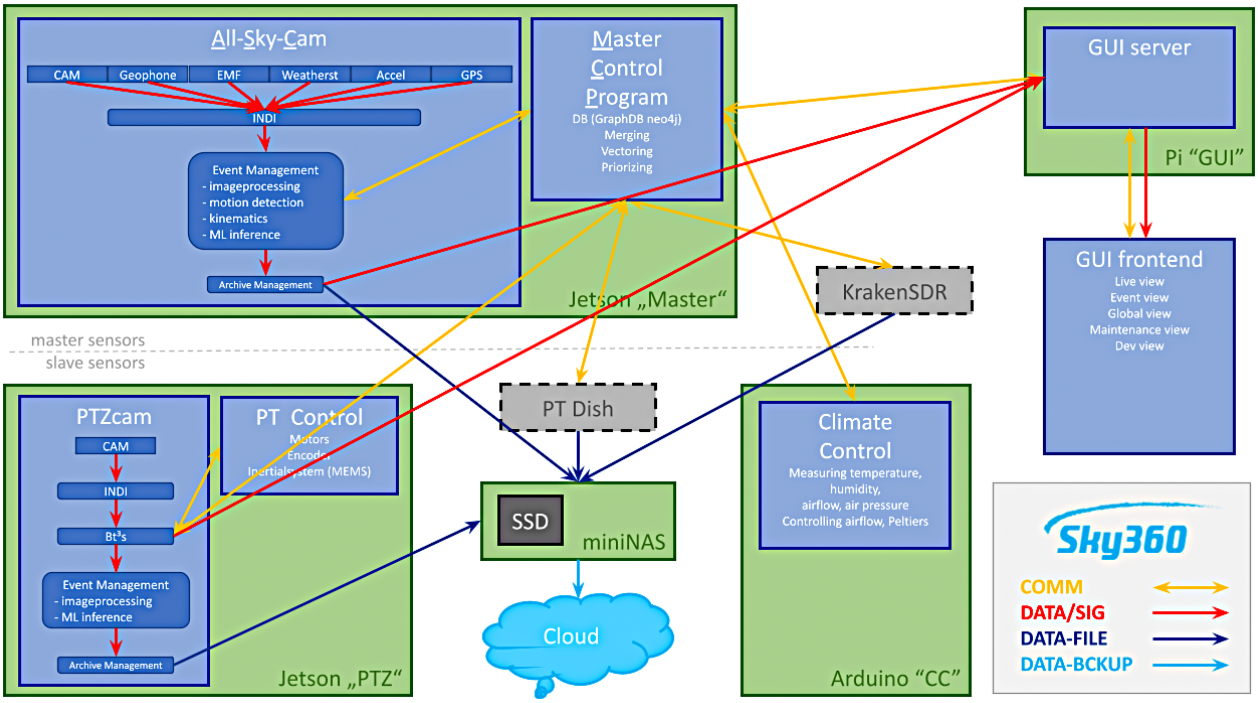
Sky360 Development and Design Concept Guide

Version 0.2

August 2022

1 Overview

The sky360 station is intended to be a platform for mining the sky on a variety of electromagnetic frequencies. Suitable sensors should monitor the progression for each group of frequencies. Therefore sensors or sensor arrays have to collect data in real time, since the events are time sensitive.
One has to start somewhere and since humans' most preferred sense is optical, one would obviously start with the wavelengths of the optical spectrum. Every other group of wavelengths will follow as soon as possible. A second topic, that one has to take into account while developing, is, that the station and it's parts have to be available to everyone and in the same manner affordable for the lowest price possible, as only few stations will exist, if these conditions are not met.
The following schematic will show, what components are suggested for the moment, and how they will act together.



2 Allskycam - ASC

2.1 Purpose

The allskycam is supposed to view what's above the horizon of a particular location and detect any changes in optical wavelengths.

2.2 Discussion of methods

2.2.1 Basics

One-Optic-Solution As the word allskycam hypothesizes, such a device is meant to show an 360° azimuthal view. Depending on how detailed such a view shall be, there are multiple ways of achieving that. One can take just one optics, which usually refers to being a fish-eye lens.

Multi-Optic-Soltuion One can split the view into a suitable amount of optics, which than will need the same amount of cameras. This is the more expensive approach, because each camera will require it's own managing electronics. Optics with smaller fields of view can be chosen. Distortions, which fish-eyes are known for, are unlikely then and, if any there are, would be caused by bad quality lenses. All that does not make it cheaper.

2.3 Hardware

2.3.1 Optics

The Rayleigh-criterion defines the resolution of an optic. It only depends on the aperture diameter in mm and the selected wavelength.
To determine, what minimal resolution you need, you have to know how huge an object is at a certain distance and to what detail you want to detect. Therefore, at first, you have to calculate the angular size of the object\detail.

$$A_s = \frac{O_d * 360 * 3600}{2 * \pi * d}$$

with A_s as angular size in arc seconds, O_d as object diameter in meters and d as distance in meters.

You could match that to the pixel size of a camera, if you are just interested in detecting the object itself. To determine that, you have to get the resolution of the optic right, where the Rayleigh-criterion kicks in.

$$R_o = \frac{\frac{(1{,}22 * \lambda * (360 * 3600)}{(2 * \pi)}}{D}$$

with R_o as resolution in arc seconds, λ as selected wavelength in millimeters and D as diameter of the optic in millimeters, too.
Since human eyes are most sensitive to green light commonly this is a good choice for the wavelength (0.00055 mm). As it is easily seen the equation depends therefor ONLY on the diameter.
If we compare the angular size of our desired object with this resolution, and the resolution has a lower value, then the optics has the right size to just detect it. If we want the optics to show us details like geometry or texture, we have to go for that - meaning, we need to determine the angular size of that.

Nyquist-Shannon-criterion The next step is to find a suitable camera with the right magnification and resolution to match the resolution of the optic. If it magnifies too much or the pixels are too large, one gets undersampling, if the magnification is too low or the pixels are too small, one gets oversampling, because the chip can resolve more than the details the optic provides.
This criterion is called Nyquist-Shannon and discusses the sampling problem.

$$R_c = \frac{(\frac{\frac{px}{1000} * (360 * 3600))}{(2 * \pi)})}{F}$$

with R_c as camera resolution in arcsec by px, px as pixel size in µm and focal length in mm.
To see, if camera and optic fit together and match the needs of resolution for the object\detail, just compare the values by dividing them.
So much for the theory. In real life, we are glad to have an atmosphere, which blurs the potential object to fell on more than one pixel. But there are some downsides to it like, the denser the atmosphere gets the more refraction and extinction is occurring. Due to being of gaseous form, temperature, pressure,

humidity and dust shares have great impact on disturbances within the atmosphere. These disturbances are called "seeing". Due to seeing the image of an object gets blurred. Since the atmosphere is not homogeneous, seeing differs from spot to spot. Under bad conditions the distance of these spots can be very small like less than 80cm.
The impact is high, so a mathematical one pixel object will cover at least four to five pixels but it's brightness decreases as well, because the same amount of light is spread on five pixels instead of just one. This reduces the signal to noise ratio a lot, but may come in handy for detection.

Field of View for fish-eye lens The field of view in case one uses a fish-eye lense, may not be derived from geometry and is rather more complex. It should be seen as a mapping or projection and therefore depends on the manufacturer's decision. In the following there will be described the only two relevant models for us in equations for mapping an equisolid angle projection and an equidistant projection.
The most common is the equisolid angle projection and the FOV (field of view) at infinity is given by:
Framesize in both cases cover x dimension, y dimension and the diagonal.

$$FOV_es = 4 * arscin(\frac{framesize}{4*focallength})$$

As an example, with Meike MK-3.5 (f=3.5mm) in combination with QHY183C camera (13.3 x 8.87 mm, $\sqrt{(13.3^2 + 8.87^2)} = 15{,}9865$ mm) a problem will appear, because the fraction $\frac{framesize}{4*focallength}$ is larger than 1 and arcsin is not defined outside -1..1.

For x dimension of 13.3 mm one gets: $4 * arcsin\frac{13.3mm}{4*3.5mm} = 287.22°$

For y dimension of 8.87 mm on gets: $4 * arcsin\frac{8.87mm}{4*3.5mm} = 157.26°$

For the diagonal it is not calculable.

Obviously if magnification goes below one, the equisolid angle would reach 360° which is where mathematics ends... Luckily physics kicks in and resolves the mathematical paradox by putting in it's way: the planet. The outlet pupil of the MK-3.5mm has a diameter of 12.5mm. The diminution in combination with how the optic is designed, gives a viewing angle instead of parallel light beams. In our case only the y dimension defines what the camera sees. Therefore this combination is a good compromise between FOV and resolution, if signal to noise ration is high.

Another popular is the equidistant projection which field is given by:

$$FOV_ed = (\frac{framesize}{focallength}) * 57.2978$$

Example calculation: $\frac{13.3mm}{3.5mm} * 57.2978 = 217,7316°$

2.3.2 Cameras

For a suitable camera there will be several options for connection, for chip choice, cooling\no-cooling, dynamic range, color or monochrome, front or backside illumination.
The highest quantum efficiencies are available for astronomical cameras only, they provide cooling and have the highest dynamic ranges. DSLR cameras are intended for taking pictures in every day situations during daylight and with documentation or artistic purpose in mind. For these purposes one does not need high resolution optics or high magnifying focal length. Color brilliance, sharpness of the image of an object nearby and good handling are topics these cameras concentrate on.

Astronomical cameras focus on getting most electrons out of a very faint light source. Bringing the optics' resolution to match the dimensions and resolution of the chip at best quantum efficiency with lowest amount of noise without systematic errors, is the main goal of scientific camera devices. These cameras will be mostly monochromatic (gray scale), because the filter can be chosen by the observer and often special wavelengths are required. There are RGB cameras which are not intended for scientific use in the first place, but offered on the market for making pretty pictures instead of gaining analyzable image data.
To increase quantum efficiency, their chips can be cooled down to around -70° below ambient temperature which reduces the necessary exposure time a lot and additionally reduces noise and static at the same time.
Astronomical cameras don't need any display and their software provides the possibility to take special frames individually like Bias frames, which contain the modulation of noise of the chip. This can be viewed, when just reading out the chip directly for that moment (0s exposure time). Dark frames, taken with closed shutter and the same exposure time like the data image, contain the dark current each conductor generates while current passes through.
Flat field frames can be taken to remove artifacts like defects of the optics, optics pollution, state of the atmosphere, humidity etc. By dividing the data image through the flat field image these artifacts can be removed from the images.
And at last the data image itself contain all of the previous images, too.
These particular cameras usually operate with long and very long exposure times from 1/1000 of a second up to 20 min. They are not very good at taking high speed videos, because usually astronomical objects don't move that fast. Special data formats like fits format are common.

Connection Modern day suitable connection are USB3.x, IP and Wifi at all means.
Security cameras can be found mostly as IP or Wifi cameras. Their intention is not to give details, but to just inform the spectator about an event happening or one, that has happened. They lack resolution in both their optics and their chip, and are optimized for video, which means just 8 bit dynamic range. They may provide higher frame rates, but are not going to extremes. For them it's suitable to be easily accessible and therefore connecting via network is a main goal.
DSLR camera can connect with USB3.x or Wifi
Astronomical camera are usually connected by USB2 or 3.

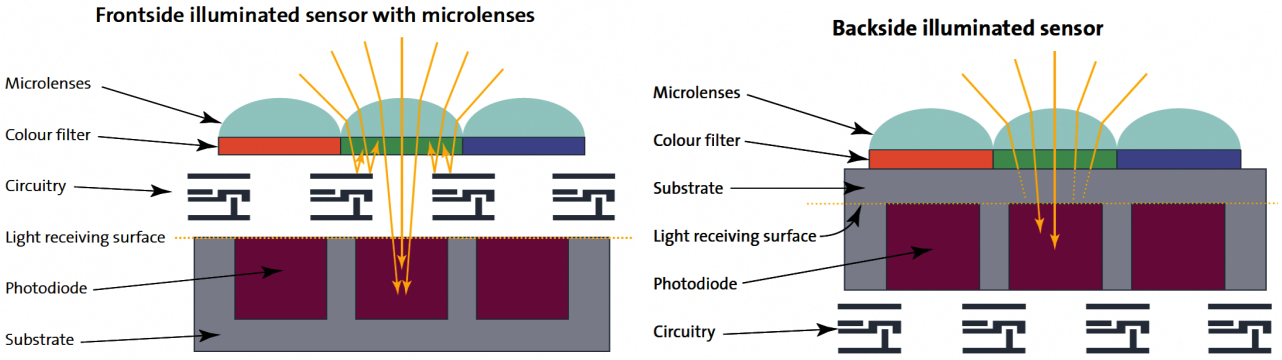
Cooling\No-cooling As mentioned above, through cooling the chip down to about -12°C CCD and CMOS chips loose a lot of their resistance and gain better quantum efficiencies and reduce noise and artifacts. Mostly this cooling is done by Peltier-elements and fans take away the heat from the hot side. This consumes power. But since the benefits are higher than the costs, it's what one wants.

Dynamic Range The dynamic range is given in bits and defines how many shades of grays or colors can be separated. One can understand this as intensity resolution. Video is usually done in 8 bit, because 256 shades per channel already is a lot of data, which has to be processed and requires powerful hardware.
If one has swayed images the usual eye-brain connection doesn't recognize all the details and 256 shades per channel in most cases is enough for entertainment. This may differ when it comes to other purposes of a recording.
The video buffer then defines, how many images per second can be stored. 12 bit cameras result in 4096 shades per channel and increases the amount of data by four times. Non-video cameras take just single images so the frame rate doesn't matter and therefore read-out times don't matter.

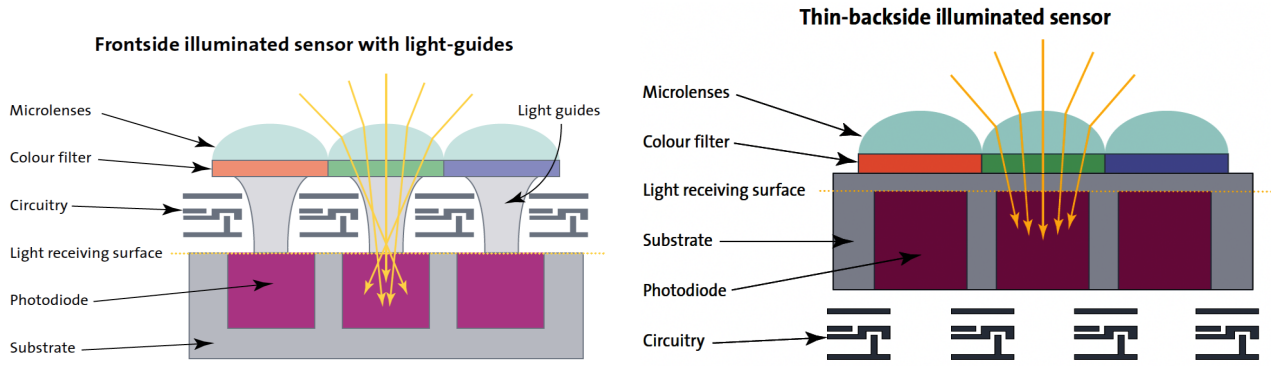
Color\Monochrome DSLR and video cameras usually make use of cmos chips with vacuum metallized filter material in a bayer matrix (from bottom left: RGGB). This results in a resolution loss by four times, because to get a color pixel four pixels are needed. Nowadays, the loss of resolution is

equaled out by pixel doubling. But just doubling the pixels results in fuzzy images. More sophisticated algorithms, that already work on the embedded hardware of the camera are available. These include edge detection, halo reduction and sharpening.
Monochrome cameras don't suffer from resolution loss. If infrared or ultraviolet as a band or as a particular wavelength is needed, for example for scientific analysis, color chips can't be used. They already have vacuum metallized filters. If one wants to detect something around 945 nm for example, a filter that only allows only 925 to 960 nm won't show anything but it's normal static. This is because even the red pixels are too far off.

Front\Back Side Illumination The difference between front and back side illumination is, that the control electronics don't cover the photon sensitive area.



As the front side illustration shows, the circuitry reflects photons back to the filter, which are then deflected back to the light receiving surface. But not all of them are and they are slightly time shifted. This reduces quantum efficiency significantly.
A different approach is shown in the following figure. To reduce or avoid the reflection, special light guides are mounted. This addition is expensive.



It is more cost effective to use back side illuminated chips with reduced thickness of the substrate.

2.3.3 Electronics

Digital cameras need some kind of peripheral electronics to be controlled with. DSLR cameras already have it within their bodies. Security cameras store it in their housings behind the chip and astronomical cameras split it up into two parts: the directly needed electronics, which control the chip and make data transfer possible, and that part on which the controlling software is run.
In our case of running an allskycam autonomous- and remotely, but with the wish to hook up some kind of display to it, all cameras have in common, that some kind of autonomous controlling unit, to execute recurring tasks, must be present. This can be found in desktop computers, laptops or embedded systems like raspberry pis.
Each software task has it's requirements. Special hardware for use with neural networks is available from NVIDIA™. They provide special libraries to use with their hardware, too. It really depends on what one wants to do under what circumstances and what budget is available, which boards one would choose.
There are three strategies that can be followed.

- Kill everything by money
- Look for the best hardware to solve the task
- See what one can do with lowest budget

One of the key features which was defined for the project, is being affordable to everyone. This renders impossible to follow the first strategy. The best hardware available for a task often results in having to spent huge amounts of money, but may not solve the problems, because availability doesn't mean that things can be delivered. This defies strategy two, leaving us with the third one. The third strategy fulfills the requirement of being affordable, but defies getting the best hardware for the task. Another requirement is, that the use of development libraries should be limited to as few as possible. Since python was chosen to be the main programming language while sticking to the most supported extensions (one doesn't want to run out of support for the next 10 to 15 years), bringing in different, not related languages causes a Moloch of stuff to maintain and to react on changes. Clean code stands above quick (and dirty) results. The project is not in a rush and denies deadlines (where not suitable). This has an impact on the search for suitable hardware, too. At the moment NVIDIA™offers it's Jetson series (nano, NX, AGX etc) with a limited production time to 2024 to 2027 which makes reaching the desire for long term usage unlikely. This can be addressed with writing code that doesn't depend on special hardware and improves performance with their follow ups.
A part of the strategy then could be, to go for the least, that just matches the requirements with compromises, but compensates these by applying clever programming.
As one can see, discussing electronics is closely related to the discussion on how to develop the software that should be run by the electronics.
This calls for a definition on the requirements and how to meet them.

Table 1:

USB3.x connector for at least 4 devices	Jetson boards, Raspberry Pi's, ATX boards, etc
Linux as operating system	any board
Hardware Neural Network support	NVIDIA™
Low power consumption	mining boards, Raspberry Pi's, Jetson boards etc
Small in size or space-saving while mounted	mini and micro ATX boards with passive cooling and no PCIe cards, Jetson boards, Raspberry Pi's

One should consider and research, where fpga boards are actually at, and if they reached affordability yet.
The list of requirements is far from complete and should be reviewed from time to time.

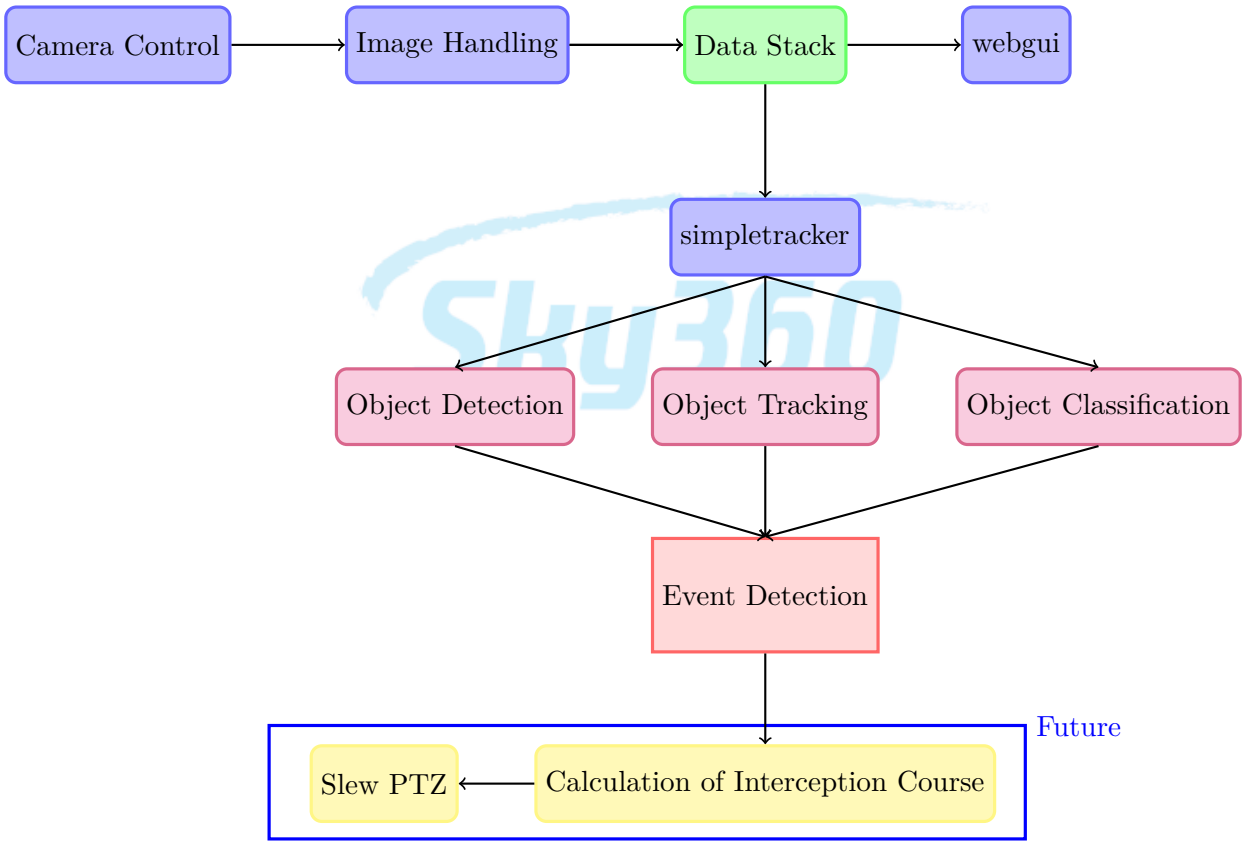
2.4 Software

This section is the convergence of multiple authors due their expertise. We will describe a road map of how we think, the development process may lead us to suitable solution.

2.4.1 General task description, overall process

There will be several tasks to think of: camera control, getting an image out of the camera, putting it on some kind of pile (stack, websocket, fila...), accessing it from a different processe on a different device, segmenting it, running intelligent algorithms for detecting an object, tracking an object, calculating it's movement path in right ascension and declination, calculating an interception course.

Figure 1: Allskycam - Flow



2.4.2 Camera Control

INDI (**I**nstrument **N**eutral **D**evice **I**nterface) is a distributed control system protocol. INDI server is basically a hub, that provides a connection between clients and device drivers. Clients are frontends to communicate with hardware drivers. And drivers are programs that communicate with the device directly.
In our case, we are going to make use of this protocol, because it exists since 2003, covers plenty of cameras and other devices, has good support and will be on the market for several additional years as it became a standard.
INDI gives support for ZWO Asi and QHY cameras as well as for v4l2 (which lets us support Skyhub's

choice of the Dahua cam for some time).
Since python was agreed on as major programming language, pyindi-client will handle the commands.

2.4.3 Acquiring Images

We will use python and pyindi-client or alikes to interface with the cameras INDI server provides. Which alike is best, will be found out during implementation.

2.4.4 Images to Stack

The outcome from INDI's camera control can be saved into images or data formats. This can of cause be PNG, TIFF or anything else, because of the suitable python library.
Another way of storing images is the data format fits which stands for **f**lexible **i**mage **t**ransport **s**ystem `https://fits.gsfc.nasa.gov/fits_documentation.html` and was developed in the late 70ies, first standardized in 1981 and is available in it's 4th version since 2016. It is not an image format because one can store spectral data or even tables and multi dimensional arrays, too. With astropy.io all that nice algorithms for opening, reading, writing, manipulating and closing fits files are available and more. With astropy.io one can even do all the necessary calculations needed in astronomical or other sciences including analysis of the stored data.
The acquired images can be sent to any type of stack in any format.

2.4.5 Data Stack

If one talks of data stacks, this can be anything from a first-in-last-out buffer to a websocket or a container system like docker. Of consideration should be, if there is a suitable graph database, which has real time capabilities.
Since we need an automation like a pipeline, ros2 (**r**obot **o**perating **s**ystem, `https://docs.ros.org/en/humble/index.html`) could be of help. This is because, from controlling a camera over data acquisition to data analysis and storing, the tasks are quite robotic. Like in programming robots, things have to be done in a particular order and with a controlled timing. One can think of a robotized mining of the sky. So ros2, as it is designed for controlling sensors, motors and make use of intelligent algorithms as being the "brain", will find its sensors in camera, geophone etc, its motors in zhe real motors to slew the PTZ later on and its "brain" in the intelligent algorithms (IA) in detection, tracking, classification and event handling.

2.4.6 Intelligent Algorithms - detection, tracking, classification... (MikeG)

As a follow-up of Skyhub, the original work was reviewed and a simple tracker, called simpletracker was implemented in python to fill in for the unfinished version of skyhub.

2.4.7 Webgui - viewing what the station does (beeks)

2.4.8 Analysis Pipeline - ros2 (Ido)

2.4.9 Archive Loader - getting stuff out of the archives (Nicola)