

# Wanted: Shadows

aneb zjišťování výšky mraků pomocí Coralu a RaspberryPi

**M. Matta, E. Plic, T. Rajchman, D. Němec, L. Kohoutková**  
**[astrox@mgplzen.cz](mailto:astrox@mgplzen.cz)**

Tým AstroX z Masarykovo Gymnázia

17. března 2023

# Kdo jsme?

## Studenti z Masarykova gymnázia

- Lucie Kohoutková
- Matyáš Matta
- David Němec
- Eduard Plic
- Tomáš Rajchman

Monday, 19 Sept • 16:19

Nějaký nápad na název týmu na tu soutěž? Máš na to 30 sekund

AstroX



# Poděkování

# Osnova

- V čem soutěžíme?
- Jak jsme na soutěž přišli?
- Co jsme vymysleli?
- Jak jsem se dostali na FAV?
- Co jsme naprogramovali?
- Co se stane, až to doběhne?
- Zdroje

# V čem soutěžíme?



- mezinárodní soutěž Astro Pi pořádaná **Evropskou vesmírnou agenturou** a **Raspberry Pi Foundation**
- mise Space Lab, která umožňuje týmům mladých studentů provést vlastní experiment na **ISS**
- programování mikropočítače Raspberry Pi programem napsaným v **Pythonu**

# Vysvětlení pojmů

# Co je to Python?

- programovací jazyk
- nejčastější jazyk pro data science a umělou inteligenci
- velmi často je vyučován jako jeden z prvních jazyků
- vzhledově lehce čitelný, oddělování bloků odsazováním



# Co je to umělá inteligence?

- poslední dobou spíše catchword<sup>1</sup>
- přesnějším popisem našeho využití je **strojové učení**
- modelu je předložen dataset<sup>2</sup> obsahující velké množství informací
- model v datech postupně začne hledat spojitosti a závislosti
- při dostatečném množství trénovacích dat je pak model schopen nalézt výsledky i na obrázcích, které neobsahují anotace<sup>3</sup>



---

<sup>1</sup> slovo, které vyvolává speciální pozornost

<sup>2</sup> soubor obrázků obsahující anotované<sup>3</sup> objekty

<sup>3</sup> soubory souřadnic ohraničující objekty



# Co je to Jupyter notebook?

- dokumenty obsahující buňky s kódem a komentáře k němu
- umožňuje spouštění kódu krok za krokem

## 1.3.3 labelmap.pbtxt

Repeat the procedure, only change the result file name to *labelmap.pbtxt*, keep both labelmaps and paste them to this Jupyter notebook.

## 1.3.4 train.tfrecord and val.tfrecord

Open the *train* folder that is in the TFRecord file export. Find the file that ends in *.tfrecord* and rename it to *train.tfrecord*. Repeat the procedure for *valid* and rename to *val.tfrecord*. Upload both files to the Colab Jupyter notebook

## 2. Preparing the bare TFLite model

This code will prepare all the files necessary to train the pure model.

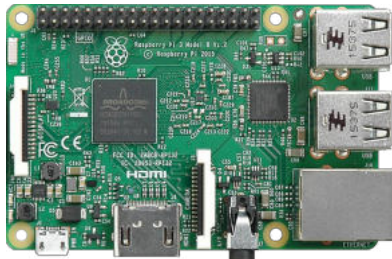
```
[ ] !git clone --depth 1 https://github.com/tensorflow/models
```

```
Cloning into 'models'...
remote: Enumerating objects: 3689, done.
remote: Counting objects: 100% (3689/3689), done.
remote: Compressing objects: 100% (3073/3073), done.
remote: Total 3689 (delta 991), reused 1503 (delta 565), pack-reused 0
Receiving objects: 100% (3689/3689), 48.74 MiB | 16.48 MiB/s, done.
Resolving deltas: 100% (991/991), done.
```

```
[ ] # Copy setup files into models/research folder
%%bash
cd models/research/
protoc object_detection/protos/*.proto --python_out=.
#cp object_detection/packages/tf2/setup.py .
```

# Co je to Raspberry Pi?

- malý jednodeskový počítač s deskou plošných spojů
- o velikosti zhruba platební karty
- operačním systémem je Raspbian
- standardně kolem 1500 korun



# Co je to Google Coral?

- multičipový modul
- zrychluje běh umělé inteligence
- připojuje se standardně přes USB-C



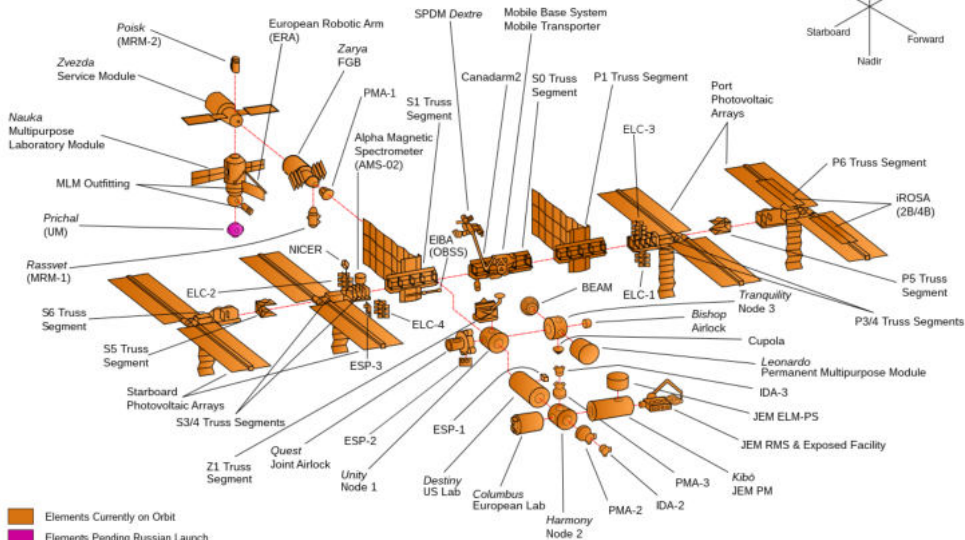
# Co je to ISS?

- jediná trvale obydlená vesmírná stanice
- ve výšce kolem 400 km
- při průměrné rychlosti okolo 7,5 km/s (27 720 km/h)
- s periodou cca 92 minut



# ISS Configuration

As of July 2021



# Jak jsme na soutěž přišli?

- doporučení od kamaráda
- nevěděli jsme, kdy se bude konat, jelikož nepřicházely žádné další informace
- na poslední chvíli jsme **14. října** vyrazili do Prahy v narychlo sestaveném týmu

planetum



# Jak probíhal hackathon?

- prohlídka planetária
- vysvětlení soutěže
- vymýšlení nápadů
- večerní umělecké promítání
- vytváření prezentace
- prezentování
- vyhlášení





Obrázek: Nejvíce nás potěšila chálka<sup>3</sup>

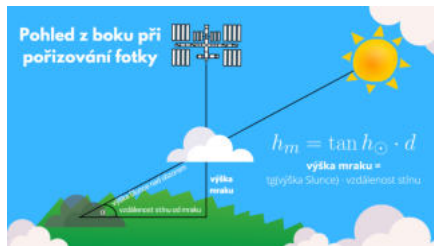
---

<sup>3</sup>chálka = potrava, jídlo



# Co jsme vymysleli?

- spoustu nerealizovatelných nápadů
  - rozpoznání krajiny
  - velikost měst v závislosti na světle
- určování **výšky mraku**
  - s pomocí stínu
  - možné určení typu mraku
- nutný záložní plán



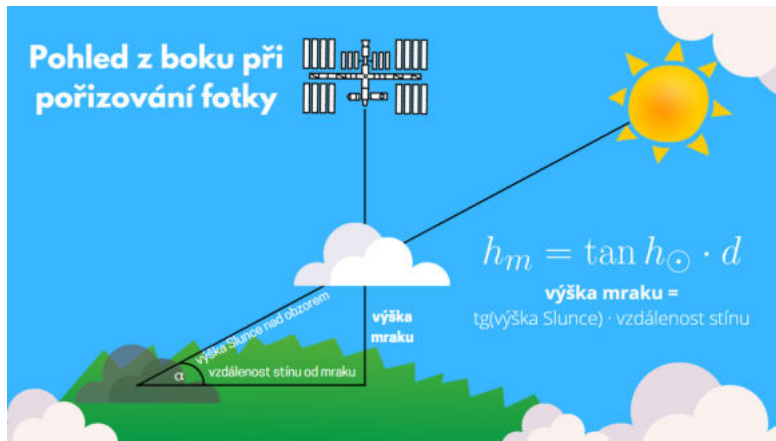
# Jak vypadá fotka



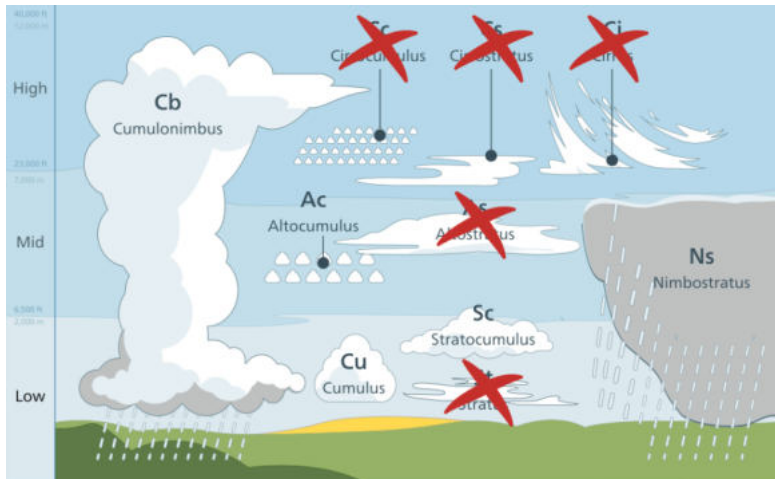
# Detailní pohled na mraky



# Trigonometrie



# Shadows not found



# Prezentace nápadu v Praze

- prezentace před tříčlennou porotou



# Praktické využití systému



Levné a efektivní určování výšky a typu mraků



Letecká doprava



Meteorologie







# Výhra v Praze

- z 6 týmů jsme obsadili **1. místo**



# Komunikace s ESA

- museli jsme připravit text a odpovědět na zadané otázky

Please tell us how you will use the Coral machine learning accelerator in your experiment.

What type of data does your team plan to gather? How will this data help test your hypothesis?

## ! Phase 1 feedback

Look at the skyfield python library. With this you can compute ISS location and the angle of solar elevation at the point below the ISS. This is a great idea. Very excited to hear about the experiment after it runs on the ISS!

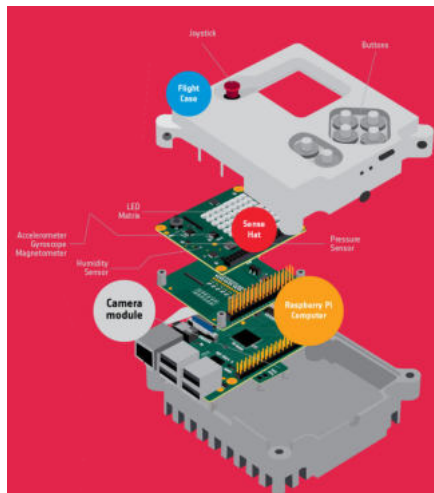
# Obdržení sady

- poté jsme obdrželi sadu **Raspberry Pi** s příslušenstvím z **Nizozemska**



# Senzory AstroPi počítače

- PIR senzor (passive infrared, senzor pohybu)
- senzor barvy a svítivosti
- gyroskop, akcelometr, magnetometr
- senzor teploty
- senzor vzdušné vlhkosti
- senzor tlaku
- kamera



## Proč jsme potřebovali být na FAV?

- ve škole nebylo možné žádnou učebnu vyblokovat na 8 hodin
- školní **internet** (studenti MG ví)
- těžká koordinace projektu z domova

## Jak jsme se na FAV dostali?

- napsali jsme email doporučenému kontaktu podle stránek ZČU
- dostali jsme odpověď od doc. Vášy
- nápad jsme představili doc. Vášovi a doc. Masopustovi
- domluvili jsme se a dostali jsme **laboratoř** s vybavením

# Laboratoř na FAV

UC 355 - “Holodíra”







2) Given  $\rho = 0.5$   
 $\sigma_1^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_2^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_{12} = 0.5 \times 0.5 = 0.25$

3) Given  $\rho = 0.5$   
 $\sigma_1^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_2^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_{12} = 0.5 \times 0.5 = 0.25$

4) Given  $\rho = 0.5$   
 $\sigma_1^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_2^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_{12} = 0.5 \times 0.5 = 0.25$

5) Given  $\rho = 0.5$   
 $\sigma_1^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_2^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_{12} = 0.5 \times 0.5 = 0.25$

6) Given  $\rho = 0.5$   
 $\sigma_1^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_2^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_{12} = 0.5 \times 0.5 = 0.25$

7) Given  $\rho = 0.5$   
 $\sigma_1^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_2^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_{12} = 0.5 \times 0.5 = 0.25$

8) Given  $\rho = 0.5$   
 $\sigma_1^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_2^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_{12} = 0.5 \times 0.5 = 0.25$

9) Given  $\rho = 0.5$   
 $\sigma_1^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_2^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_{12} = 0.5 \times 0.5 = 0.25$

10) Given  $\rho = 0.5$   
 $\sigma_1^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_2^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_{12} = 0.5 \times 0.5 = 0.25$

377 (1.000)

1) Given  $\rho = 0.5$   
 $\sigma_1^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_2^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_{12} = 0.5 \times 0.5 = 0.25$



2) Given  $\rho = 0.5$   
 $\sigma_1^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_2^2 = 0.5 \times 0.5 = 0.25$   
 $\sigma_{12} = 0.5 \times 0.5 = 0.25$



377 63XXXX

TARGET:  
Funkční prototyp

PRÁCE

a) schematic dataset (G-H)

→ pre-made dataset (.jpg)

→ ISS 2022 data → označit

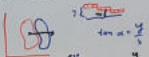
I.) připravit masku X

II.) rozkouskovat

III.) označit

P: kde je sever a jih? Lucka a David

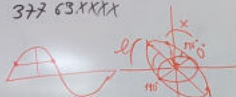
P: čímž slunce? Eda



$$x = r \cdot \cos(\alpha)$$

$$y = x \cdot \tan(\alpha)$$

$$\begin{matrix} x+2 > 3 \\ y+2 > 4 \end{matrix}$$



$$l = \sin(\alpha) \cdot r$$

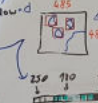
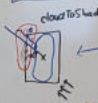
$$d = \cos(\alpha) \cdot r$$

$$l = \sin(\alpha) \cdot r$$

$$d = \cos(\alpha) \cdot r$$



1	2	3
4	5	6
7	8	9



Toni a Plaz

$$x+1$$

$$y = x \cdot \tan(\alpha)$$

$$in\_range = 100 \text{ a } \dots$$

$$d = \cos(\sin^{-1}(\frac{l}{r})) \cdot r$$

$$l = 25,0^\circ \quad d = \pm 39^\circ$$

$$l = 30,0^\circ \quad d = \pm 36,6^\circ$$

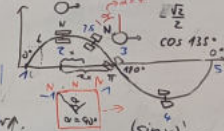
$$l = 40,0^\circ \quad d = \pm 28,4^\circ$$

kde 'l' je  
kompasní síla  
'd' je přesměr N

[25,0; 39]

[30,0; 36,6]

[40,0; 28,4]



$$(\sin x)' = \cos x$$

$$|\sin x|$$

$$54,6^\circ$$

$$54,6^\circ$$

$$54,6^\circ$$

$$0^\circ \sim 54,6^\circ$$

$$-45^\circ \sim 0^\circ (N)$$

$$0^\circ \sim -54,6^\circ$$

$$45^\circ \sim 0^\circ (S)$$

$$y = x?$$

- 1 → 1
- 2 → 0
- 3 → -1
- 4 → 0

# 0 programu

# Jak detekovat mraky?

# Použít RS-Net<sup>5</sup>

- po přečtení studie o RS-Net [3] architektuře jsme mysleli, že by se tento kód dal použít
- metoda postupně se zmenšujících bounding boxů <sup>4</sup>
- pro komplikovanost jsme nebyli schopni tuto metodu aplikovat

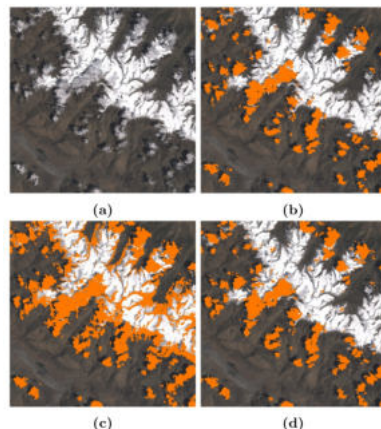


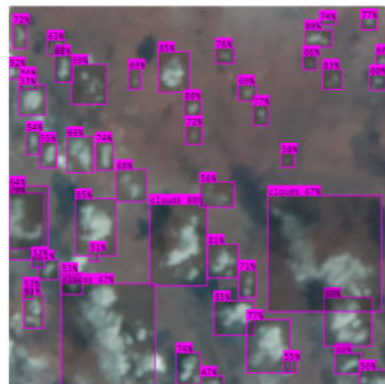
Fig. 5. Example of a SPARCS scene (LC81480352013195LGN00\_32) classified using Python Fmask and RS-Net, showing (a) RGB scene, (b) ground truth, (c) Fmask prediction and (d) RS-Net single band prediction.

<sup>4</sup>čtveřice souřadnic ohraničující objekt

<sup>5</sup>Remote Sensing Network

# Object detection umělá inteligence

- vytrénování jednoduchého modelu na detekci mraků a stínů
- výpočet výšky ze vzdálenosti souřadnic
- detekce stínů se ukázala být příliš komplikovaná a nespolehlivá
- touto metodou detekce jen mraků



# Jak se trénuje „umělá inteligence“?



# Jak se připravuje dataset?

- použili jsme **Roboflow**<sup>7</sup> [1]
- sehnali jsme obrázky z minulého ročníku Astro Pi
- vybrali jsme 16 obrázků (pozor na overfitting<sup>8</sup>)
- rozřezali jsme je na 16 sektorů
- ručně jsme udělali bounding box anotace




---

<sup>7</sup> online služba umožňující kolaboraci na anotaci společného datasetu


<sup>8</sup> trénování modelu na příliš ideálních datech


Annotations

## Attributes

 zchop-meta-x000-y970-n021.jpg  
485X485 0.24MP

 Updated Jan 20, 2023  
9:59AM GMT+01:00

 RaspberryPi RP\_imx477

 Default Set

Attributes

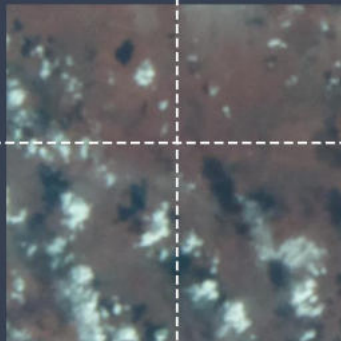
Comments

Tags

History

Raw Data

SHORTCUTS



- 60%  + RESET



1940 px



485 px

# Jak jsme získali model?

- použili jsme Jupyter notebook podle YouTube videa [2]
- Google Colab<sup>12</sup> díky velkému výkonu
- vytrénování Tensorflow<sup>10</sup> 1.x modelu a konverze na kvantovaný **TFLite** model pro Coral<sup>11</sup>
- několika hodinové trénování modelu



---

<sup>10</sup>bezplatná knihovna a platforma pro trénování modelů od Googlu

<sup>11</sup>nám poskytnuté příslušenství k Raspberry Pi pro zrychlení běhu AI modelů

<sup>12</sup>služba poskytující bezplatný hosting Jupyter notebooků

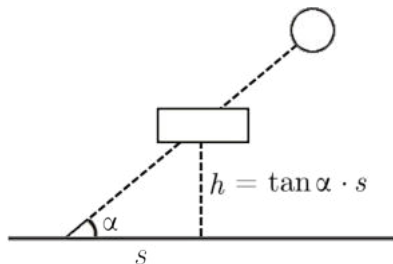
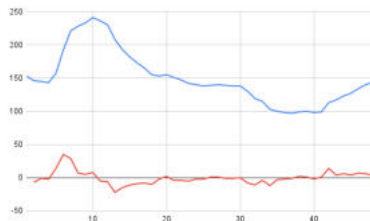
# Jak jsme model implementovali?

Praktická ukázka kódu třídy `ai`

# Jak detekovat stíny?

# Cloud-shadow fingerprint

- svítivost jednotlivých pixelů ve směru, kde by se měl nacházet stín
- bod s nejvyšší svítivostí by byl mrak a bod s nejnižší svítivostí by byl stín



## Jakým směrem hledat stíny?

- stíny budou padat podle azimutu Slunce nad obzorem
- azimut Slunce je závislý na tom, kde je sever
- sever **není** fixní a nedá se z fotky určit
- Raspberry sice má magnetometr, ale pro kvalitní určení toho, kde je stín, by se odchylky musely pohybovat v jednotkách stupňů
- ...alespoň to tvrdila ESA



# Jak zjistit sever?

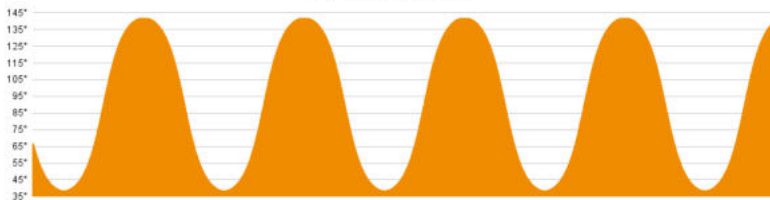
# Proč ne kompas?

- nepřesnost 20° a nesmyslné hodnoty

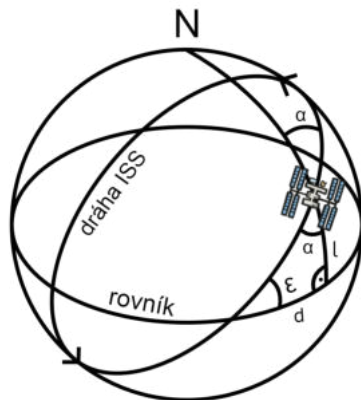
Sever podle kompasu



Skutečný sever



# Sever vůči ISS

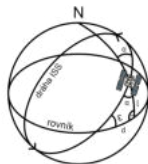


- $\alpha$  ...úhel mezi severem a směrem pohybu ISS
- $\varepsilon$  ...sklon dráhy ISS vzhledem k rovníku
- $l$  ...zeměpisná šířka
- $d$  ...pomocná proměnná ("zeměpisná délka")

# Sever vůči ISS

- použijeme **sférickou sinovou větu**

$$\frac{\sin(d)}{\sin(\alpha)} = \frac{\sin(l)}{\sin(\varepsilon)} \Rightarrow \sin(d) = \frac{\sin(l)}{\sin(\varepsilon)} \sin(\alpha)$$



- **kotangentový vzorec** pro čtyři parametry:

$$\cos(90^\circ) \cos(d) = \cot(l) \sin(d) - \cot(\varepsilon) \sin(90^\circ)$$

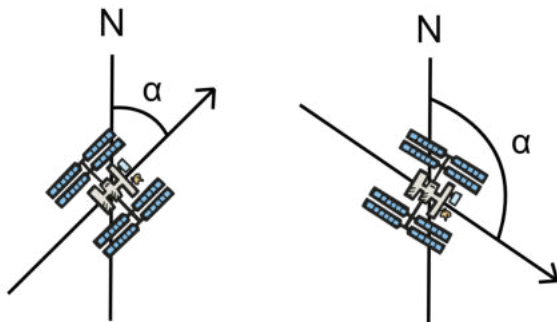
$$\cot(\varepsilon) = \cot(l) \sin(d)$$

- zkombinujeme

$$\cot(\varepsilon) = \cot(l) \frac{\sin(l)}{\sin(\varepsilon)} \sin(\alpha) \Rightarrow \sin(\alpha) = \frac{\cos(\varepsilon)}{\cos(l)}$$

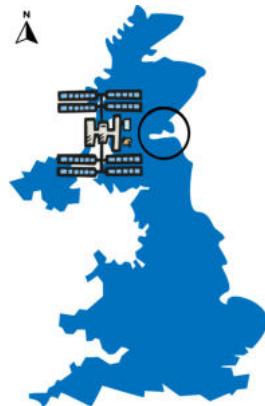
# Sever vůči ISS

- konečná závislost  $\alpha$  na zeměpisné šířce:  $\alpha = \arcsin\left(\frac{\cos(\varepsilon)}{\cos(l)}\right)$
- vzorec vrátí ostrý úhel  $\rightarrow$  nutná korekce  $\alpha$ :
  - **stoupající** šířka - úhel **zůstává**
  - **klesající** šířka - vezme se jeho **doplňěk ke 180°**



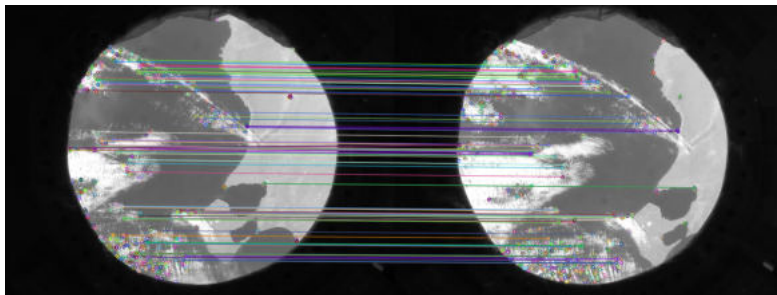
# Quō vādis ISS

- fotka může být **pootočená**



# OpenCV: knihovna

- knihovna pro **manipulaci s obrazem**
  - rozpoznávání obličejů
  - identifikace objektů
  - klasifikaci lidských akcí ve videu
  - mnoho dalších...



# OpenCV: posun obrazu

```
# finding same "things" on both images
def calculate_features(image_1, image_2, feature_number):
    orb = cv2.ORB_create(nfeatures = feature_number)
    keypoints_1, descriptors_1 = orb.detectAndCompute(image_1_cv, None)
    keypoints_2, descriptors_2 = orb.detectAndCompute(image_2_cv, None)
    return keypoints_1, keypoints_2, descriptors_1, descriptors_2

# connecting same "things" on photo
def calculate_matches(descriptors_1, descriptors_2):
    try:
        brute_force = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
        matches = brute_force.match(descriptors_1, descriptors_2)
        matches = sorted(matches, key=lambda x: x.distance)
        return matches
    except:
        return 0
```

- z posunu obrazu lze zjistit **natočení kamery**



# OpenCV a Sever vůči ISS

- zkombinováním **polohy severu vůči ISS** a **natočení** kamery získáme **skutečnou polohu severu** na fotce

```
alpha_k=np.arcsin(np.cos(51.8/57.29577951)/np.cos(latitude_avg/57.29577951)) * 57.29577951
corrected_alpha_k=0
# correcting "the equator situation" with looking if ISS moves up or down
if latitude_image_1>latitude_image_2:
    corrected_alpha_k=180-alpha_k
else:
    corrected_alpha_k=alpha_k

# combining both informations to get real position of north on photo
north_position = all_edoov_coefficient - corrected_alpha_k
if north_position < 0:
    north_position = north_position + 360
return north_position
```

- natočení kamery → sever na fotce → slunce na fotce → směr ke stínům

# Jak jsme program implementovali?

Praktická ukázka kódu třídy `north`

# Jak detekovat stíny?

# Jak projet jednotlivé pixely?

1. vypočteme střed mraku podle dat z třídy ai

```
def calculate_cloud_data(counter_for_shadows):  
    x_max = data[counter_for_shadows]['xmax']  
    y_max = data[counter_for_shadows]['ymax']  
    x_min = data[counter_for_shadows]['xmin']  
    y_min = data[counter_for_shadows]['ymin']  
    x_centre_of_cloud = (x_min+x_max)/2  
    y_centre_of_cloud = (y_min+y_max)/2  
    x_centre_of_cloud = round(x_centre_of_cloud, 0)  
    y_centre_of_cloud = round(y_centre_of_cloud, 0)  
    x_centre_of_cloud = int(x_centre_of_cloud)  
    y_centre_of_cloud = int(y_centre_of_cloud)  
    x_cloud_lenght = abs(x_max - x_min)  
    y_cloud_lenght = abs(y_max - y_min)
```

## Jak projet jednotlivé pixely?

2. rozdělíme informaci o úhlu na kvadrant a základní úhel

### Směr kvadrantu a úhlu

Úhel je počítán po směru hodinových ručiček od severu (i.e. od vrchní strany), kvadranty jsou definovány obdobně, je to tedy jinak než je běžná matematická definice.

```
if 0 <= angle <= 90:  
    q = 1  
    angle_final = angle  
if 90 < angle <= 180:  
    q = 2  
    angle_final = angle - 90  
if 180 < angle <= 270:  
    q = 3  
    angle_final = angle - 180  
if 270 < angle <= 360:  
    q = 4  
    angle_final = angle - 270
```

# Jak projet jednotlivé pixely?

## 3. úhel přepočteme na přírůstek $x$ a přírůstek $y$

■  $\alpha = 22.5^\circ$

$$x \ += \ 0.38$$

$$y \ += \ 0.92$$

```
angle_radians = np.radians(angle)
x_increase_meta = np.sin(angle_radians)
y_increase_meta = np.cos(angle_radians)
y_increase_meta = -y_increase_meta
x_increase_meta = np.round(x_increase_meta,5)
y_increase_meta = np.round(y_increase_meta,5)
x_increase_meta_abs = abs(x_increase_meta)
y_increase_meta_abs = abs(y_increase_meta)
```

## Jak projet jednotlivé pixely?

animace

4. přírůstky přepočteme tak, aby alespoň jeden z nich byl 1

→  $x += 0.38, y += 0.87$

→  $x += 0.41, y += 1$

```
# make at least one variable 1 and the other smaller than 1
if x_increase_meta_abs > y_increase_meta_abs:
    x_increase_final = 1
    y_increase_final = y_increase_meta_abs/x_increase_meta_abs
if x_increase_meta_abs == y_increase_meta_abs:
    x_increase_final = 1
    y_increase_final = 1
if x_increase_meta_abs < y_increase_meta_abs:
    x_increase_final = x_increase_meta_abs/y_increase_meta_abs
    y_increase_final = 1
if angle_final == 0:
    x_increase_final = 0
    y_increase_final = 1
```

# Jak projet jednotlivé pixely?

## 5. vypočteme tzv. limit hledání stínů

```
# automatic limit calculation
# here we set basically how far we should search for shadows
sun_altitude_for_limit = shadow.sun_data.altitude(latitude, longitude, year, month,
day, hour, minute, second)
sun_altitude_for_limit_radians = sun_altitude_for_limit*(np.pi/180)
limit_cloud_height = 12000 #meters
limit_shadow_cloud_distance = limit_cloud_height/np.tan
(sun_altitude_for_limit_radians)
limit_shadow_cloud_distance_pixels = limit_shadow_cloud_distance/126.48
limit = limit_shadow_cloud_distance_pixels
```



# Jak projet jednotlivé pixely?

- pro každý sektor je loop<sup>13</sup> definován znova
- máme tři situace,  $x > y$ ,  $x = y$  a  $x < y$
- celkem máme 12 (4 sektory, 3 situace) možností, jak kód může běžet
- kvůli komplikacím s univerzálním řešením jsme nakonec definovali každou situaci zvlášť

```
while True:
    # this is the count we use to run pixel by pixel
    count += 1
    if x_increase_final_abs < y_increase_final_abs:

        # check if x_sum is bigger than 1
        y_sum = abs(y_sum)
        if x_sum >= 1:
            x_sum -= 1
            x += 1

        # read pixel value
        data = (pix[x,y])

        # save the value, also search in only red
        value = round(average(data))
        value_red = data[0]

        # append onto a list
        list_of_red.append(value_red)
        list_of_values.append(value)

        # add to y_sum and move pixel x for 1
        y -= 1
        x_sum += x_increase_final_abs
        if x > total_x or y > total_y:
            break
```

<sup>13</sup> smyčka, běží dokud je podmínka platná

## Jaký je výsledek?

- získáme list obsahující jednotlivé průměrné hodnoty z RGB pixelů
- a také list obsahující hodnoty jen v červeném spektru
- ty poté dále vložíme do jednotlivých funkcí na výpočet vzdálenosti mezi stínem a mrakem

```
[153, 146, 145, 143,  
157, 192, 221, 228,  
233, 241, 236, 230,  
208, 193, 182, 173,  
165, 155, 153, 155,  
151, 147, 142, 140,  
138, 139, 140, 139,  
138, 138, 130, 119,  
115, 103, 100, 098,  
097, 099, 100, 098,  
099, 113, 117, 123,  
127, 134, 140, 144]
```

## Metoda první: minimum-maximum

- v listu najdeme nejvyšší a nejnižší hodnotu
- zjistíme pozice těchto hodnot v listu
- vypočteme vzdálenost mezi stínem a mrakem

```
def calculate_using_min_max(list_of_values):  
    def main():  
        # find items in list corresponding to the  
        # lowest and highest point  
        shadow_low = min(list_of_values)  
        cloud_high = max(list_of_values)  
  
        # find of said items in the list (their  
        # order)  
        shadow_location = list_of_values.index  
        (shadow_low)  
        cloud_location = list_of_values.index  
        (cloud_high)  
  
        # find the difference  
        shadow_lenght = shadow_location -  
        cloud_location  
        return shadow_lenght, cloud_high,  
        shadow_low, cloud_location,  
        shadow_location
```

## Metoda první: minimum-maximum

- když získáme záporný výsledek, stín je před mrakem
- tento pixel můžeme smazat - neovlivní to vzdálenost mezi skutečným stínem a mrakem
- opakujeme dokud nám nevyjde kladný výsledek

```
while True:
    # in case that shadow is found before a
    # cloud in the line, we delete the value as
    # its false and repeat
    if shadow_lenght <= 0:
        list_of_values.remove(cloud_high)
        shadow_lenght, cloud_high,
        shadow_low, cloud_location,
        shadow_location = main()
    else:
        break
```

## Metoda druhá: maximum-change

- nejprve data  
překonvertujeme na  
změny
- i.e. první derivativ původní  
funkce
- odečteme od sebe  
sousedící pixely a dáme  
do nového listu

```
while True:
    try:
        current_data = list_of_values[n]
        previous_data = list_of_values
        [n-1]
        change_in_data =
        current_data-previous_data
        if n == 0:
            pass
        else:
            list_of_changes.append
            (change_in_data)
        n+=1
    except:
        break
```

## Metoda druhá: maximum-change

- poté opět zjistíme umístění těchto hodnot v listu
- podle pozice opět vypočteme vzdálenost mezi stínem a mrakem

```
# self-explanatory
shadow_low = max(list_of_changes)
cloud_high = min(list_of_changes)

shadow_location = list_of_changes.index(
    shadow_low)
cloud_location = list_of_changes.index(
    cloud_high)

# find difference between the two pixel
lengths
shadow_lenght = shadow_location -
cloud_location
return shadow_lenght, cloud_high,
cloud_location
```

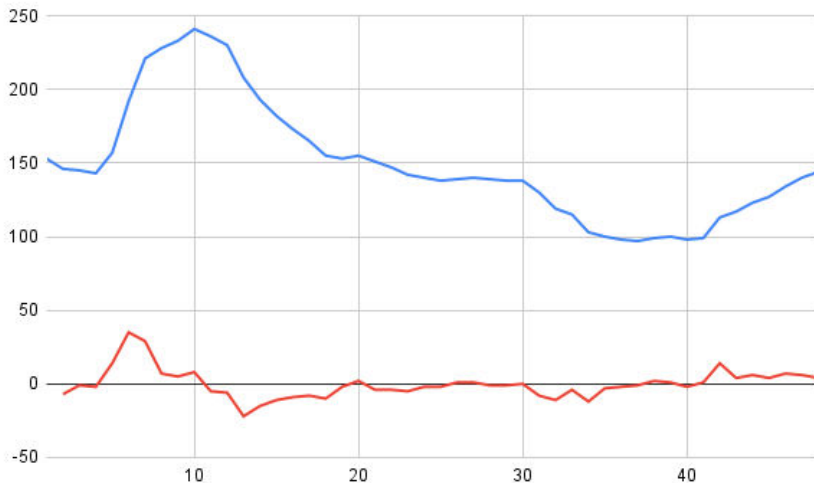
## Metoda druhá: maximum-change

- nakonec opět přidáme ochranu proti záporným výsledkům
- smyčka běží, dokud není výsledek kladný
- začneme v listu hledat až na pozici 10, aby nedošlo k chybě, kdy **nejvyšší nárůst není způsoben stínem ale mrakem**

```
# then we check for clouds after shadows and if
# necessary re-run the local main function (see
# above)
while True:
    n = 0
    if shadow_lenght <= 0:
        item_to_be_deleted = list_of_values
        [cloud_location]
        list_of_values.remove(item_to_be_deleted)
        shadow_lenght, cloud_high, cloud_location
        = main()
    else:
        break

return shadow_lenght
```

# Jak vypadá výsledek v grafu?





# Zprůměrování metod

- nyní máme definované všechny potřebné funkce a metody
- využijeme tři výsledky
  - max-change na průměru
  - max-change v červené
  - min-max na průměru
- metody zprůměrujeme
- tím získáme vzdálenost mezi stínem a mrakem
- délku z pixelů převedeme na metry pomocí konstanty

```
# here we calculate the shadow-cloud distance by
each respective method
shadow_lenght_min_max = calculate_using_min_max
(list_of_values)
shadow_lenght_max_difference =
calculate_using_maximum_change(list_of_values)
shadow_lenght_max_difference_red =
calculate_using_maximum_change(list_of_red)

# we put the methods together
shadow_lenght_final =
(shadow_lenght_max_difference
+shadow_lenght_min_max
+shadow_lenght_max_difference_red)/3

# calculate distance based on a distance in pixels
lenght = int(shadow_lenght_final) * 126.48
```

## Korekce výsledku

- vypočtenou délku je třeba přepočítat z odvěsny na přeponu
- jak je vidět v (animaci), počet pixelů odpovídá delší odvěsně, nikoliv přeponě
- výpočet jsou prosté goniometrické funkce

```
# because original lenghts are adjacent  
lenghts and not hypotenuse we will have  
to convert  
angle_final_radians = angle_final*(np.pi/  
180)  
if angle_final <= 45:  
    lenght = lenght/np.cos  
        (angle_final_radians)  
if angle_final > 45:  
    lenght = lenght/np.sin  
        (angle_final_radians)
```

# Finalizace výsledku

- pomocí již zmíněné rovnice  $h = \tan \alpha \cdot s$  vypočteme výšku mraku
- ještě předtím pomocí třídy `sun_data` zjistíme výšku Slunce nad obzorem (i.e.  $\alpha$ )
- výsledek `shadow_lenght`, který jsme doposud počítali v rovnici odpovídá  $s$

```
# now we calculate the sun altitude using a function
altitude = shadow.sun_data.altitude(latitude,
longitude, year, month, day, hour, minute, second)
```

```
# calculate final values
altitude_radians = altitude*(np.pi/180)
cloudheight = np.tan(altitude_radians)*lenght
cloudheight = np.round(cloudheight,2)
```

# Jak funguje třída sun\_data?

```
def altitude(coordinates_latitude, coordinates_longitude, year, month, day, hour,
minute, second):
    # use the NASA API to be able to calculate sun's position
    ts = api.load.timescale()
    ephem = api.load("ftp://ssd.jpl.nasa.gov/pub/eph/planets/bsp/de421.bsp")

    # define sky objects
    sun = ephem["Sun"]
    earth = ephem["Earth"]

    # given coordinates calculate the altitude (how many degrees sun is above the
    horizon), additional data is redundant
    location = api.Topos(coordinates_latitude, coordinates_longitude, elevation_m=500)
    sun_pos = (earth + location).at(ts.tt(year,month,day,hour,minute,second)).observe
    (sun).apparent()
    altitude, azimuth, distance = sun_pos.altaz()
    altitude= float(altitude.degrees)
    return(altitude)
```

# Jak jsme program implementovali?

Praktická ukázka kódu třídy `shadow`

# Potíže při prvním testování

- zpracování celého obrázku  
ca. 2.2 minuty
- maximum pro class  
north ca. 0.4 minuty
- řešením multithreading<sup>14</sup>

If you need to do more than one thing at a time, you can use a multithreaded process. There are a number of Python libraries that allow this type of multitasking to be included in your code. However, to do this on the Astro Pis, you're only permitted to use the `threading` library.

**Only use the `threading` library if absolutely necessary** for your experiment. Managing threads can be tricky, and as your experiment will be run as part of a sequence of programs, we need to make sure that the previous one has ended smoothly before starting the next. Rogue threads can behave in an unexpected manner and take up too much of the system resources. If you do use threads in your code, you should make sure that they are all managed carefully and closed cleanly at the end of your experiment. You should additionally make sure that comments in your code clearly explain how this is achieved.

---

<sup>14</sup>proces, kde běží více vláken současně, i.e. je schopen vykonávat více než jeden příkaz naráz

# Dvouvláknový proces

- jedno vlákno na produkci dat
- jedno vlákno na zpracování dat
- definovali jsme jako třídy
- samotné spuštění pod `if` funkcí

```
# classes for threads
class photo_thread(threading.Thread): ...
class processing_thread(threading.Thread): ...
if __name__ == '__main__':
    # there are many advantages to multithreaded operation
    # compare to monothread
    # suppose we ignore photos taken in the complete darkness, i.
    # e. half the time,
    # but because the processing is so much slower than the
    # photographing we will have many unprocessed images that could
    # take advantage of being run in the dark.
    # also because the processing is about 2 minutes/full image
    # (as of v2.4),
    # we would have to force-quit operation often just in order
    # to take photos in which the north class can detect similar
    # objects (when too far apart openCV would fail)
    # two threads are brilliant because even IF something in the
    # processing goes unexpectedly wrong we will still get the data
    # and photos for processing on Earth
```

# Vlákno generativní



# Vlákno generativní

- sbírá SenseHat data
- fotí v intervalu daným polohou Slunce
- podmíněno 3 GiB místem
- a 2.7 hodiny časem
- ukládá správná EXIF data<sup>16</sup>

```
if ISS.at(t+0.002777777777).is_sunlit(ephemeris)
and ISS.at(t-0.002777777777).is_sunlit(ephemeris):
    image_path = str("./main/img_" + str
(count_for_images_day) + ".jpg")
    count_for_images_day += 1
    photo_sleep_interval = 12*photo_multiplier
    main()
    camera.capture(image_path)
else:
    image_path = str("./main/night_img_" + str
(count_for_images_night) + ".jpg")
    count_for_images_night += 1
    photo_sleep_interval = 45
    main()
    camera.capture(image_path)
    photo.compress(image_path)
```

---

<sup>15</sup> pomocné, data generující

<sup>16</sup> metadata, obsahující informace o poloze a čase, ukládá se přímo do fotografie

# Vlákno zpracovací

## Inicializace

# Hlavní loop

- měří natočení kamery (=edoov\_coeficient)
- běží dokud hodnota není dost přesná
- výhodou více vláken je, že je jedno jak dlouho poběží

```
while (datetime.now() < start_time + timedelta(
seconds=256)) or eda_bad_accuracy:
    global all_edoov_coeficient
    if eda_count+1 < count_for_images_day:
        try:
            i_1=str(eda_count)
            before = "main/img_"
            image_1=str(before + i_1 + ".jpg")
            i_2=str(eda_count + 1)
            image_2=str(before + i_2 + ".jpg")
            list_medianu = north.
            find_edoov_coeficient(image_1,
            image_2)
            to_print = "Edoov koeficient was
            defined at " + str(list_medianu)
            + " counted clockwise."
            shadow.print_log(to_print)
            eda_bad_accuracy = accuracy()
            all_edoov_coeficient =
            list_medianu
            eda_count += 1
        except:
            pass
    else:
        to_print = "Waiting for an image,
        currently at " + str(eda_count)+ "
        now going to sleep."
        shadow.print_log(to_print)
        sleep(photo_sleep_interval)
```

## Podmínka přesnosti

- pokud není výsledek dost přesný neopustí loop
- chceme přesnost pod tři stupně
- využíváme medián, nikoliv průměr
- minimálně 4 minuty měření

```
def accuracy():  
    eda_bad_accuracy = False  
    if len(store_edoov_coefficient) > 1:  
        if abs(all_edoov_coefficient -  
            list_medianu) > 3:  
            eda_bad_accuracy = True  
    return eda_bad_accuracy
```

# Vlákno zpracovací

## Hlavní proces

# Extrakce EXIF metadat

- první využijeme námi definované funkce na extrakci metadat

```
# we need to set the name up first
imageName = str("main/img_" + str
(initialization_count) + ".jpg")
image_2_path = imageName

# then we define the coordinates, see
class shadow subclass coordinates for
details but it's mostly export from EXIF
data
global latitude
global longitude
latitude = shadow.coordinates.get_latitude
(image_2_path)
longitude = shadow.coordinates.
get_longitude(image_2_path)

# we extract the time information from
the image, see class exifmeta for more
information
year, month, day, hour, minute, second =
exifmeta.find_time_from_image
(image_path=image_2_path)
```

## Proč obrázek rozsekávat?

- pro anotaci rozsekáno kvůli limitacím Roboflow
- zrychlení programu
- metodou rozsekání lze zpřesnit souřadnice

```
# split image into many
split.file_split(north_main=
north_main, image_id = full_image_id,
image_path=image_2_path) # creates
a ./chop/... folder and puts the
chops into it with "astrochop_n"
syntax
```

```
# this id is also to be seen in the
final csv, it will be useful for
backtracking back on Earth
sector_id = 0
```

```
# when we start a loop for all images
in the chop, those were created with
the split above, it will always be
just 16 images
for images in os.listdir("./chop/"):

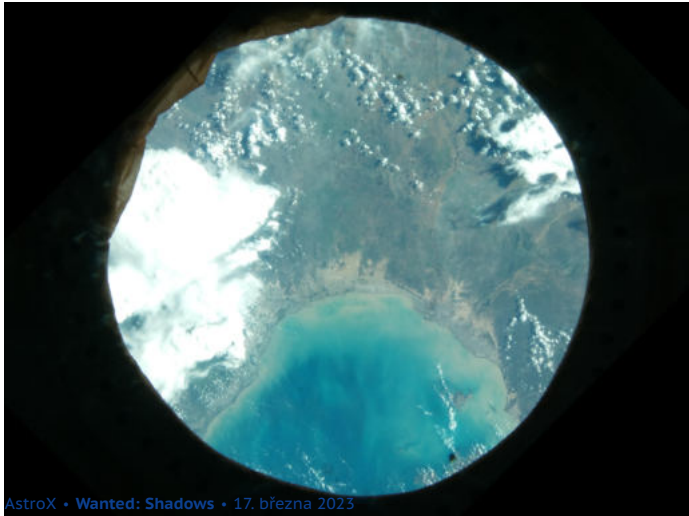
```





```
# rotate
```

```
im = im.rotate(north_main)
```





```
# crop
left = 1075
top = 520
right = 3015
bottom = 2460
im1 = im.crop((left, top, right, bottom))

# save as metadatum
im1.save('meta.jpg')
```



```

for x0 in range(0, width, chopsize):
    for y0 in range(0, height, chopsize):
        box = (x0, y0,
                x0+chopsize if x0+chopsize <
                    width else width - 1,
                y0+chopsize if y0+chopsize <
                    height else height - 1)
        file_name = "./chop/astrochop_" + str
            (within_loop_counter) + "_" + str(x0) +
            "_" + str(y0) + "_" + ".jpg"
        img.crop(box).save(file_name,exif=exif)

# we calculate the centre coordinates of
the chop based on its name and original
exif data
chop_latitude, chop_longitude = gps.
cloud_position(file_name)
latitude, latitude_ref, longitude,
longitude_ref = gps.
convert_decimal_coordinates_to_legacy
(chop_latitude, chop_longitude)

```

# Rozsekání obrázků

- pokud je Slunce nad obzorem
- funkce `file_split` obrázek otočí, ořízne, rozseká a ke každému chopu přiřadí přesnější souřadnice

```

for x0 in range(0, width, chopsize):
    for y0 in range(0, height, chopsize):
        box = (x0, y0,
                x0+chopsize if x0+chopsize <
                    width else width - 1,
                y0+chopsize if y0+chopsize <
                    height else height - 1)
        file_name = "./chop/astrochop_" + str
            (within_loop_counter) + "_" + str(x0) +
            "_" + str(y0) + "_" + ".jpg"
        img.crop(box).save(file_name,exif=exif)

# we calculate the centre coordinates of
the chop based on its name and original
exif data
chop_latitude, chop_longitude = gps.
    cloud_position(file_name)
latitude, latitude_ref, longitude,
longitude_ref = gps.
convert_decimal_coordinates_to_legacy
(chop_latitude, chop_longitude)

```

## Kdy nehledat

- slunce pod  $5^\circ$  → špatná kvalita
- obrázek moc bílý (tj. jeden velký mrak)
- moc velký mrak - zakrývá velkou část stínu, nepřesné výsledky

# Implementace umělé inteligence

- funkce z třídy ai
- úhel počítán vnitřní funkcí
- while podmínka je počet mraků (vždy 10)

```
# then we check if the image is not all-clouds or
all-sea with a brightness function, see the respective
class for more information
if 50 < properties.calculate_brightness(chop_image_path)
< 190 and properties.calculate_contrast(chop_image_path)
> 10:
    try:
        # the image is fed to the ai model, which
        returns a dictionary of cloud boundaries and
        accuracies, see the ai class for more details
        global data
        data = ai.ai_model(chop_image_path)

        # we will use this counter to label the
        dictionary correctly
        counter_for_shadows = 0

        # the angle where shadows shall lay is
        calculated using the north data and sun azimuth
        angle, see the shadow class for more details
        angle = shadow.calculate_angle_for_shadow
        (latitude, longitude, year, month, day, hour,
        minute, second)

        # this loop runs through all the clouds detected
        in an image and writes the data into the final
        csv file
        while counter_for_shadows <= 9:
```

# Implementace detekce stínů

- vypočteme počáteční pixel
- podmínka délky
- využijeme funkci třídy shadow

```
# this code will figure us out the cloud bbox
data, respectively its centre
x_centre_of_cloud, y_centre_of_cloud,
x_cloud_lenght, y_cloud_lenght = shadow.
calculate_cloud_data(counter_for_shadows)
```

```
# we check that the cloud is not too long as too
long clouds would be useless and slow the program
down
```

```
if x_cloud_lenght < 100 and y_cloud_lenght < 100:
```

```
# we add a simple error handling to make sure
we can handle unexpected expectations
```

```
try:
```

```
# this piece of code will return either
data or "error" string, that will happen
in rare cases
```

```
# so that the loop does not crash
completely and skip the cloud we use the
"error" string
```

```
result_shadow = shadow.calculate_shadow
(file_path=chop_image_path,
x=x_centre_of_cloud, y=y_centre_of_cloud,
angle=angle, image_id=sector_id,
cloud_id=counter_for_shadows)
```

# Když není, co zpracovat

- zpřesňujeme edoov\_coeficient
- případně se kód na pár vteřin uspí

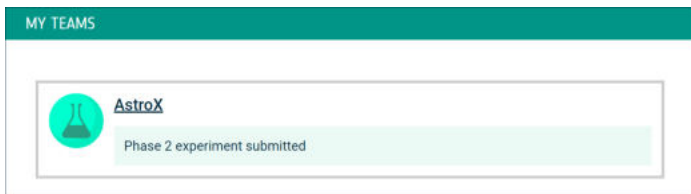
```
if eda_count+1 < count_for_images_day:
    try:
        i_1=str(eda_count)
        before = "main/img_"
        image_1=str(before + i_1 + ".jpg")
        i_2=str(eda_count + 1)
        image_2=str(before + i_2 + ".jpg")
        list_medianu = north.
        find_edoov_coeficient
        (image_1, image_2)
        to_print = "Edoov koeficient
        was defined at " + str
        (list_medianu) + " counted
        clockwise."
        shadow.print_log(to_print)
        eda_bad_accuracy = accuracy()
        all_edoov_coeficient =
        list_medianu
        eda_count += 1
    except:
        pass
else:
    to_print = "Waiting for an image,
    currently at " + str(eda_count)+
    " now going to sleep."
    shadow.print_log(to_print)
    sleep(photo_sleep_interval)
```



# Odeslání celého kódu


# Způsob odeslání

- deadline **24.2.2023** v 23:59
- zazipovaný soubor s `main.py` a daty pro umělou inteligenci
- přísné podmínky



## Přísné podmínky kódu

- ukončení do 3 hodin
- nemožnost připojení na internet
- rozběhnutí jedním příkazem
- nesmí generovat errorry
- jeden hlavní soubor
- název souboru `main.py`
- další...



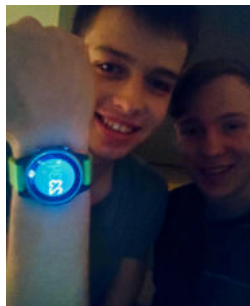
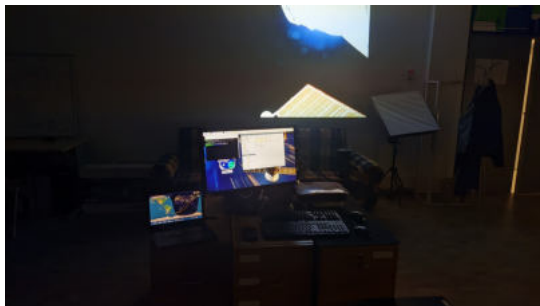
```
C:\Windows\system32\cmd.exe
```



```
C:\Users>python3 main.py
```

# Odeslání kódu

- cca 15 minut před termínem





# Data, která nám přijdou



# Údaje o mracích

Starting: Thread1

Starting: Thread2

...

Edoov koeficient (= natočení kamery) was defined at  
95.21860055030247 counted clockwise.

...

Skipped chop due to the brightness being 46.490580205916594

Cloud number 0 did not meet maximal lenght criteria

Cloud number 1 has a height of 6803.16

Cloud number 2 has a height of 4251.97

Cloud number 3 has a height of 5782.68

Cloud number 4 has a height of 3231.5

Didn't find shadow of cloud number 5

Cloud number 6 has a height of 3061.42

Cloud number 7 did not meet maximal lenght criteria

Cloud number 8 has a height of 4422.05

Cloud number 9 has a height of 4932.29

...

Exiting: Thread1

Exiting: Thread2

Done main thread



# Ostatní údaje z Raspberry

Podmínky			Čidlo barvy				Natočení			Magnetometr			Zrychlení			Gyroskop			Čas
temp	pres	hum	red	green	blue	clear	yaw	pitch	roll	mag_x	mag_y	mag_z	acc_x	acc_y	acc_z	gyro_x	gyro_y	gyro_z	datetime
42,3	962,0	17,9	0,0	1,0	1,0	2,0	200,1	350,6	16,0	-17,8	15,8	25,5	0,2	0,3	0,9	0,0	0,0	0,0	20:42:45
42,4	962,0	18,3	0,0	1,0	1,0	2,0	204,7	349,2	6,0	-36,8	32,0	53,5	0,2	0,3	0,9	0,0	0,0	0,0	20:42:59
42,3	962,0	18,2	0,0	0,0	0,0	2,0	208,5	346,4	359,7	-44,5	38,4	65,1	0,2	0,3	0,9	0,0	0,0	0,0	20:43:13
42,3	962,0	18,3	0,0	1,0	1,0	2,0	210,9	345,5	350,3	-47,8	41,2	69,2	0,2	0,3	0,9	0,0	0,0	0,0	20:43:27
42,2	962,0	18,2	0,0	0,0	0,0	2,0	214,4	344,1	342,3	-49,0	42,6	71,3	0,2	0,3	0,9	0,0	0,0	0,0	20:43:42
42,3	962,0	18,3	0,0	0,0	0,0	1,0	217,1	345,2	335,2	-49,6	43,1	71,8	0,2	0,3	0,9	0,0	0,0	0,0	20:43:56
42,2	962,0	18,6	0,0	0,0	0,0	2,0	219,9	347,3	328,5	-49,9	43,2	72,1	0,2	0,3	0,9	0,0	0,0	0,0	20:44:10
42,3	962,0	18,4	0,0	0,0	0,0	2,0	221,8	347,2	325,4	-50,1	43,4	73,0	0,2	0,3	0,9	0,0	0,0	0,0	20:44:24
42,5	962,0	18,7	0,0	0,0	0,0	1,0	222,6	348,4	321,0	-49,9	43,2	73,2	0,2	0,3	0,9	0,0	0,0	0,0	20:44:38
42,2	962,0	18,2	0,0	0,0	0,0	2,0	222,3	347,5	317,2	-49,6	42,7	73,1	0,2	0,3	0,9	0,0	0,0	0,0	20:44:52
42,1	962,0	18,1	0,0	0,0	0,0	2,0	223,8	348,5	315,8	-50,0	43,3	73,3	0,2	0,3	0,9	0,0	0,0	0,0	20:45:06
42,1	962,0	18,3	0,0	0,0	0,0	1,0	223,0	348,0	313,5	-50,0	43,4	73,5	0,2	0,3	0,9	0,0	0,0	0,0	20:45:21
42,0	961,9	18,6	0,0	0,0	0,0	1,0	222,9	348,7	312,0	-49,5	42,8	72,5	0,2	0,3	0,9	0,0	0,0	0,0	20:45:35
41,8	961,9	18,7	0,0	0,0	0,0	2,0	223,1	352,0	312,7	-50,0	43,0	73,3	0,2	0,3	0,9	0,0	0,0	0,0	20:45:49
41,7	962,0	18,4	0,0	1,0	1,0	2,0	222,2	352,3	311,8	-50,1	42,8	73,5	0,2	0,3	0,9	0,0	0,0	0,0	20:46:03
41,7	962,0	18,5	0,0	1,0	1,0	3,0	221,8	351,0	309,6	-50,0	42,5	73,0	0,2	0,3	0,9	0,0	0,0	0,0	20:46:17
41,6	961,9	18,7	0,0	1,0	1,0	2,0	221,7	351,3	308,2	-50,2	42,7	73,4	0,2	0,3	0,9	0,0	0,0	0,0	20:46:31
41,6	961,9	18,5	0,0	0,0	0,0	2,0	221,8	353,0	307,5	-50,0	42,8	73,5	0,2	0,3	0,9	0,0	0,0	0,0	20:46:45
41,7	962,0	18,6	0,0	0,0	0,0	2,0	221,9	353,0	311,4	-50,1	42,6	72,8	0,2	0,3	0,9	0,0	0,0	0,0	20:47:02
41,7	961,9	18,5	0,0	0,0	0,0	1,0	219,7	353,5	311,5	-50,2	42,4	73,2	0,2	0,3	0,9	0,0	0,0	0,0	20:47:20
41,8	961,9	18,6	0,0	0,0	0,0	2,0	219,6	355,1	313,2	-50,3	42,5	74,0	0,2	0,3	0,9	0,0	0,0	0,0	20:47:37
41,7	961,9	18,6	0,0	0,0	0,0	2,0	219,1	356,5	316,2	-50,2	42,5	73,7	0,2	0,3	0,9	0,0	0,0	0,0	20:47:58
41,8	961,9	18,4	0,0	1,0	1,0	3,0	219,8	356,1	317,5	-50,6	42,5	74,0	0,2	0,3	0,9	0,0	0,0	0,0	20:48:12
41,6	961,9	18,5	0,0	1,0	1,0	2,0	221,5	354,4	317,7	-50,5	42,5	73,5	0,2	0,3	0,9	0,0	0,0	0,0	20:48:30
41,7	961,9	18,4	0,0	1,0	1,0	3,0	221,0	348,7	320,6	-50,2	42,4	73,3	0,2	0,2	1,0	0,0	0,0	0,0	20:48:49
41,6	962,0	18,2	0,0	1,0	1,0	3,0	221,0	346,0	322,6	-50,2	42,3	73,0	0,2	0,3	0,9	0,0	0,0	0,0	20:49:06
41,6	961,9	18,5	0,0	0,0	0,0	2,0	221,0	350,4	314,2	-50,4	42,5	71,9	0,2	0,3	0,9	0,0	0,0	0,0	20:49:23
41,6	961,9	18,7	0,0	0,0	0,0	1,0	218,4	356,4	309,1	-50,2	42,9	72,5	0,2	0,3	0,9	0,0	0,0	0,0	20:49:42

# Naložení s daty

- údaje z obrázků necháme ještě jednou přepočítat
- data zanalyzujeme
- vytvoříme čtyřstránkovou zprávu



**Děkujeme vám za pozornost!**

Nyní máme prostor pro dotazy!

## Bibliografie I

- [1] B. Dwyer et al. Roboflow. 2022. URL: <https://roboflow.com>.
- [2] Edje Electronics. *How to Train TensorFlow Lite Object Detection Models Using Google Colab*. Youtube. 2023. URL: <https://www.youtube.com/watch?v=XZ7FYAMCc4M>.
- [3] Jacob Høxbroe Jeppesen et al. „A cloud detection algorithm for satellite imagery based on deep learning“. In: *Remote sensing of environment* 229 (2019), s. 247–259.



Raspberry Pi



FAKULTA APLIKOVANÝCH VĚD  
ZÁPADOČESKÉ UNIVERZITY  
V PLZNI