Machine Learning in Velo LHCb monitoring and calibration in Run I and II.

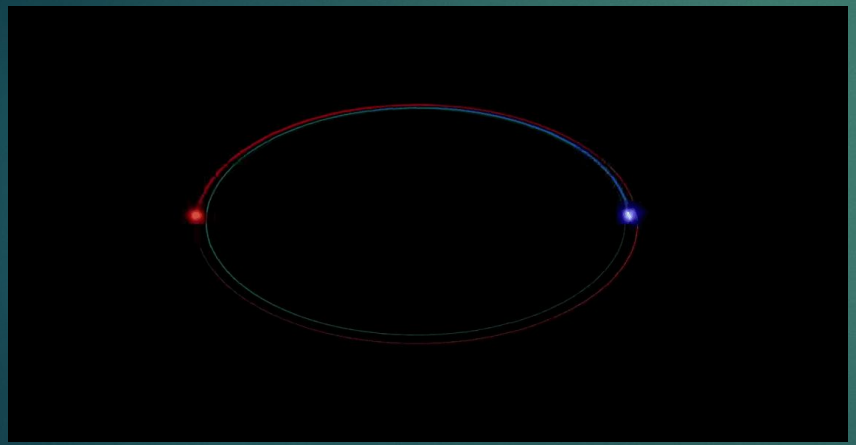
MACIEJ W. MAJEWSKI, MATEUSZ KARPIK – AGH UST

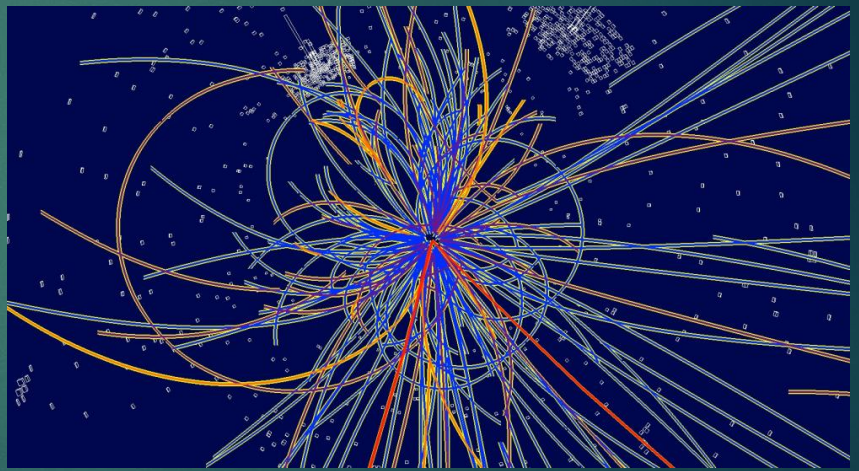
Agenda

- LHCb
- VeLo in LHCb
- Calibration of Velo
- Anomaly detection and Velo calibration modelling
- Velo and calibration forecasting
- Future of Velo

CERN and LHC (Large Hadron Collider)

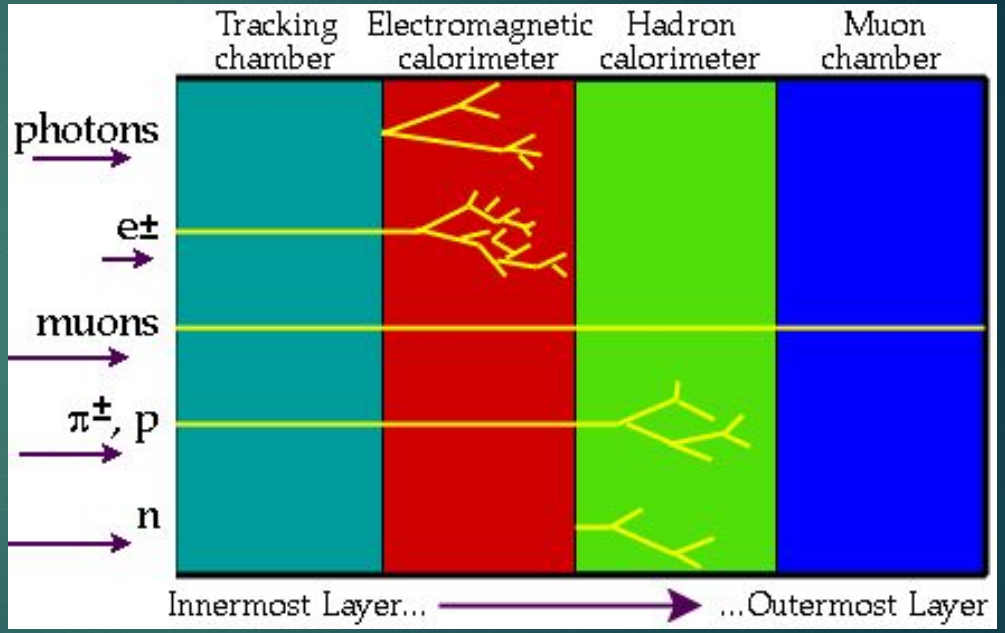
The collisions



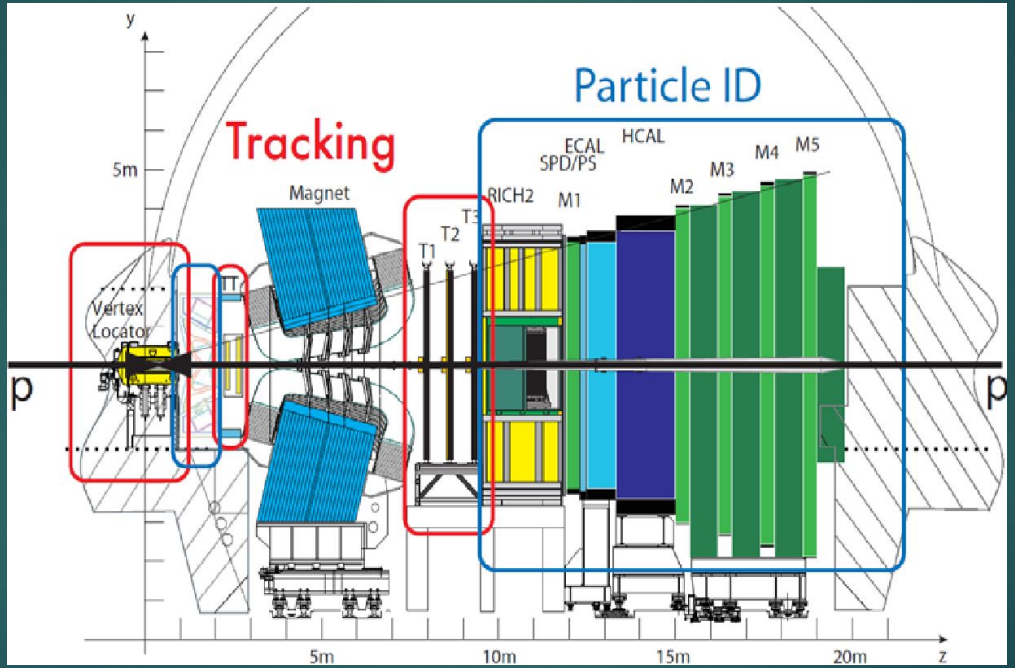


Seeing the particles

- Different particles can be detected by different components
- No single best „sensor”
- Most detectors are made of layers (subsystems)



Tracking nad Identification

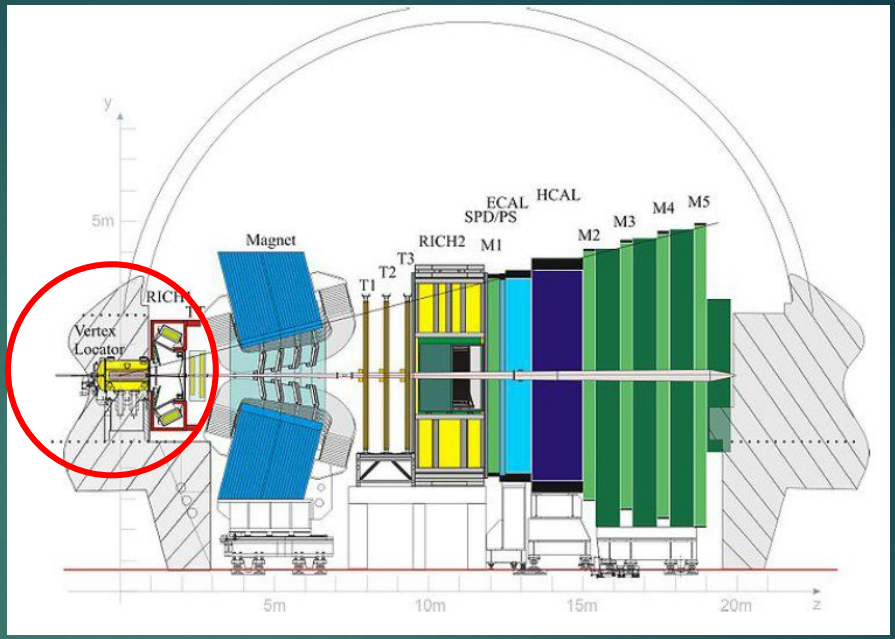


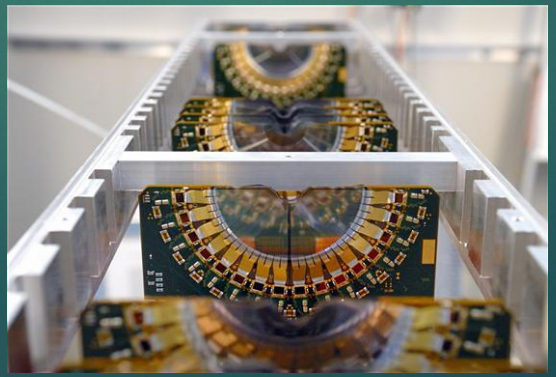
Velo in LHCb

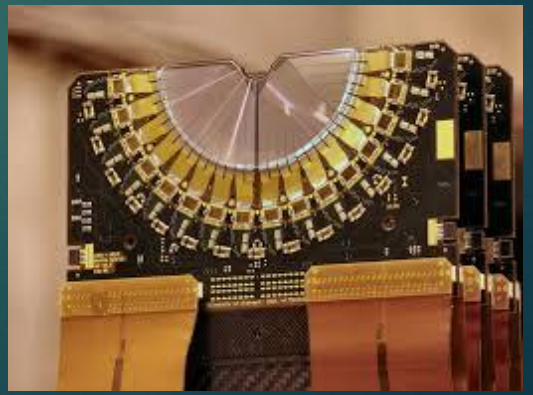
Vertex locator == VELO:

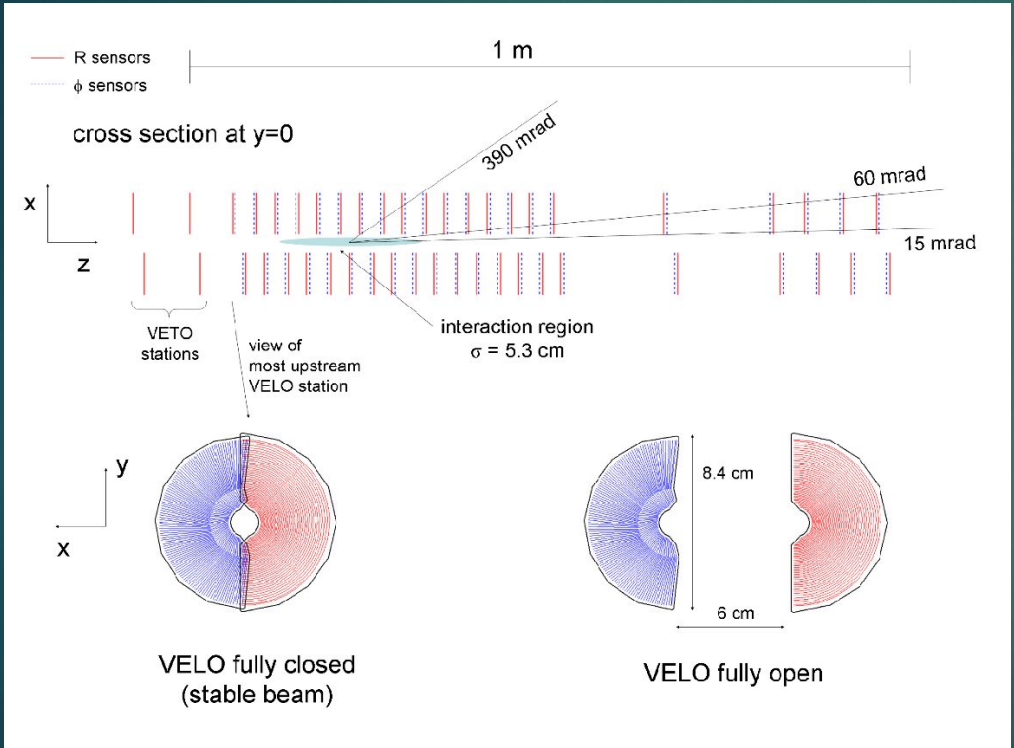
particle tracker located close to the beam

VELO consists of 42 modules





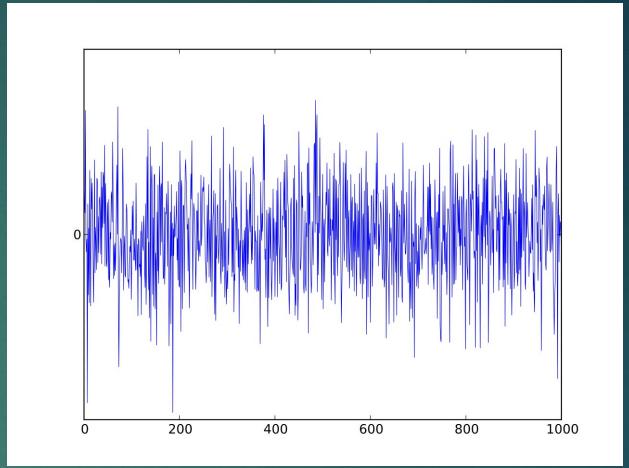


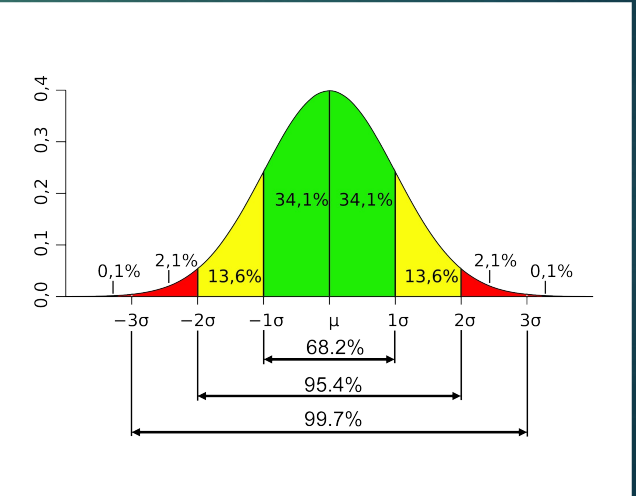


- Each contains silicon strips
- each module has 2 sensors
- R and Phi type (polar coordinate system)
- 2048 channels for each sensor
- the alignment of the modules depicted on the left

Velo Calibration

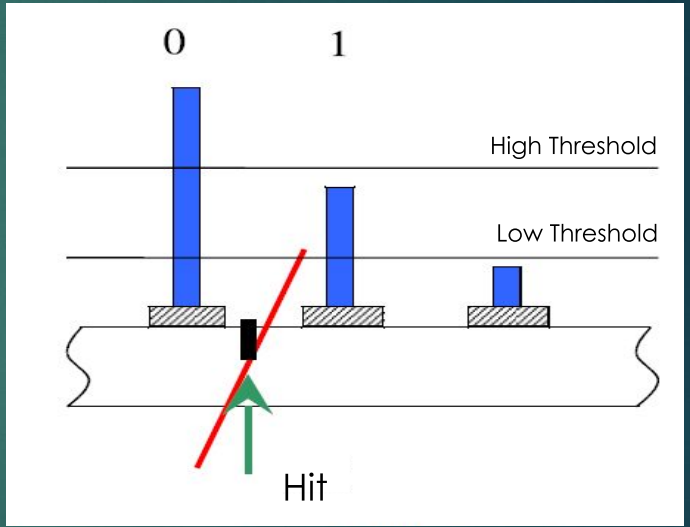
- During the no-beam, for each of the strips, we listen for noise coming from the signal.
- The distribution of the noise of the strip signal produces two important variable;
  - Mean of noise
  - Standard deviation
- Those values are calculated per channel/strip



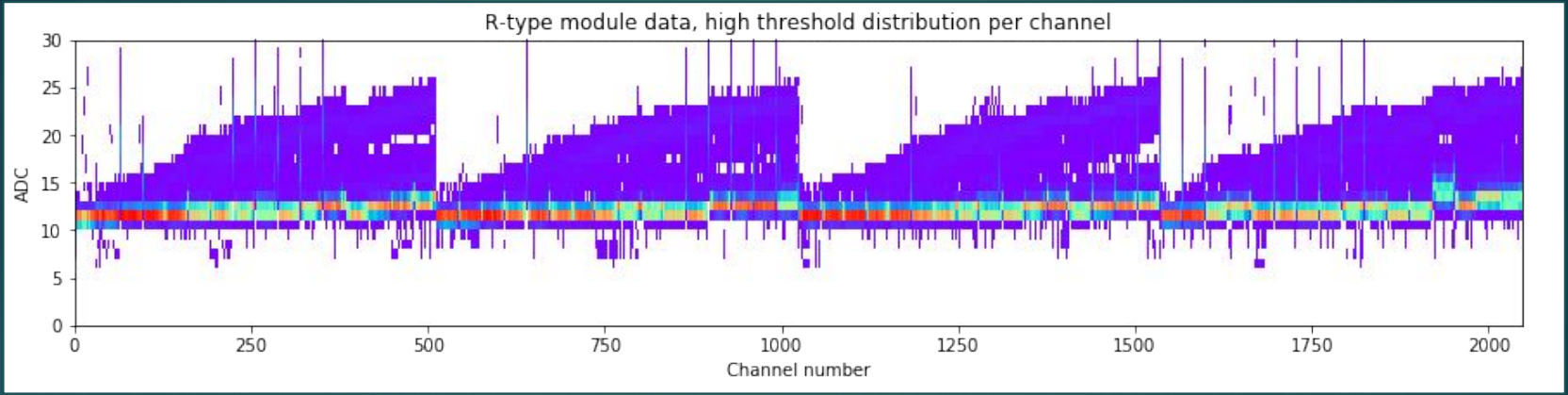


Velo Calibration

- We used calculated parameters to „denoise" the readout
- What we do is we use thresholds
- High threshold is 5σ
- Low Threshold is 60% of High Threshold
- The clustering algorithm uses them to indicate the hit
- We also use centers of the distributions to properly level the calibration - those are pedestals



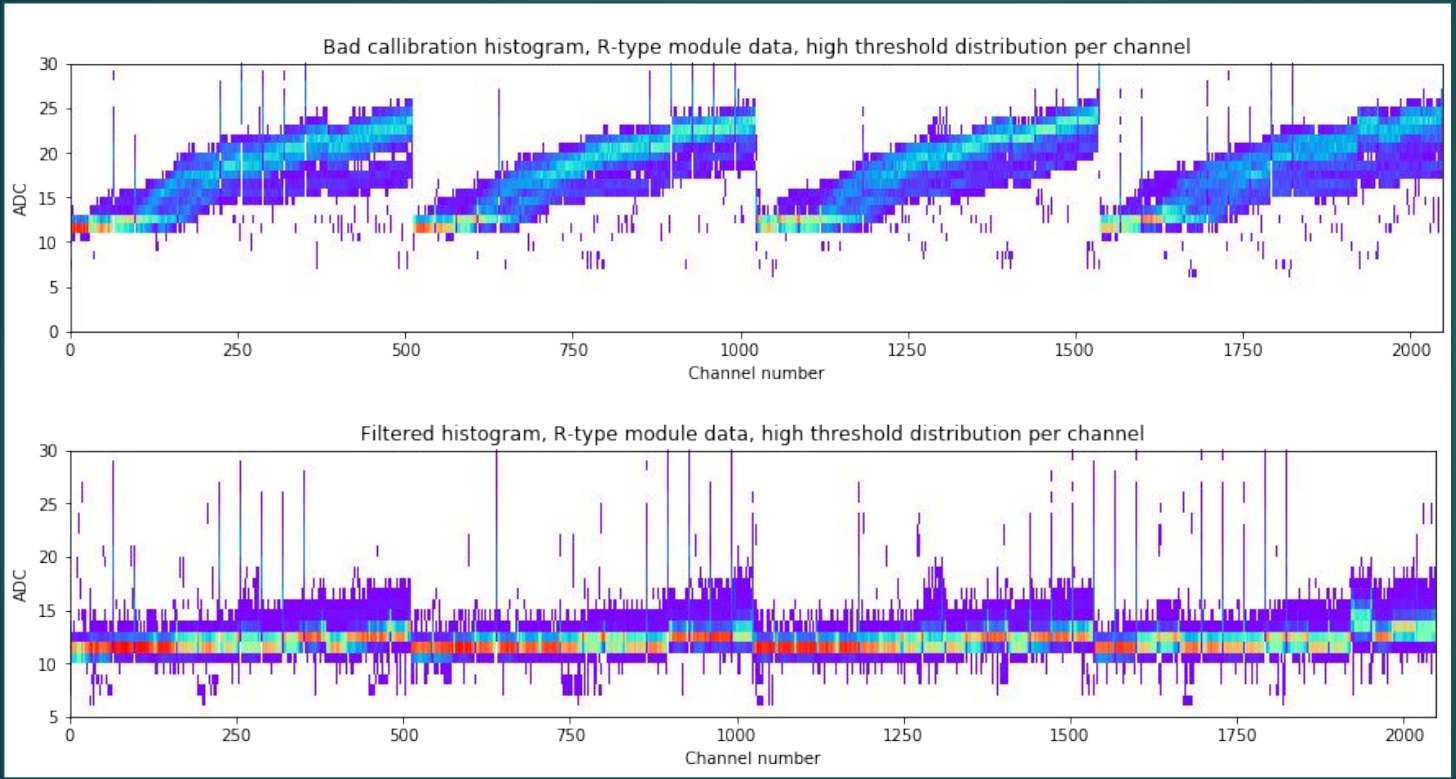
Velo Calibrations



- This is 2D histogram
- You can see „ghosting" effects in the top
- Those are 2 „bad" calibrations
- We separate them

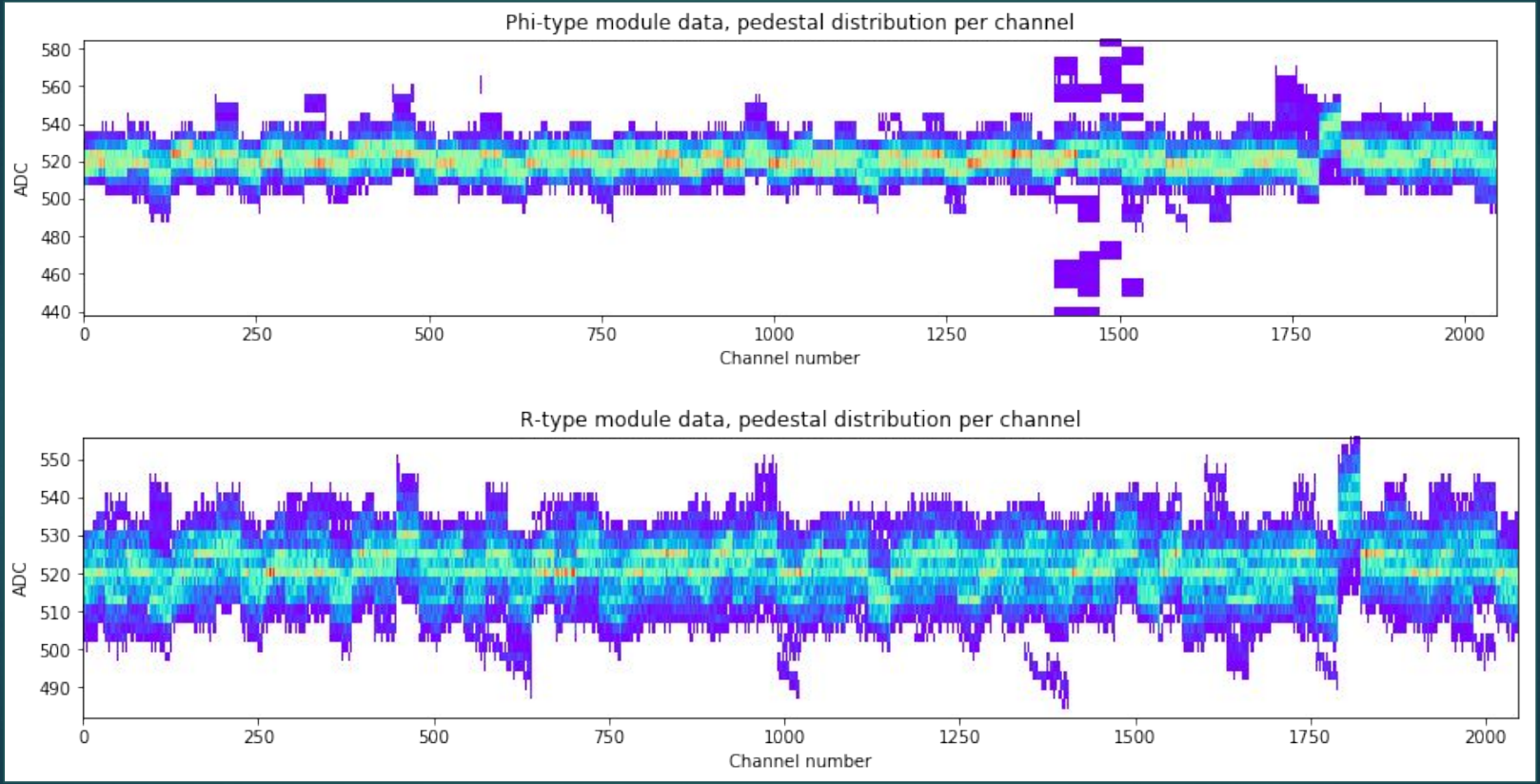
When I say "bad" or "good" calibration, i mean outlying, or non-outlying calibraiton.

Velo Calibration – separation



- „Bad" calibrations
- All others

Velo Calibration - pedestals



Velo Calibration - model

Problems:

There is not that many Velo calibrations (about 32 in total).

Each calibration has **170 000** parameters.

You can split those data in many different ways:

- per sensors (R or Phi)
- per module (one of 42)
- per groups of channels
- or you can combine those e.g. Phi sensors of module 2-5, first 100 channels

How to create a model that would be able to check calibration in any of those dimensions?

Velo Calibration - model

So in order to get meaningful insight we have to use something very smart; Probabilistic Programming.

It is very useful when we have good knowledge of underlying process.

It is **whitebox** kind of machine learning.

We can get inference from the model, but also we can generate cases.

This was implemented using library called Pymc3.

Two models:

- One for "Good" (non anomaly) calibration
- Second one for all of the calibrations

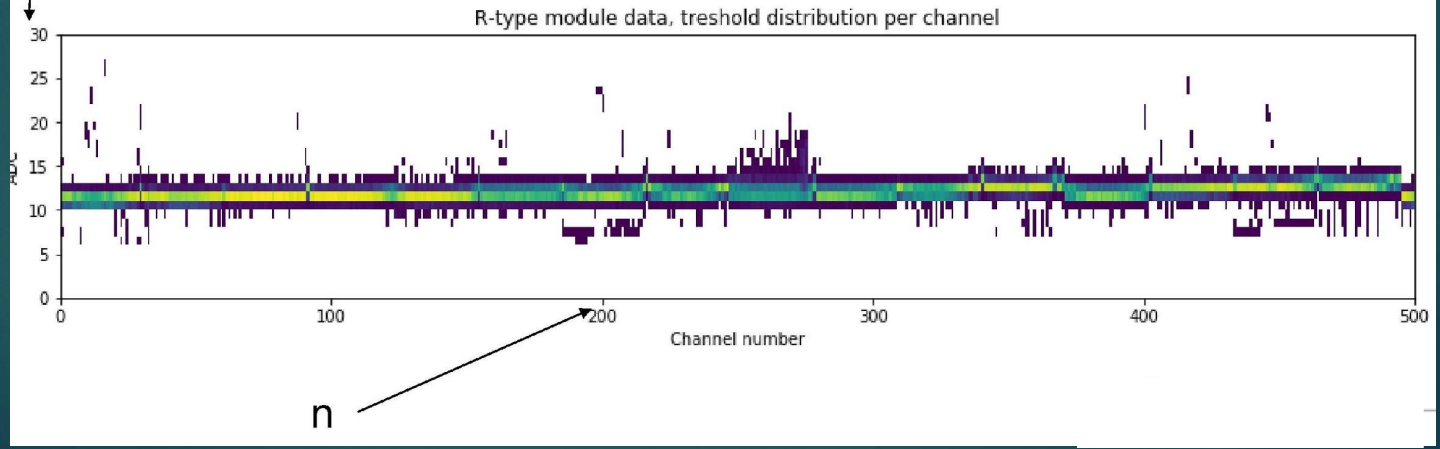
We use only one type of sensor for now: "R". We treat all 42 modules as exactly the same.

Velo Calibration - model

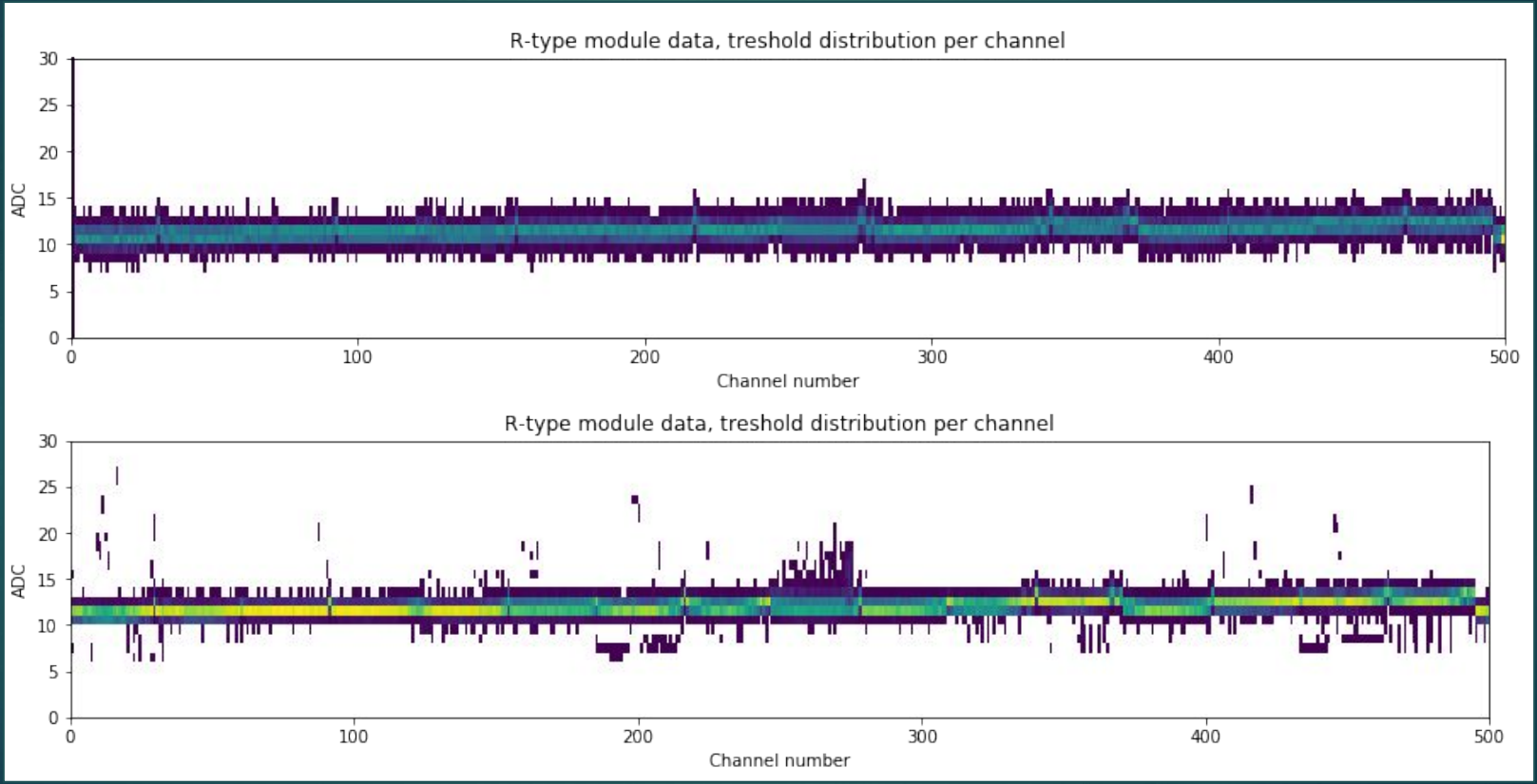
T_n - Threshold value for channel number n (only „good” callibrations!!)

$T_n \sim Gaussian(\mu = \mu_n, \sigma = \sigma_n)$

$n \in [0,1983]$ (excluded header-crosstalk)



Velo Calibration - modeling



Velo Calibration - modeling

- Ok, so we can model “good” calibrations
- How about anomalies?
- How to express level of anomaly?

Velo Calibration - model

T'_n - Threshold value for channel number n (**all** callibrations!!)

x - Artificial linear metric for callibration conditions (how much it differs from other callibrations). The higher this value is, the more it differs from „normal" cllibration.

y= ax+b

$$T'_n \sim Gaussian(\mu = (x * \mu'_n + \mu_n), \sigma = (x * \sigma'_n + \sigma_n))$$

$$T'_n \sim Gaussian(\mu = (x * \mu'_n + \mu_n), \sigma = (x * \sigma'_n + \sigma_n))$$

What this model says: The distribution of this parameter in N-th sensor is Gaussian, with the center in μ that is linearly dependent on some state X (same for every sensor, at a given time), with the constant slope μ'n (that is inner characteristic of a given N-th sensor) and intercept of μn (as well an inner characteristic of a given N-th sensor).

And sigma is analogical.

Velo Calibration – complete model

$$T_n \sim Gaussian(\mu = \mu_n, \sigma = \sigma_n)$$

$$T'_n \sim Gaussian(\mu = (x * \mu'_n + \mu_n), \sigma = (x * \sigma'_n + \sigma_n))$$

Unknown

Total dataset	
Calibration date	**X**
2011-03-07	1
2012-08-02	3
2012-07-30	10
2012-08-01	10
All others (already calculated)	0

Previous model already gave us some of the parameters

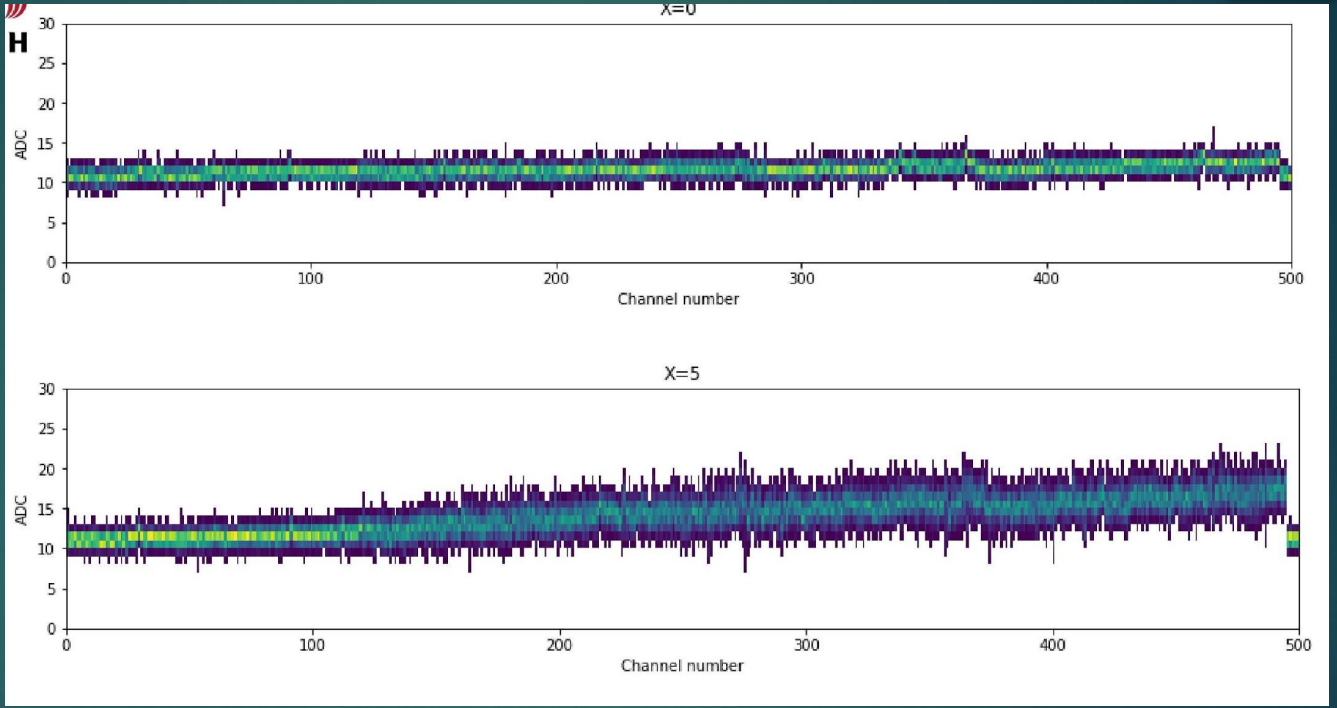
Velo Calibration – complete model

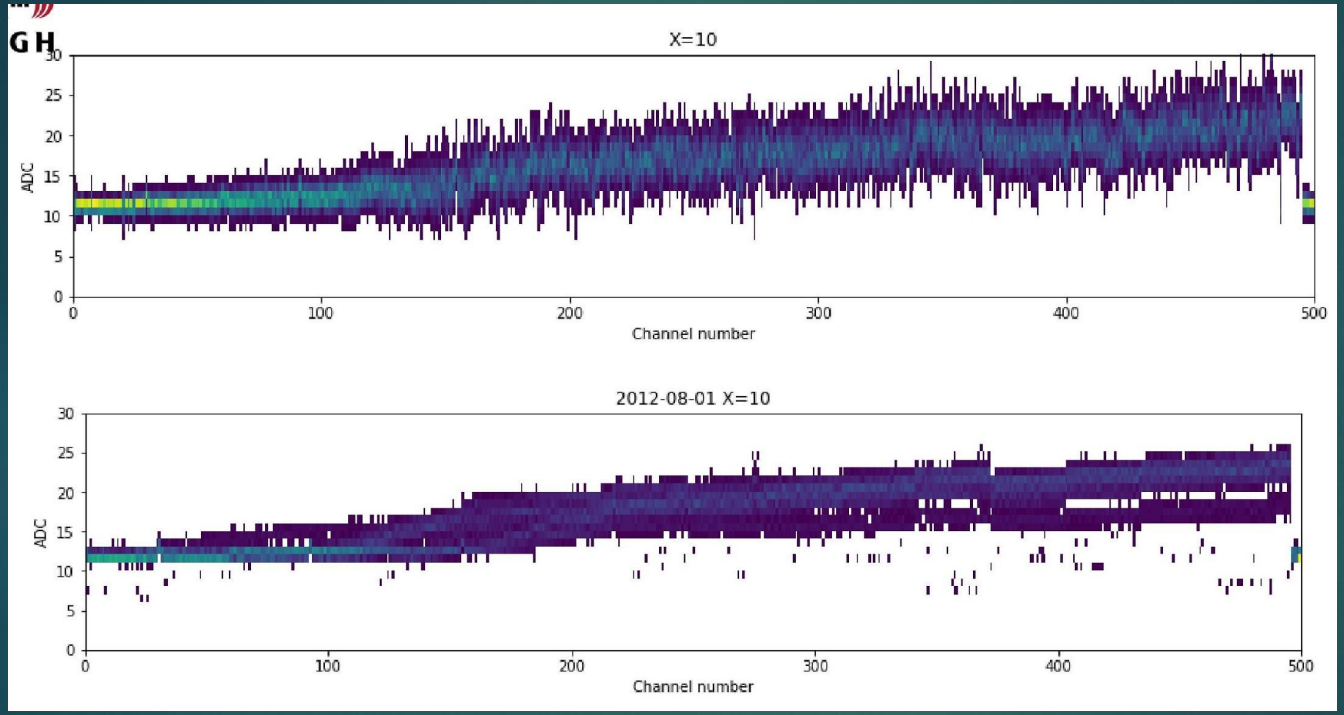
- If we learn the μ'n, μn, σ'n, σn then the model is complete.

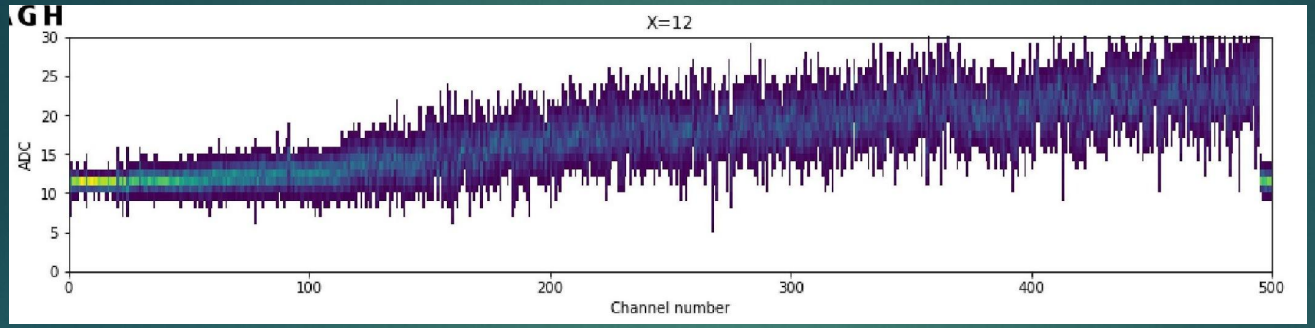
$$T'_n \sim Gaussian(\mu = (x * \mu'_n + \mu_n), \sigma = (x * \sigma'_n + \sigma_n))$$

Velo Calibration – complete model

- So given the actual calibration we can calculate the parameter X, that indicates anomalies in that particular calibration.
- The higher the X value, the higher is the chance of calibration being an anomaly.
- The “X” is also called “Outlierness”.


X=0
H
ADC
Channel number
X=5
ADC
Channel number


X=10
ADC
Channel number
2012-08-01 X=10
ADC
Channel number

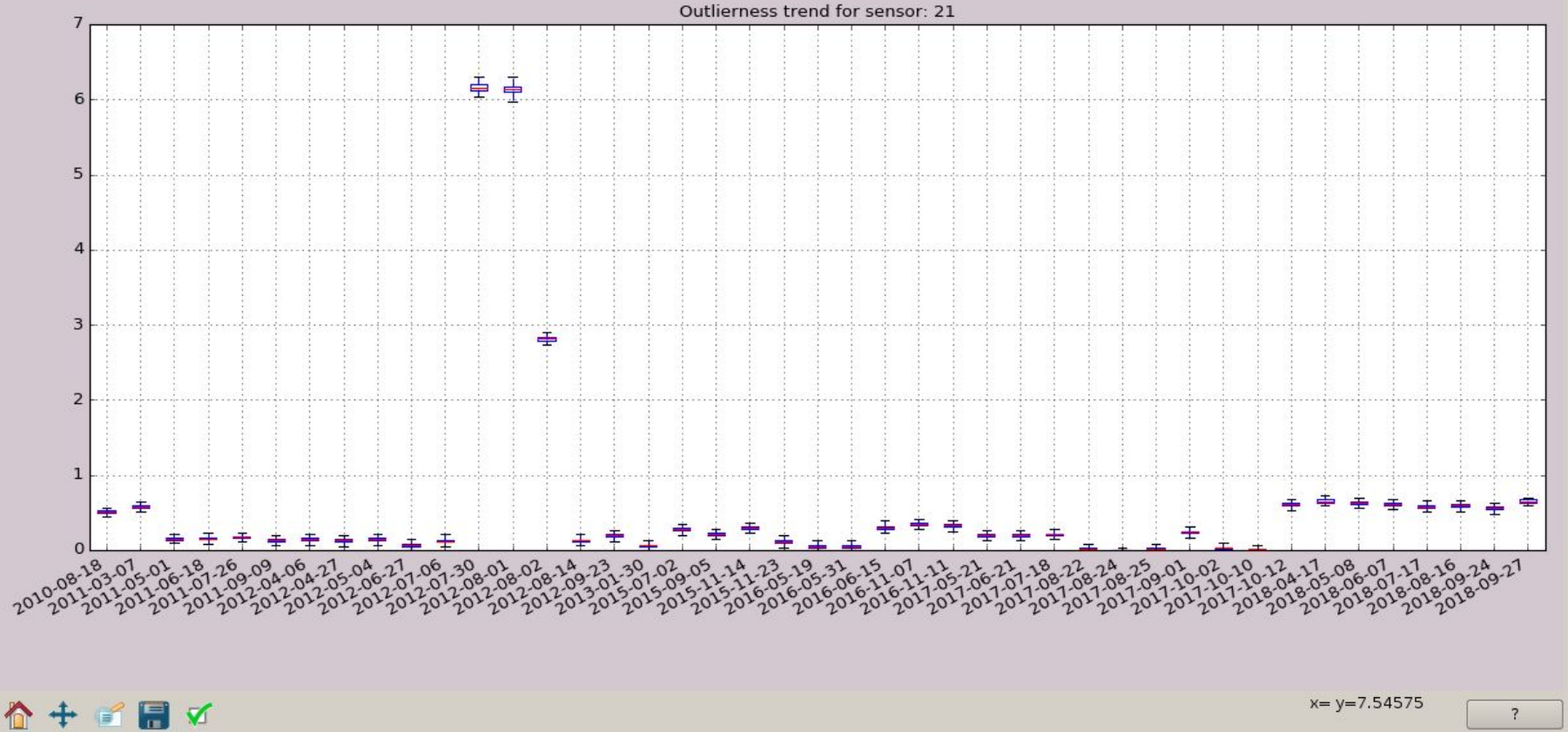

X=12
ADC
Channel number
0
5
10
15
20
25
30
0
100
200
300
400
500

Lovel - monitoring for velo

- This model was added in final months of the last run to the monitoring sotware
- (Screenshots next)

TELL1 | Special analyses

Pedestal | Hit threshold | low threshold | outlierness trend R | outlierness trend sensor | outlierness trend phi | outlierness noise

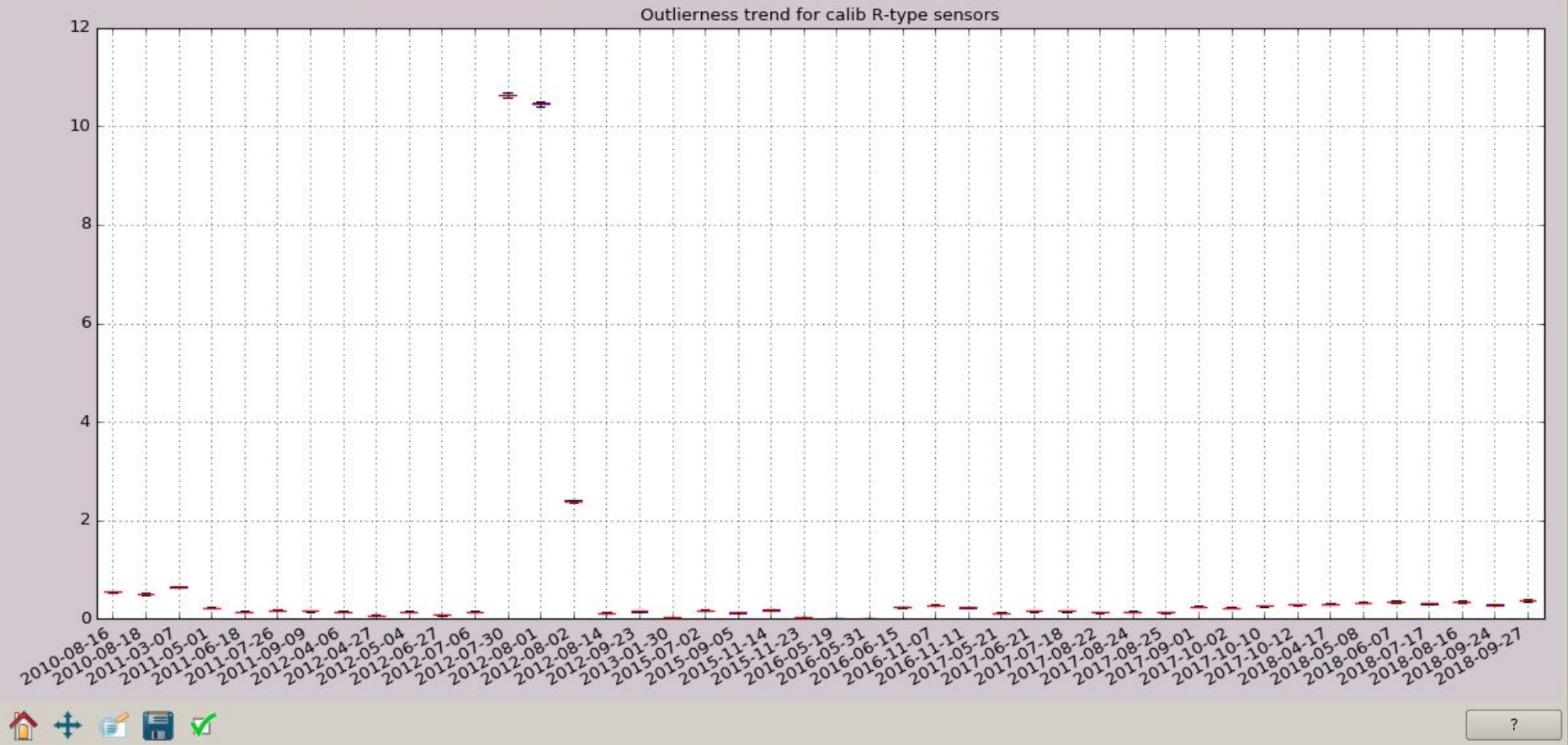


x= y=7.54575

?

TELL1 | Special analyses

Pedestal | Hit threshold | low threshold | outlierness trend R | outlierness trend sensor | outlierness trend phi | outlierness noise



?

Velo Calibration – complete model

- We can tell what is the anomaly level of the calibration
- … of module, or group of channels (!!)
- We can focus on groups of strips
- If the value is high, then the detector might need recalibration
- Actually worked in few last months before technical stop
- In principle, this should be aplicable to new pixel Velo (more on this later)

Software: https://gitlab.cern.ch/mmajewsk/calina

Calibration forecasting

- In principle the calibration should be done as often as possible
- Because this process is time consuming and requires specific conditions there are some restrictions on calibration, especially after the planned upgrade
- So in reality the calibration is conducted when it is needed
- Analysing a lot of parameters in order to determine whether calibration is needed might be automated

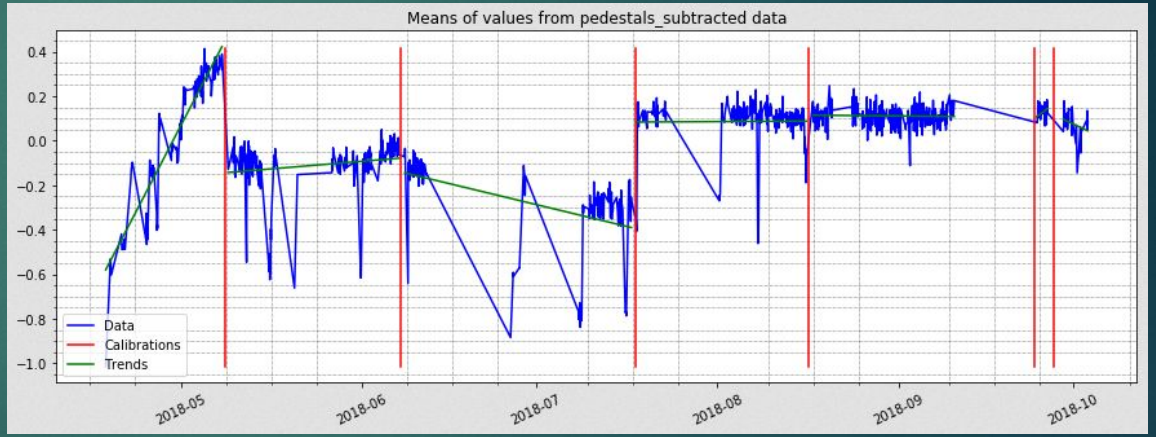
Calibration forecasting

Lets automate it with machine learning:

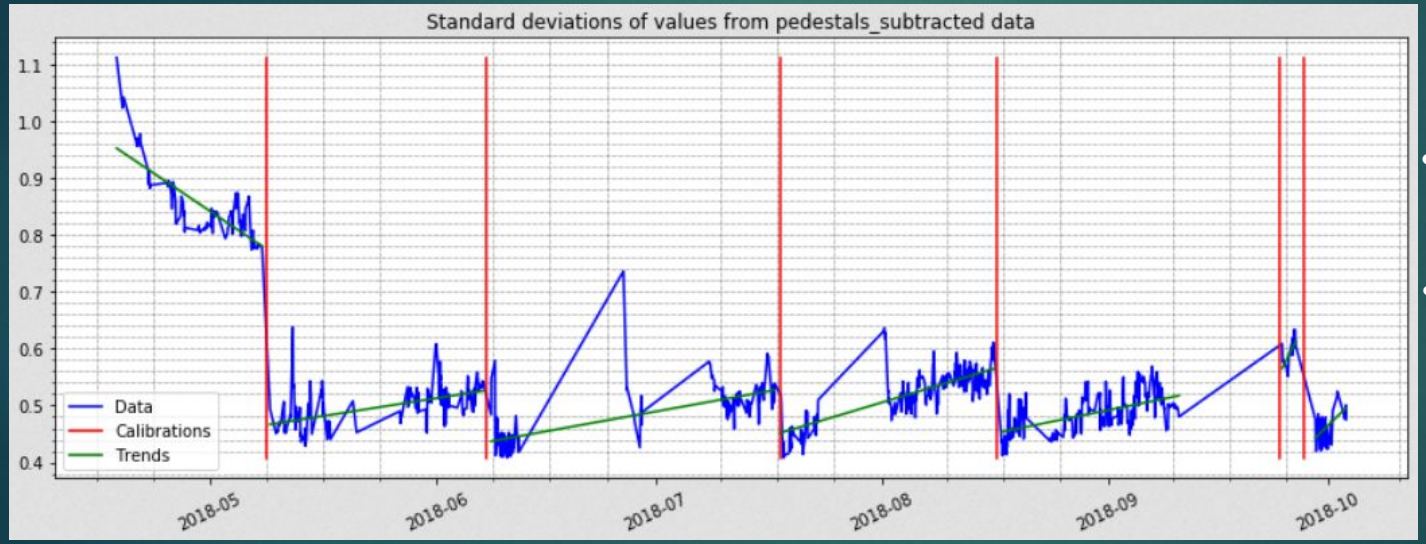
- We assume that our dataset contains perfectly timed calibrations.
- We are basing our system on the raw signal from the detector and calibration.

Means from raw data with pedestals substracted - 2018

Mean of all pedestals subtracted.
This shows the relations of pedestals to calibration.
But we can't really see the calibrations.



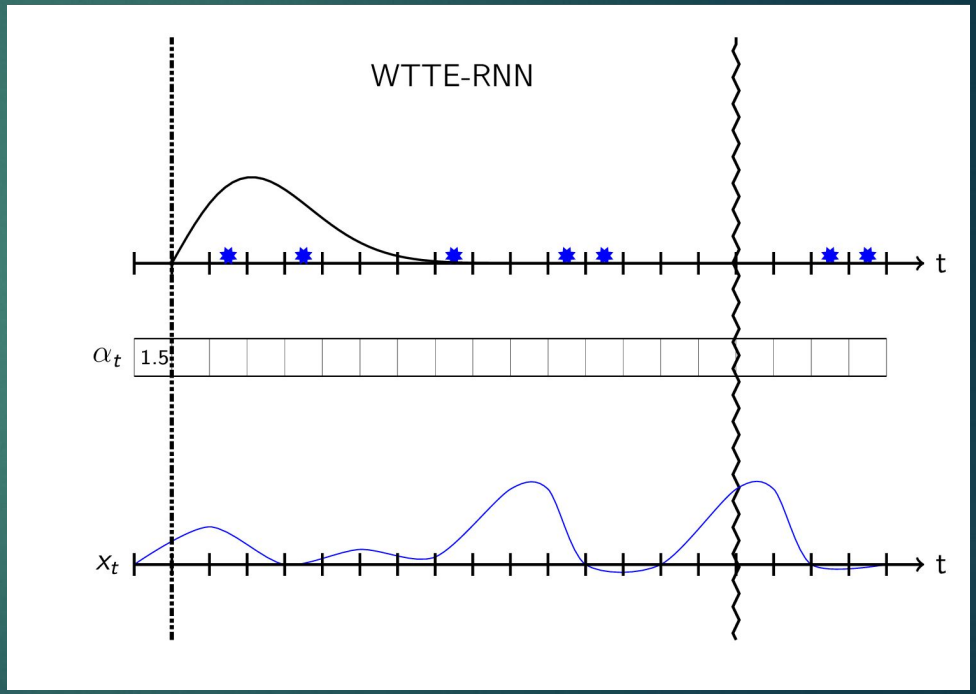
Standard deviation from pedestals substracted data - 2018



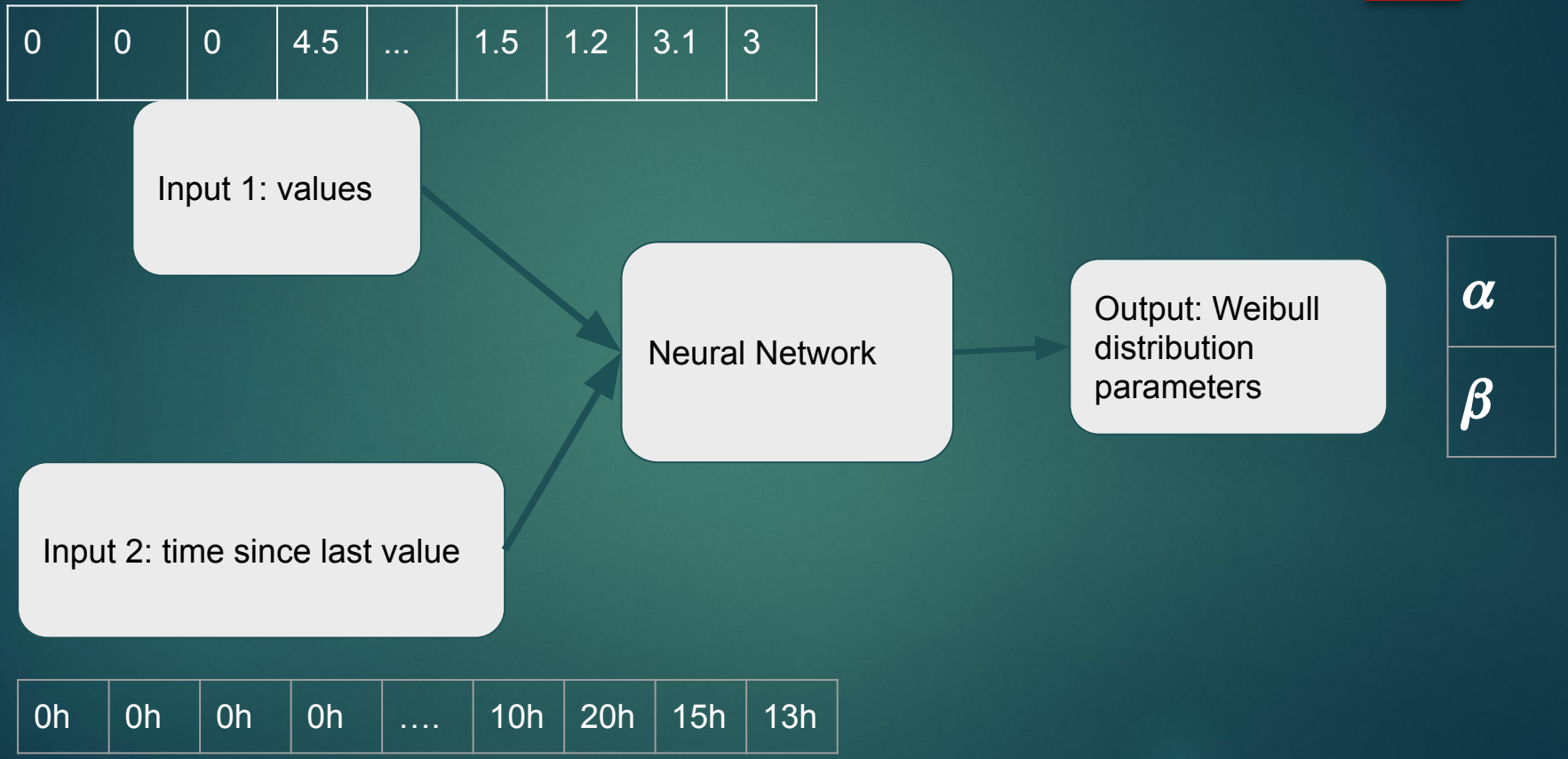
- We can somewhat see the calibrations happening in the spikes of this parameter
- **If we can do this, we could also be able to predict calibration**
- Assuming that our calibration expert has never made any mistake

Calibration prediction

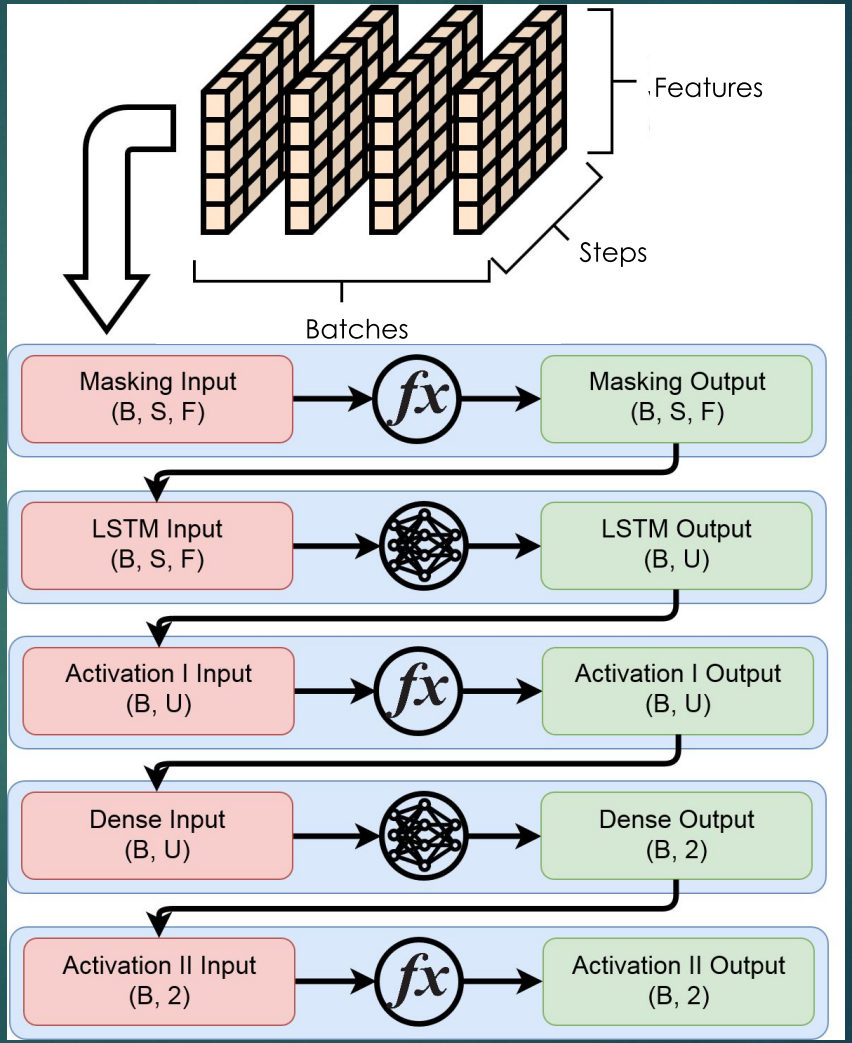
- WTTE-RNN
- Weibull Time To Event Recurrent Neural Network
- We use weibull distribution to set the probability of occurance of an event
- In this case callibration
- **We assume that the calibrations done in the past were done with perfect timing**



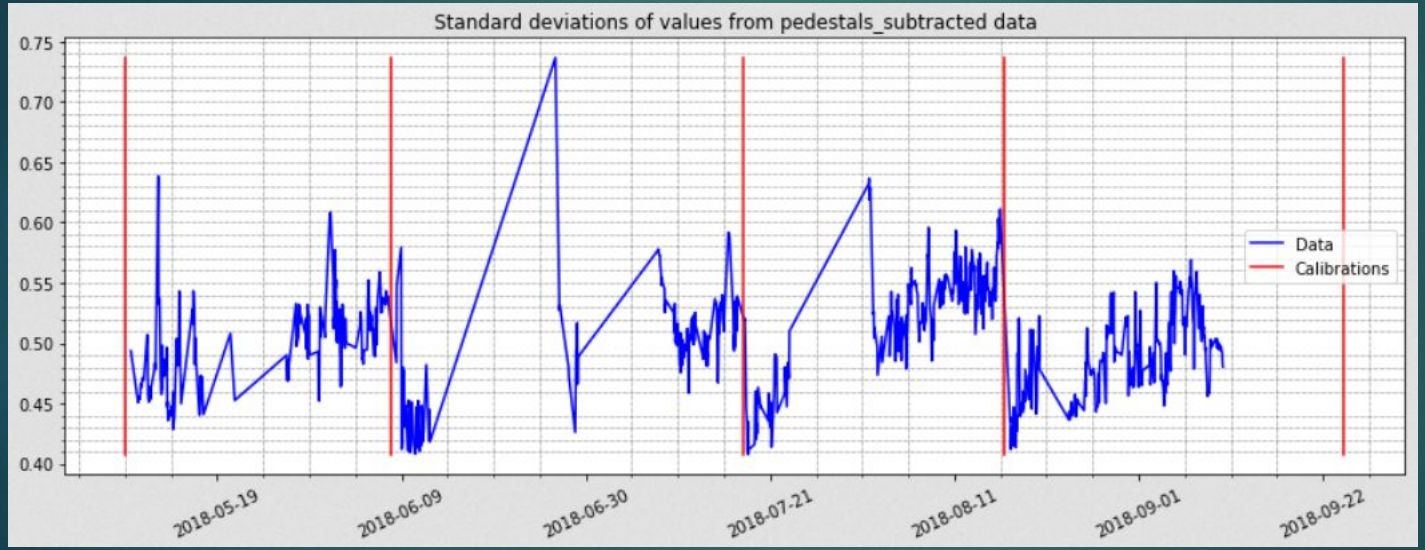
Calibration prediction - I/O



The model architecture

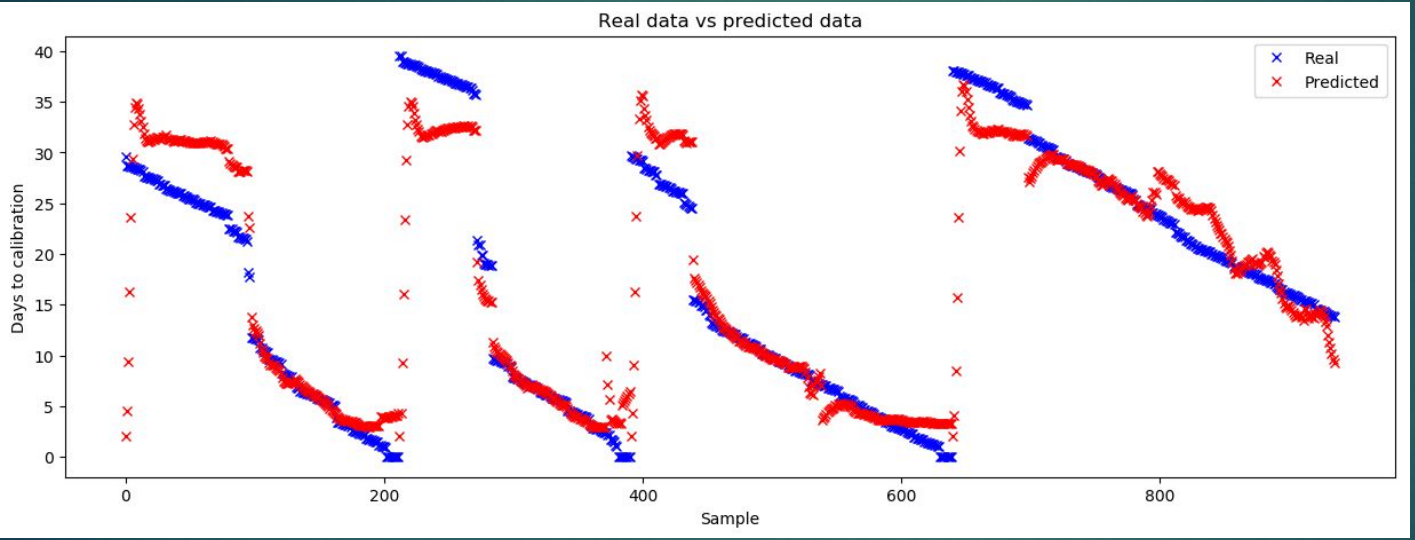


Training Data



Data after 2018-09 is test data

Results of calibration prediction– all data



- We predict DTE - days to event, in this case days to calibration
- We can predict calibration, basing only on the pedestal subtracted data!

Raw pedestals, and calibration prediction

- The WTTE method makes for a good candidate for inteligent calibration triggering in the future.
- We are still studying per-module WTTE behaviour
- The standard deviation of pedestals subtracted, was a good indicator for calibration -> analogical values in pixel Velo should be considered.
- **This work is as well still undergoing**
- Based on "WTTE-RNN : Weibull Time To Event Recurrent Neural Network"
  Mateusz Karpik

Future

Velo Pix:

All of the presented results are basing on model of velo that worked in Run I and II.

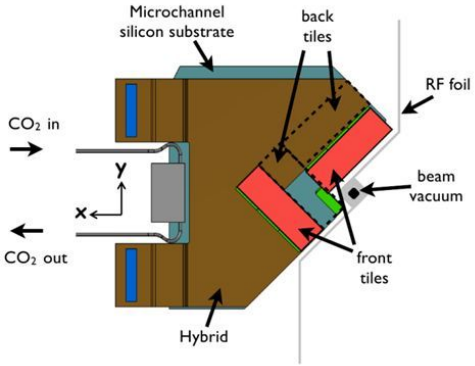
VeloPix will take place of Velo in LHCb.

All of this work was created with keeping in mind the new VeloPix.

The number of pixels in new VeloPix ~ 65 000 000

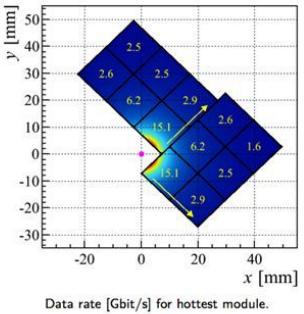
VeloPix ASIC module (12 ASICs)

Module of 12 VeloPix ASICs [1]:



See Sophie Richards' overview of the VELO upgrade: "LHCb VELO Upgrade"

Track rates for module [2]:



Data rate [Gbit/s] for hottest module.

Highly non-uniform radiation dose: 8×10^{15} to 2×10^{14} n_{eq}/cm^2

Peak rates:
Hottest chip 15.1 Gbits/s
hottest module: 61.2 Gbits/s

Agenda

- ✅ LHCb
- ✅ VeLo in LHCb
- ✅ Calibration of Velo
- ✅ Anomaly detection and Velo calibration modelling
- ✅ Velo and calibration forecasting
- ✅ Future of Velo

The end.

The end.

Velo Calibration - model


```
def score(self, x: np.ndarray, draws: int = 800, tune:int =200, **kwargs) -> MultiTrace:
    """
    :param x: np.ndarray of shape (n,m), m dimensions is irrelevant
    :param draws: passed into pm.sample()
    :param tune: passed into pm.sample()
    :param kwargs: passed into pm.sample()
    """
    kwargs['progressbar'] = kwargs.get('verbose', self.verbose)
    with pm.Model() as model:
        outlierness = pm.Uniform('x', 0, 20)
        observations = pm.Normal(
            'obs',
            mu=self.centers + self.k * outlierness,
            sd=self.sds + self.m * outlierness, observed=x
            )
        step = pm.Metropolis(vars=[outlierness])
        trace = pm.sample(draws=draws, tune=tune, step=step, **kwargs)
    return trace
```

Velo Calibration - model


```
with pm.Model() as model:
    sds = pm.Uniform("sds", 0, 0.75, shape=positive.shape[1])
    centers = pm.Uniform("centers", 0,50, shape=positive.shape[1])
    observations = pm.Normal("obs", mu=centers, sd=sds, observed=positive)


with model:
        step = pm.Metropolis(vars=[sds,centers, observations])
        trace = pm.sample(**settings, step=step, njobs=1, chains=1)
pm.backends.text.dump(args.dump_path, trace)
```

T_n - Threshold value for channel number n (only „good” callibrations!!)

$T_n \sim Gaussian(\mu = \mu_n, \sigma = \sigma_n)$

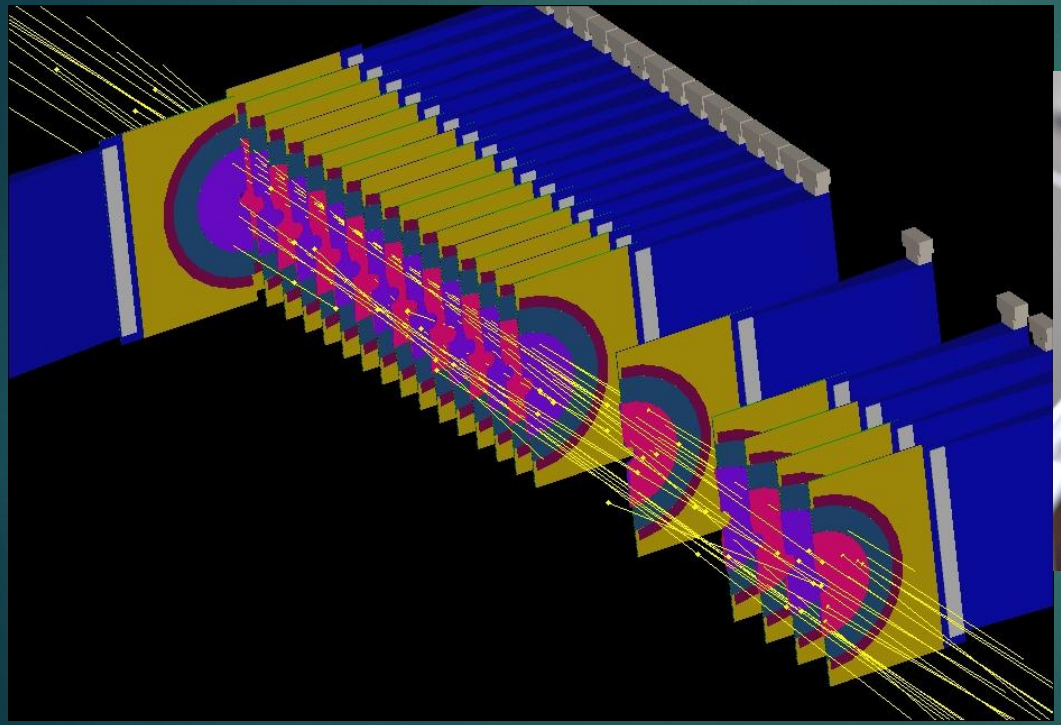
$n \in [0,1983]$ (excluded header-crosstalk)

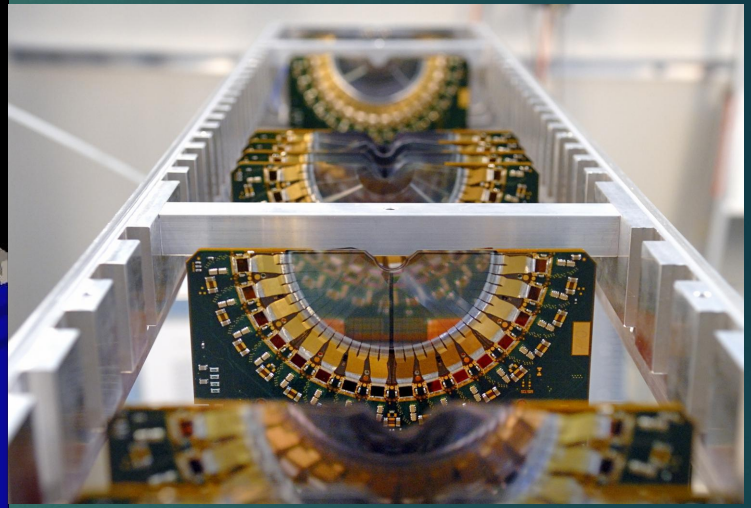
✔ Particle Detectors

- Detectors are composed of subsystems
- We record tracks (paths of the particles)
- As well as type (Energy)
- But it is hard to do both
- Detector's sensors are simmilar to camera sensors

- ✔ Particle Physics
- ✔ Particle Detectors
- Machine Learning Stuff
  - Calibration and Anomaly Analysis
  - Problems of Tracking
  - GAN in Calorimetry
  - Where is most of the machine learning stuff in High Energy Physics (TMVA)
  - 3D imaging in LArTPC

Anomaly Detection - modules





Velo Calibration - model


```
with pm.Model() as model:
    sds = pm.Uniform("sds", 0, 0.75, shape=positive.shape[1])
    centers = pm.Uniform("centers", 0,50, shape=positive.shape[1])
    observations = pm.Normal("obs", mu=centers, sd=sds, observed=positive)


with model:
        step = pm.Metropolis(vars=[sds,centers, observations])
        trace = pm.sample(**settings, step=step, njobs=1, chains=1)
pm.backends.text.dump(args.dump_path, trace)
```

(Steps of modelling)

```
meancenters = trace['centers'].mean(axis=0)
meansds = trace['sds'].mean(axis=0)
pd.DataFrame(meancenters).to_csv(os.path.join(args.dump_path,"{}_meancenters.csv".format(args.sensor_type)), index=False)
pd.DataFrame(meansds).to_csv(os.path.join(args.dump_path,"{}_meansds.csv".format(args.sensor_type)), index=False)
```

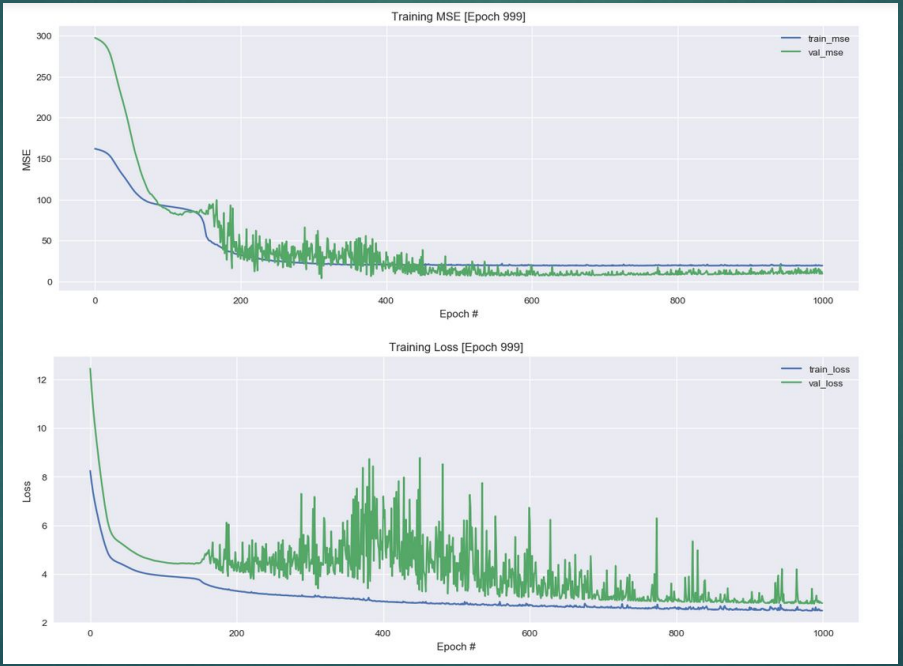
Velo Calibration - model


```
meancenters = trace['centers'].mean(axis=0)
meansds = trace['sds'].mean(axis=0)
pd.DataFrame(meancenters).to_csv(os.path.join(args.dump_path,"{}_meancenters.csv".format(args.sensor_type)), index=False)
pd.DataFrame(meansds).to_csv(os.path.join(args.dump_path,"{}_meansds.csv".format(args.sensor_type)), index=False)
```

```
meancenters = pd.read_csv(os.path.join(args.dump_path,"{}_meancenters.csv".format(args.sensor_type)))
meansds = pd.read_csv(os.path.join(args.dump_path,"{}_meansds.csv".format(args.sensor_type)))
with pm.Model() as model:
        x = pm.Uniform('x', 0,40, observed=xfactor[:,np.newaxis])
        k = pm.Uniform('k', -10,10,shape=negative.shape[1])
        m = pm.Uniform('m', -10,10,shape=negative.shape[1])
        observations = pm.Normal("obs", mu=meancenters.values.T[0]+k*x, sd=meansds.values.T[0]+m*x, observed=negative)

with model:
        step = pm.Metropolis(vars=[k,m, observations])
        trace = pm.sample(**settings, step=step, njobs=1, chains=1)
pm.backends.text.dump(os.path.join(args.dump_path,"slope"), trace)
```

$$T'_n \sim Gaussian(\mu = (x * \mu'_n + \mu_n), \sigma = (x * \sigma'_n + \sigma_n))$$


Training MSE [Epoch 999]
train_mse
val_mse
MSE
Epoch #
Training Loss [Epoch 999]
train_loss
val_loss
Loss
Epoch #