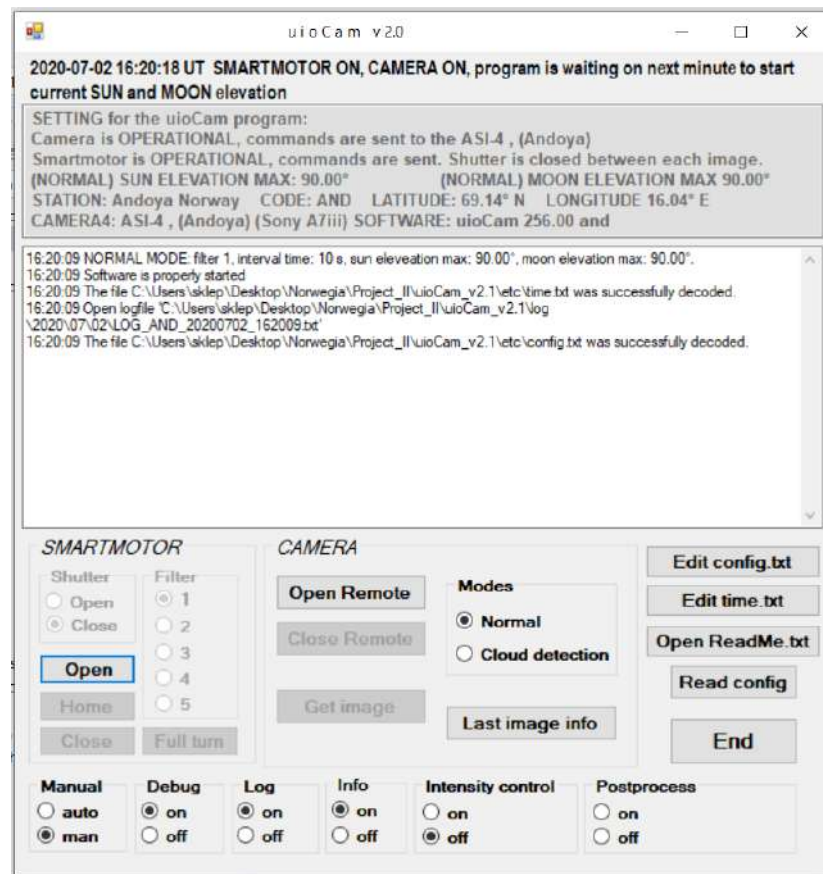


Figure 2.20: How the new application window looks for uioCam v 2.1.

Table 2.5: Summary of uioCam modes.

Mode	Interval	Filter	Day/Night	Exposure control	Post-process	Camera program
Normal	600 s	1 (no filter)	No difference	Off	Off	AUTO
Cloud detection	600 s	1 (no filter)	Different ISO	On	On	MANUAL

### New modes in uioCam

In version 2.1 there are only two modes available in uioCam - Normal mode and Cloud Detection mode.

**Normal mode** offers the skeletal functionality of uioCam. By default it takes all-sky images with a default interval time of 600 s, without any filter, post-processing or exposure control. It saves all of the pictures made without checking the quality of them. For efficient work with this mode the AUTO setting on the camera is recommended.

**Cloud Detection mode** automatically runs the whole cloud detection routine. The camera needs to be in MANUAL setting for this mode to work. By default the Cloud Detection checks if it is daytime or night-time and adjusts ISO and filters accordingly. Afterwards it takes a photo in the set interval, checks the exposure of the photo, adjusts the shutter speed setting, if needed, and saves only the well-exposed photos.

If PINE or similar script would be written then all well-exposed photos would be automatically post-processed using a script and then compressed photos, cloud masks and projections onto a map would be saved.

Both modes can be modified using the "config.txt" file.

The default settings for the mode are shown in Tab.2.5

### 2.8.2 ICE

ICE - Intensity Check Executive is an application made from python script into the .exe file that will run the necessary intensity check for the uioCam software and handle demosaicking and compression of RAW pictures.

The .exe file was made using auto-py-to-exe package, more here:

<https://dev.to/eshleron/how-to-convert-py-to-exe-step-by-step-guide-3cfi>

### Automatic detection of image area.

The all-sky images are always circles, leaving in the corners black, unused borders that falsify the brightness histogram. The problem is that the radius and the position of the circle may vary depending on the position in which the camera was mounted. To mask the dark borders the ICE script automatically detects the circle of the photo area. First we blur (with gaussian blurring) the photo to minimize the chance that any circular clouds or reflections will be detected as circles. Then we apply binary and Otsu thresholding to make sure that nothing inside the image circle will influence the detection. Lastly we use

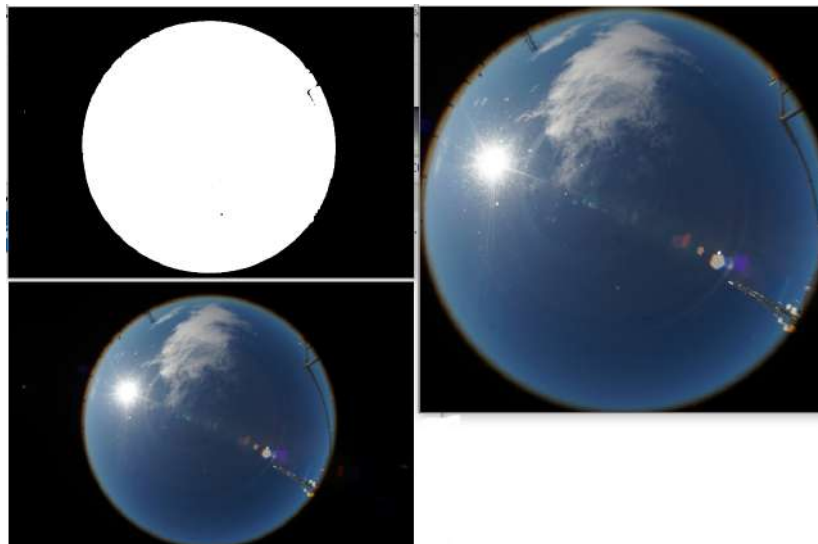


Figure 2.21: Left bottom is the original image, left top is the photo after thresholding. On the right the original image is cropped in the way to fit the detected circle.

Hough transform to detect the circles with parameters making sure that we will only find one, large circle that could be the photo area or none at all instead of many circles possibly visible on the photo.

For cases in which no circle can be detected (underexposed photos for example) a warning is issued and the script continues by assuming a default radius and position of the circles.

The method requires further testing, especially during the night, but for now it was found that it works well on the all-sky photos. In case of photos containing completely unrelated images (test photos of desk, wall, etc., without the fish-eye lens at all, see Fig.2.8.2) the script tends to give false positives and assume the circle in the middle part of the photo. While it isn't ideal this doesn't crash the script and I leave that like it is in sincere hope that future users will know the difference between an all-sky image and a random photo. An example can be seen in Fig.2.8.2

### **Automatic detection of Remote ISO and shutter speed settings.**

To make the automatic intensity control user-proof and easy to use we automated the detection of current ISO and shutter speed settings in Remote. The main issue is the fact that currently uioCam communicates with Remote only by activating Remote window and sending keyboard shortcuts to the active window. This makes it impossible to read the image settings directly from Remote or send a command that would say "set ISO to 50". The only thing that



Figure 2.22: On the left there is a test photo made without fish-eye lens. Right top shows the result of the thresholding, this is the image that is fed into Hough circle detection and on the right bottom we can see the image cropped to fit the circle detected by the Hough transform. There is no visible circle that could be detected and the script did not show a warning to indicate that it will use default circle coordinates. The script detected the circle were there was none.

uioCam is currently capable of simply sending shortcuts that mean "change ISO/shutter speed setting one value up/down". At the same time Remote works independently from the uioCam so whatever changes the user will put into the Remote using mouse, keyboard or just changing the settings on the camera itself will not affect the uioCam in any way.

To prevent many bugs steaming from this issue and to make sure that the functioning of the software will not be dependent on the user remembering to put in a set of default values into uioCam a routine to read settings from Remote and store them in uioCam was implemented.

This routine works in function *updateRemotestat()*. It acquires one image (with the mechanical shutter closed) from Remote, waits for the Remote to save it, accesses the metadata of the photo and picks up the necessary values from the exiv data. Then it compares the value to an array of strings defined in uioCam that mirrors most of the settings available in Remote. The routine switches on every time when the intensity control is switched ON in any way. This way we make sure that ICE always knows what the camera settings are currently and can interpret it correctly.

After the update the newly made photo is removed from the disk to save space.

This routine has couple of issues, some that can be possibly solved in the future, and some that are just an artefact of communicating with Remote in such a round about way.

- The settings available in Remote as AUTO (automatic ISO) and BULB (shutter speed controlled by trigger) will show up in exiv data as the actual values that were used to make the photo. Then *updateRemotestat()* will

read them as concrete values introducing chaos, because in Remote the AUTO and BULB settings are simply the last available options. If AUTO value for ISO will lead to the Remote shooting a photo with ISO equal to 100 `updateRemotestat()` will pick it up and set the value for ISO as 3. If `MakeBrighter()` will be called it will switch the ISO value to 4, expecting that it will equal to ISO 125 in Remote. Instead Remote will switch to another available value after AUTO which is ISO 50. Without changing the whole idea behind communicating with Remote it is impossible to prevent the user from setting AUTO or BULB in Remote or on the camera and introducing bugs into the program. **WARNING FOR USERS:** Don't set Remote or the camera into ISO AUTO and shutter speed BULB ever. It will make the program function in random ways.

- High ISO values: exiv standard does not support ISO higher than 64000. Any ISO value higher than that will show up in exiv data as ISO 65535. As such using this function it is impossible to correctly recognize such ISO values (available in Remote) as 80 000, 102 400, 128 000, 160 000 and 204 800. Fortunately this problem may be solved (isn't solved yet): every time that `updateRemotestat()` will read 65535 it can change ISO as long as it will reach recognisable value of ISO 64 000. If it needs to change the value to the higher ISO it just needs to keep track of how many times `makeBrighter()` was called. Still, this issue may be left to solve for the next lucky person to end up with this project considering that having ISO higher than 64 000 is a rarely needed luxury.
- Too long exposure time: `updateRemotestat()` does not actually wait for the Remote to finish acquiring the photo as it does not have any internal knowledge about anything that Remote does. Currently the wait time is set for 2 s and if the shutter speed is set higher than 1 s or so the function will pick up the previous picture, instead of the right one. Waiting for the maximum time of 30 second would be utterly ridiculous and such long exposure time will probably not be used considering that it would make clouds and stars blur with the movement. Outside of writing proper warnings in the manual and text window of `uioCam` I see no viable way to prevent this issue.

**Issue with intensityOff/On:** For unknown reasons when radio button intensity control On is clicked `uioCam` runs first anything inside `intensityOff_CheckedChanged` and then anything that is inside `intensityOn_CheckedChanged`. When radio button Off is clicked similar issue appears: first `uioCam` runs whatever is in `intensityOff` and then what is in `intensityOn`. To make sure that `updateRemotstat` is run only when needed `intensity_control_helper_troll` is used. Don't ask, just leave a pot of cream for helping trolls outside. It works.

### 2.8.3 Sun blockade

During clear and partially cloudy days the Sun is the biggest obstacle to cloud detection. Because of its brightness it is overexposing a large portion of the picture introducing white pixels into it and confusing all functions based on brightness histograms (used both to adjust the exposure time and post-process the pictures). To deal with this problem a simple, mechanical sun blockade was proposed.

It would be a thin, metal arm, extending in an arch above the fish-eye lense and moving around the lens following the Sun. This movement would have to be synchronized with Sun/Moon tracking available in uioCam or controlled some other way.

### 2.8.4 Post-processing software

My intention was to create a python script called PINE and implement it in uioCam the same way as ICE script - as an executable file that would pick up photos from predefined folders. This script would pick up each photo with good exposure, enhance its quality, crop it if needed, detected the clouds, projected the clouds onto a map and dewarped the fish-eye lens image. Then it would save all in compressed formats.

Depending on the functionality and developer skills in python and image processing I estimate that finishing such script and testing it would be a good 8-12 week project.

### 2.8.5 For users

#### Switching on uioCam

1. Make sure that all the hardware is set up, switched on and connected.
2. If in doubt make sure that all the paths in uioCam code and ICE script are up to date.
3. Make sure that Remote is set-up properly, and that the path for saving files is correct (should lead to folder last in uioCam folder).
4. Switch on uioCam, now you have two options under "Manual":

auto - uioCam will open the connection to Smartmotor, open up Remote and read whatever is set in file "config.txt". If "config.txt" says so uioCam will start making photos in appropriate intervals with mode/functions set in "config.txt". This is default mode for conducting measurements.

man - uioCam will stop doing whatever is written in "config.txt". Connection to Smartmotor, Remote and everything else needs to be opened up by hand. This is a default mode for testing. NOTE: uioCam always waits for a new minute to start twice before everything runs smoothly. This is especially important in auto mode which will actually start to run

properly after around 2 minutes. NOTE II: Opening up Remote, choosing a camera from the list and sending commands to Remote works through uioCam by sending virtual keyboard shortcuts to Remote. So the windows will become active on its own, and if that will be interrupted with user's command from a mouse, for example, the set-up of uioCam may not be complete. NOTE III: Note II goes also for the use of ICE functionality. It takes a moment for uioCam to send all necessary keyboard shortcuts to Remote and activating different windows at the time will disrupt it.

### uioCam functions

- Smartmotor functions - those control the Smartmotor connected to All-Sky system

Open button - will try to open and diagnose connection to the Smartmotor. If the message says "Timeout" it means that the program ran into a problem and can not connect with Smartmotor. Check the cable connection and make sure that Smartmotor is switched On.

Home button - turns the Filter wheel to position 1.

Close button - closes connection to Smartmotor.

Shutter Open/Close - allows to open or close the mechanical shutter. NOTE: If opened in manual mode the shutter won't close until the button Close will be clicked. The shutter may overheat if it stays open too long so for tasks requiring shutter open for a long time we recommend to block the shutter with a nail or other physical obstacle. NOTE II: The camera is quite sensitive to light and doesn't have its own mechanical shutter. If possible do not keep the shutter open for a long time when the camera is up in the observation dome.

Filter wheel - allows to turn the filter wheel to a specific filter or make a whole turn. The filters are described in numbers in the inside of the filter wheel.

- Camera functions

Open/Close Remote - opens Remote or activates its windows if the Remote is already switched on, chooses the first camera from the list and closes Remote program when needed. In auto mode those things happen automatically.

Get image - acquires one image if the intensity control is switched off or acquires one image with a right exposure if the intensity control is switched on (which may mean a lot of pictures shot and discarded).

Last image info - displays information on the last photo, taken directly from the exiv data from the picture, so it will always reflect the actual settings with which the photo was made, no matter what is set in uioCam or Remote. NOTE: saving RAW photos takes a while so sometimes "Last" photo may be in fact "One before last".



Modes - allows to switch between Normal and Cloud modes. Modes are described in Sec.2.8.1. Using radio buttons at the bottom of the interface, switching the modes here or changing settings in "config.txt" are interchangeable and mean basically the same. It may create some chaos, if for example someone would change Normal mode settings in "config.txt" to have all characteristics of Cloud mode. Then if radio button will be switched to Normal mode the program will run with whatever is written in the configuration file anyway.

- Radio buttons

Debug, Log and Info radio buttons - those control which log text files should be produced during running uioCam.

Intensity control - controls if uioCam should adjust automatically ISO and shutter speed until the photos have the right exposure. For this to work the camera needs to be in Manual program. Every time the intensity control is switched on in any way (Radio button, config.txt, mode radio button) uioCam will make a photo with a shutter closed to synchronize its settings with those currently in Remote. More information can be found in Sec.2.8.2.

Postprocessing - that one is for show. Or rather for functionality that uioCam v2.1 does not possess.

- Edit files - those buttons open up writable .txt files in which most of the changeable settings of uioCam reside.

Edit config.txt - this opens the most comprehensive configuration file. It controls the time of intervals between photos, the characteristic of the modes and all button settings. Essentially changing "config.txt" may completely replace using buttons on the interface. The changes are loaded into uioCam only after config.txt is saved and "Read config" button is clicked or uioCam is restarted.

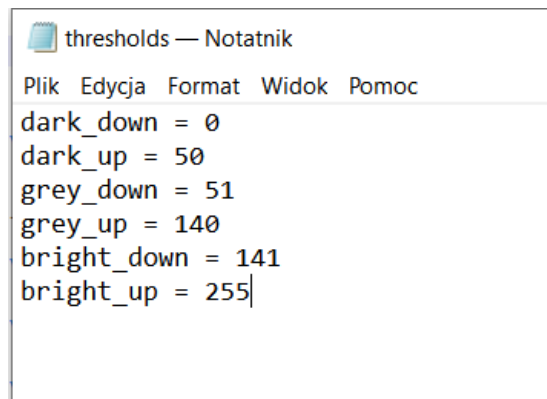
Edit time.txt - time.txt sets limits for when the picture taking may take place. It's useful if for whatever reason during couple of weeks of measurements we don't want any pictures from specific period.

Open ReadMe.txt - just open ReadMe file.

Read config - loads config.txt anew into uioCam, all options are updated.

- End - Closes Remote, connection to Smartmotor and uioCam software.

NOTE: There is one more .txt file that can be adjusted, but in most cases shouldn't be. It's "thresholds.txt" that can be found in etc folder, next to config.txt. This is a file from which ICE takes its values for thresholds on brightness histogram. It has format like in Fig. 2.8.5. All relevant pixels in a picture are sorted into "dark", "grey" and "bright" categories when ICE is running and



```
thresholds — Notatnik
Plik  Edycja  Format  Widok  Pomoc
dark_down = 0
dark_up = 50
grey_down = 51
grey_up = 140
bright_down = 141
bright_up = 255
```

Figure 2.23: Look at thresholds.txt, file providing histogram thresholds for ICE script.

the ratio of numbers of those pixels define if the picture is labelled as "underexposed", "good exposure" or "overexposed". So in the case when, for example, accepted pictures seem to bright you may try to adjust the bright thresholds and ICE will sort more photos into "overexposed" category. I recommend copying the original file and testing new ideas carefully, because the current thresholds are the results of long testing.

END NOTE: In general uioCam comes from simpler times and may require patience on your side, dear user. It was not tested properly so unintended stops and breaks in the middle of measurements are certainly a possibility.

### 2.8.6 For developers

#### Moving to new computer

**Operating system (OS)** The old system used Windows XP (32-bit). The main program, uioCam, was written in Microsoft Visual C 2010 on Windows 7. The new software for remote control of the Sony camera needs a 64-bit OS. A laptop with Windows 10 (64-bit) is therefore chosen as the new working station, but this caused problems with the old uioCam software.

**Remote** For uioCam v2.1 to work Imaging Edge from Sony needs to be installed at the new computer. We mostly need only the Remote application, but it comes as a package with a Viewer and Editor.

Once Remote and uioCam is installed make sure that White Balance is set for "clear skies", image size is "L", quality is "fine", format is "RAW + JPEG" and that Remote's folder for saving pictures leads to .../uioCam/pictures/last.

Quality and image size does not matter that much, but the wrong path will lead to uioCam not working properly. White Balance should be set for one specific setting as to avoid channel ratio adjustments by Remote.

**Moving uioCam to a new computer** To run the old program on a new computer with updated OS, the easiest way to get it going is to download Visual Studio and create a new project. Visual Studio 2017 was downloaded to the student laptop with Win10, and uioCam was recompiled. After many errors and fixes the program actually compiled successfully. And this is how to do it (in VS 2017):

1. Choose New → Project. Choose Installed → Visual C++ → CLR → CLR Empty Project (if this does not exist, download more stash from Visual Studio, make sure "C++/CLI support" is installed).
2. Right click on the project, choose Add → New Item. Choose Visual C++ → C++ File (.cpp). Add a .cpp file for the main function, named something like "uioCam.cpp" (use the name of the project you just made).
3. Right click on the project, choose Add → New Item. Choose Visual C++ → UI → Windows Form. (Let it be named MyForm.h, it makes life easier later on).
4. Copy all files (.cpp and .h) from the original folder of the project to the folder of your project.
5. Right click on "Header Files" and choose Add ! Existing Item and select all the .h files in your project folder, and include them. Repeat for the .cpp files and the "Source Files" folder.
6. Copy whatever is in the original main file (uioCam.cpp or similar) over to the new uioCam.cpp (or similar) file in your project.
7. Copy whatever is in the original MyForm.h file over to the new MyForm.h file. Make sure you have the MyForm.cpp file as well.
8. Make sure that under project Properties → Linker → Input → Additional Dependencies you have user32.lib
9. Choose Project → Properties → Linker. Then go to System and change Subsystem to "Windows (/SUBSYSTEM:WINDOWS)", and go to Advanced and change Entry point to "main". Note: this may give an error in MyForm.h [Design] (to fix: undo this step, restart VS and reopen MyForm.h [Design]). Doing this step prevents the terminal window to pop up when the program is started. So if further development is needed, and the terminal is wanted, this step can wait. But for the final user, the terminal window will be of no use, so it's better to just remove it. If the terminal window is closed, the application will also be closed, which could be impractical during an experiment.

And that should be it, hopefully it will compile on first try. Also, remember to create and copy the config.txt, time.txt, intensity.txt, thresholds.txt and sunmoon.txt in the etc folder, and update all paths in the define.h file. Make

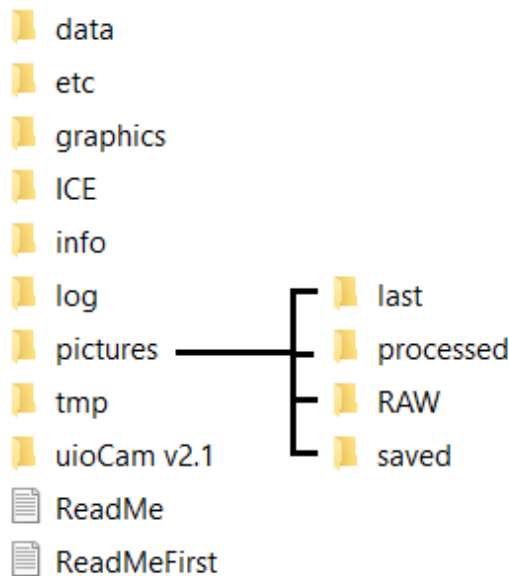


Figure 2.24: Set-up of folders for uioCam v2.1. Different set-up is of course possible, but make sure to update all paths in uioCam and ICE code.

sure that imagepath is correct, and that all folders from the original set-up exist. Some will be created automatically if they don't, but some may create problems.

### The folder set-up on the student laptop

**Modifying or moving ICE** If any changes need to be done in code of ICE script the new version of the script has to be made into an .exe file by using auto-py-to-exe package and then the .exe file has to be exchanged in uioCam folder. Make sure that all the paths in ICE are still relevant for specific computer and usage.

### 2.8.7 End note for future developers

The software controlling All-Sky system is a frankenstein's monster. It mostly works, but the most of it issues stems from the fact that it combines C++, Python, .NET, Visual C, application produced by Sony (probably Java) and bits of unidentified lines controlling Smartmotor. The various parts communicate with each other not via normally expected canals but by picking up info from text and image files, sending keyboard shortcuts to each other and probably black magic as well.

Consulting others and knowing uioCam far too closely after couple of months of work I recommend for future developer to consider starting from scratch instead of revamping once again uioCam.

## 2.9 Appendix I: Codes, older documents and scripts

The full archive of All-Sky project can be found on a public github account. In case of any troubles or questions contact: joanna.szulc.pl@gmail.com.

<https://github.com/Joanna-Szulc/Cloud-detection-ASC>



# Bibliography

- [1] Stefan Hagemann, Lennart Bengtsson, and Gerd Gendt. On the determination of atmospheric water vapor from gps measurements. *Journal of Geophysical Research: Atmospheres*, 108(D21), 2003.
- [2] B. Paláncz J.L. Awange. *Geospatial Algebraic Computations*.
- [3] É. Borbás. Derivation of precipitable water from gps data: an application to meteorology. *Physics and Chemistry of the Earth*, 23(1):87 – 90, 1998.
- [4] Davis E. S. Duncan C. B. Hajj G. A. Hardy K. R. Kursinski E. R. Meehan T. K. Young L. E. Yunck T. P. Melbourne, W. G. The application of space-borne gps to atmospheric limb sounding and global change monitoring.
- [5] T. Ning, J. Wang, G. Elgered, G. Dick, J. Wickert, M. Bradke, M. Sommer, R. Querel, and D. Smale. The uncertainty of the atmospheric integrated water vapour estimated from gnss observations. *Atmospheric Measurement Techniques*, 9(1):79–92, 2016.
- [6] X-W Shi X-M Sun and Y-P Han. Reflection of polarized light in ice-water mixed clouds. *Journal of electromagnetic waves and applications*, 20(12), 2006.
- [7] G. P. Konnen and H. G. Begbie. *Polarized Light in Nature*. Cambridge University Press.
- [8] Weitao Lu Qingyong Li and Jun Yang. A hybrid thresholding algorithm for cloud detection on ground-based color images. *Journal of atmospheric and oceanic technology*, 2011.
- [9] R. Chauvin, J. Nou, S. Thil, A. Traoré, and S. Grieu. Cloud detection methodology based on a sky-imaging system. *Energy Procedia*, 69:1970 – 1980, 2015. International Conference on Concentrating Solar Power and Chemical Energy Systems, SolarPACES 2014.
- [10] J. Yang, Q. Min, W. Lu, Y. Ma, W. Yao, T. Lu, J. Du, and G. Liu. A total sky cloud detection method using real clear sky background. *Atmospheric Measurement Techniques*, 9(2):587–597, 2016.