AGH University of Science and Technology
Faculty of Physics and Applied Computer Science

Doctoral thesis

Maciej Witold Majewski

Application of machine learning methods for the analysis of the calibration of the LHCb VELO detector, studies of irradiated silicon pixel sensors and reconstruction of the neutrino interaction in LARTPC detector

Supervisor: **dr hab. inż. Tomasz Szumlak**

Cracow, January 2021

Declaration of the author of this dissertation:

Aware of legal responsibility for making untrue statements I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

Declaration of the thesis Supervisor:

This dissertation is ready to be reviewed.

Application of machine learning methods for the analysis of the calibration of the LHCb VELO detector, studies of irradiated silicon pixel sensors and reconstruction of the neutrino interaction in LARTPC detector

Streszczenie

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

Quisque facilisis erat a dui. Nam malesuada ornare dolor. Cras gravida, diam sit amet rhoncus ornare, erat elit consectetuer erat, id egestas pede nibh eget odio. Proin tincidunt, velit vel porta elementum, magna diam molestie sapien, non aliquet massa pede eu diam. Aliquam iaculis. Fusce et ipsum et nulla tristique facilisis. Donec eget sem sit amet ligula viverra gravida. Etiam vehicula urna vel turpis. Suspendisse sagittis ante a urna. Morbi a est quis orci consequat rutrum. Nullam egestas feugiat felis. Integer adipiscing semper ligula. Nunc molestie, nisl sit amet cursus convallis, sapien lectus pretium metus, vitae pretium enim wisi id lectus. Donec vestibulum. Etiam vel nibh. Nulla facilisi. Mauris pharetra. Donec augue. Fusce ultrices, neque id dignissim ultrices, tellus mauris dictum elit, vel lacinia enim metus eu nunc.

Contents

Listing of figures

This is the dedication.

Acknowledgments

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

In the beginning there was nothing, which exploded.

Terry Pratchett

0 Introduction

In a brief moment, a universe happened. After the bing bang, the matter (leptons that form protons and neutrons) came to life in a time of about 2-3 human breaths. The currently known physics says that while this is certainly possible, an equal ammount of antimatter should be produced as well. Yet, we do not observe the universe to be symmetrical, cosmic ying-yan; equal parts matter, and anti matter. This is one of the greatest riddles of current physics. One of the goals of the LHCb experiment at CERN was to find and study the processes that can contribute to this assymetry. The grand promise of modern physics is the theory of everything; a universe boiled down to single theory, pheraps even to a single equation. This dream is partially realised by the Standard Model, the theory that describes three fundamental forces in physics. Weak nuclear, strong uclear, and electromagnetic force, come together into single equation. To some, it may be suprising that all of the mechanics in the know universe can be rooted just in a small theory. But the emergence of complex mechanism from simple rules is prevailant in science. Game of Life - a cellural automata simulation, althoug containg small set of rules can make increasingly complex structures. Most of the known chemistry that guides the biological processes, can be explained almost exclusevily "just" using the electrical potential and probability. As particle physicists, we see this emergence of complexity even at very basic level. Just by colliding two protons, we see a multitude of emergent possibilities, many different particles, and vast sea of probable combinations of fundemantal building blocks. Studying those combinations of possibilities is a bread and butter of modern particle physics. One of the misconceptions about High Energy Physics, is that it is like grandiose hunt through a murky and empty forrest for one big prey. In reality it is more like egg farming. Everyday we check new chicken coops, expeting eggs, and usually we find exactly that. On some days, we revisit old coops to be double sure that hens are still producing the eggs. But still, we hope, that there is this one chicken that makes gold eggs. And we can't deny it does not exist, until we check all of the coops.

This steady, monotone and patient work, of checking all of the branches of physics is facilitated by international community of scientist, of which CERN scientists constitute by far the largest group, with the biggest particle accelerator - LHC (Large Hadron Collider). The LHC is a circullar accelerator, located in the underground tunnel under Swiss-French border. The large portion of the physicist working at CERN are parts of four main experiments Alice, Atlas, CMS and LHCb. During my PhD studies I had a privilige to be one of the collaborators of the LHCb experiment. Such extraordinary science done with the LHCb spectrometer, couldn't be possible without years of preparation, and without overcoming technical challenges posed by the detectors hardware. Cooling system, electronics, mechanic design, all of the subsystem, and parts of the detector spawn new problems to be solved by the collaboration. I am glad to introduce the reader into my thesis, describing some of the solutions to those problems.

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo.

Quoteauthor Lastname

1 High Energy Physics

The idea of atoms - building blocks of nature stretch back as far as ancient greece, to Leucippus and Democritus[5], and prevailed through the ages, until the beginning of the 20th century, when it was solidified. In 1897 J.J.Thompson discovered electron using cathode ray and noticing it's negative charge[6]. Later in 1911 Rutherford discovered the nucleus using the gold foil scattering experiment[2]. Shortly after that in 1932 the model of the nucleus containing neutron and proton was developed[4][3]. This opened the - How energy physics came to life - The standard model - searches for physics at cern - cPT

1.1 The standard model

1.2 the proton proton collisions

1.3 CP parity

1.4 Tree-level determination of the unitarity triangle angle γ.

1.5 that problem with the proprtion of muon to electron

This is some random quote to start off the chapter.

Firstname lastname

2

The Large Hadron Collider beauty experiment

2.1 VELO

2.1.1 Velopix

THIS IS COPY PASTE WIP

2.1.1.1 Velo module

The Large Hadron Collider and it's detectors were planned to be incrementally improved over time. In Run III of the LHC, there will be a significant increase in the luminosity, and the VELO must addapt to that change. For this purpose the VeloPix ASIC was adapted from TimePix.

The new Velo for the Run III upgrade, has a modular design. Each module contains 12 VeloPix asics. VeloPix asics are rectangular silicon pixel detectors, 255x255 pixels, each pixel has rectangular shape with $55x55\mu m^2$ size. There is 52 modules, placed simillarilly as strip velo's module.

2.1.1.2 Signal, trims and clustering

The most drasti change in VeloPix operation is the goal of changing entirely the readout type from analog to digital. VeloPix has several modes of readout, but in the end the asynchronic hit position information will be the only signal coming from the readout electronics. But this requires proper internal calibration of the pixels.

2.1.1.2.1 VeloPix calibration

THIS SUBSECTION IS WIP Simillarilly to strip Velo, VeloPix will listen for the noise distribution. In case of the VeloPix, each of the pixels has its own voltage thresh-

old. If this threshold is exceeded in a pixel, we register a hit. There are two types of values that can be set in VeloPix to achievie optimal threshold: Global Threshold and Trim. Trims are indivudal values per each pixel. Trims are 4-bit (it has range of 16 values). Global Threshold GT is a voltage value that common for all of the pixels. Independantly of the GT value, all of the pixels can differ in sensitivity. First we set all pixels to use their $Trim_0$ value, and we scan the reaction of the pixels by moderating GT value, and holding it for constant ammount of time T_C. Then we repeat this operation for $Trim_{15}$. In both trim cases we record total number of active pixels (pixels that registered hit) $N_{HITS}(GT)$ as a function of GT, as well as the number of registered hits $H_{P_{x,y}}(GT)$ in each pixel $P_{x,y}$, in a GT.

The mean of the gaussian distribution of hits $H_{P_{x,y}}$ for each of the steps of GT is used to calculate lower bound for trim $TrimValue_{0_{x,y}}$ in case of $Trim_0$, as well ass to calculate $TrimValue_{15_{x,y}}$ in $Trim_{15}$. That means that scan in TRIM0 and TRIM15 is used to set the range of voltage for the trims. The $TrimValue_{0_{x,y}}$ is used as the lower range for all of the 16 trims in each individual pixel and the $TrimValue_{0_{x,y}}$ is used as the maximal range of voltage for the trims, and the step of the strim is $TrimStep_{x,y} = TrimValue_{15_{x,y}} - TrimValue_{0_{x,y}}/15$.

That process allows us to calculate calibrate trim values for each of the pixel. Now in order to determine which of the trim value should be set for pixel, we use the $N_{HITS}(GT)$ in each of the two scans ($Trim_0$ and $Trim_{15}$). Each of those scans produces a gaussian distribution. We calculate mean of the distribution in TRIM0 M_{TRIM0}, and TRIM15 M_{TRIM15}. The goal value of the pixels trims is calculated as $GT_{target} = M_{TRIM0} + M_{TRIM15}/2$.

Then each of the pixels has its trim value $PixelTrim_{x,y_a}$ set as close to GT_{target}.

Then in order to exclude the noise from the signal coming from the pixels, the effective global threshold $GT_{effective}$ is set to $GT_{target} + 1000electrons$. So the actual threshold for the individual pixel $Th_{x,y} = PixelTrim_{x,y_a} + GT_{target} + 1000electrons$)

2.1.1.2.2 Calibration example Let's say that we start the scan with global threshold value of $GT = 1100DAC$, and increase to $1600DAC$, with a step of $1DAC$.

For example, if the mean from TRIM0 was 1111, and mean from TRIM15 was 1431, this means that the trim step is (1411-1111/15) == 20. This means that the TRIM1 is 1131, TRIM2 is 1151 and etc.

2.1.2 Monitoring and calibration problems

2.2 RICH

2.3 Trackers

2.4 Calorimeters

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo.

Quoteauthor Lastname

3 Machine Learning methods

4

Machine Learning methods and analysis for VELO

This chapter is dedicated to the research of the VELO detector, and possibility of application of machine learning methods for it's purposes. Most of the scope of the work relates to the version of VELO used in runs 1 and 2 of the LHC, but it also contains studies for the upgraded VELO. Those studies have been conducted in different timespans. So for the section 4.1 the data used (2010-2017) doesn't contain the data that was used (2010-2018) for the analysis in sections 4.3 and 4.4. The structure of sections in this chapter is keeping the chronological order of the conducted studies.

4.1 Run 1 and 2 calibration analysis

4.1.1 The Data

The data used in this section comes from a timespan 2010-08-18-017-06-21, and contains 30 calibration. It relates to the calibration data. The most significant for the VELO set of data are the following parameters: Pedestals P, Low Threshold L_t and High Threshold H_t. As discussed Low Threshold and High Threshold only differ in scaling factor, so in further analysis in this section only high threshold and pedestal parameters will be used. The most useful tool for the analysis of this dataset is the 2D histogram, with a channel number on X axis, and a parameter value on the Y axis. The color of the histogram marks the intensity, or a number of occurance of the (X,Y) pair in the dataset. When there is no occurance of a gicen pair of values, the color is set to white. The color scale is different for each of the plots, unless mentioned otherwise. The color scale in most of the plots is ommited, and it follows the intuitive rule of warmer colors meaning more occurances. This kind of figure has prooved to be very useful, since it can aggregate multiple dimentions of the data.

Before going further a discussion of the dimentionality of the data is necessary. The VELO sensor has

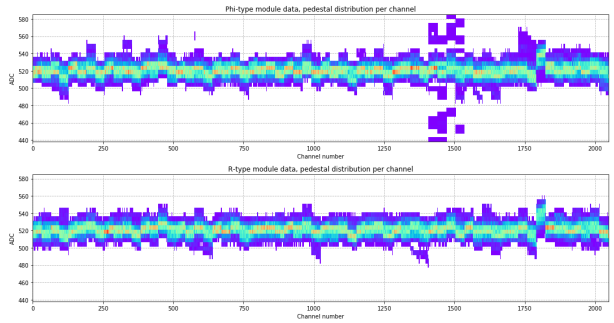


Figure 4.1.1: Histogram of the all of the pedestal values, across all of the calibrations The histogram above is $P_{Ch,R,\#,T}$ and the below $P_{Ch,\varphi,\#,T}$).

2048 channels, which constitues the channel dimmention. The sensors are or R and φ type, for each of the 42 modules. Those are make two additional dimentions of the data. The final one is the calibration date, which can be different in further sections.

Dimention name	Symbol	Size
Channel	Ch	2048
Sensor type	R, φ	2 (R or φ)
Module number	$\#$	42
Calibration date	T	30

Table 4.1.1: Table of dimentionality of the calibration dataset.

We will be using the following notation to denote the slice of dataset: $ParameterType_{Ch,S,\#,T}$. Where *ParameterType* stand for the type of the parameter like P or H_t and the other symbols are explained in the Table 4.1.1. Single symbol without values marks the use of full range of data on that dimension. Examplary notation: $P_{Ch100-Ch500,R,\#11,T}$ is a slice of R-type sensor channels from 100 to 500 in module $\#11$ in all of the calibrations.

4.1.2 Pedestals

The Fig. 4.1.1 depicts all of the pedestals value in all of the dataset. The value oscillates around 520 ADC, with visible artifacts at channels 1400-1520 in R type sensor, and near channel 1750 in both sensor types. The first artifact (rangees 1400-1520) comes solemly from the sensor #85 , visible in the Fig. 4.1.2. This is believed to be a malfunction of the sensor in range of channels in this particular sensor. The other artifact near channel 1750, is believed to be coming from a design flaw, which placed a clock line too close to those channels. This is not a major problem, since the purpose of pedestals is to level the differences in a basic level of the signal.

The radiation constantly damages the sensors, resulting in need of asjusting bias voltage The pedestals are actually a way of fine-tuning the levels of the signal. If the value of the pedestals would exhibit a trend this could mean that the bias voltage adjustment is not sufficient, or it doesn't give desired result. Thus, a study of the trend of the pedestals is important aspect of this analysis. The trend was calculated by fitting a line ($y = ax + b$) to the pedestal values in the time (calibration) domain. The value of the linear

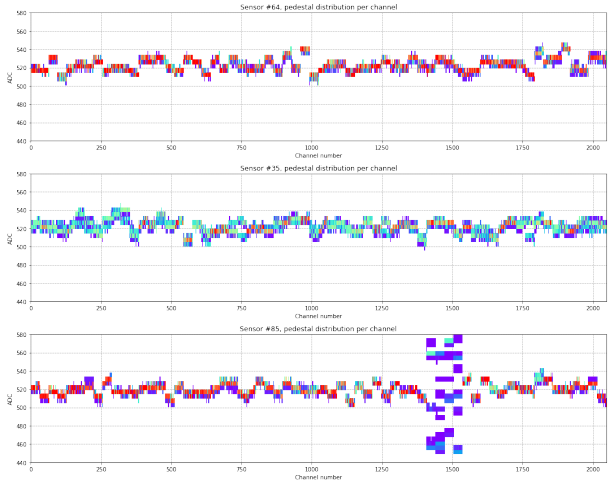


Figure 4.1.2: Pedestal plot for sensors $P_{Ch,R,\#64,T}$ $P_{Ch,R,\#35,T}$ $P_{Ch,R,\#85,T}$. #64 and #35 represent a typical pedestal values, while in #85 there is a malfunction visible close to channel 1500.

coefficient (a) which is responsible for the trend, is calculated individually for each of the channels of the detector. The distribution of this coefficient is visible in the Fig. 4.1.3. The mean value of the coefficient is $9.63e-05$ which is so close to zero that it can be assumed that there is no overall trend.

4.1.3 Threshold

Of the two studied parameters, the high threshold parameter is the one actually responsible for the sensitivity of the detection of the hit. It is one of th many steps of filtering the signal coming from VELO. Further, the hit information is crutial for the reconstruction algorithms and particle detection and recognition. At first glance, a 2D histograms visible in Fig. 4.1.6 is cluttered by outlying values. The most visible ones are near value 127 across all channels. This is actually an expected sight, as setting a value to 127 means the maximal possible value which makes the data coming from the channel unusable. This is a masking value, and occurances of those masks are discussed in detail in the next section. This comes from afaulty sensor #67 and dates '2016-11-07' and '2016-11-11', where the threshold value reaches 400 ADC. This likely an error of that could accidentally get into the dataset, and yet there is no explanation for those values. Moving on, we ignore those values, and limit the analysis to the range of values close to 0.

The Fig. 4.1.4 depicts $H_t(Ch, R, \#, T)$ and $H_t(Ch, \varphi, \#, T)$ in range of values from 0 to 30. One of the immidiatelly visible things are the vertical strips of low intensity, spaced throughout the channels. This a header cross-talk effect, in which the signal is disturbed. In some cases it might be useful to exclude the channels that contain the header crosstalk effect, and the full list of the those channels is present in the appendix. The $Ch*$ denotes all of the channels withouth the header cross-talk effect. The reader can examine the difference of the histogram with and without header cross-talk in the Fig. 4.1.4 and 4.1.5.

Another insight coming from the 2D histogram coming from the Fig. 4.1.5 is that there are regularities of low intensity, repeating roughly every 512 channels. The periodicity of those regularities comes from the design of the sensors, and the pattern of the sitribution of that the channels and their lines. (@TODO what lines) More precisely they are recognised as imperfect calibrations coming from two

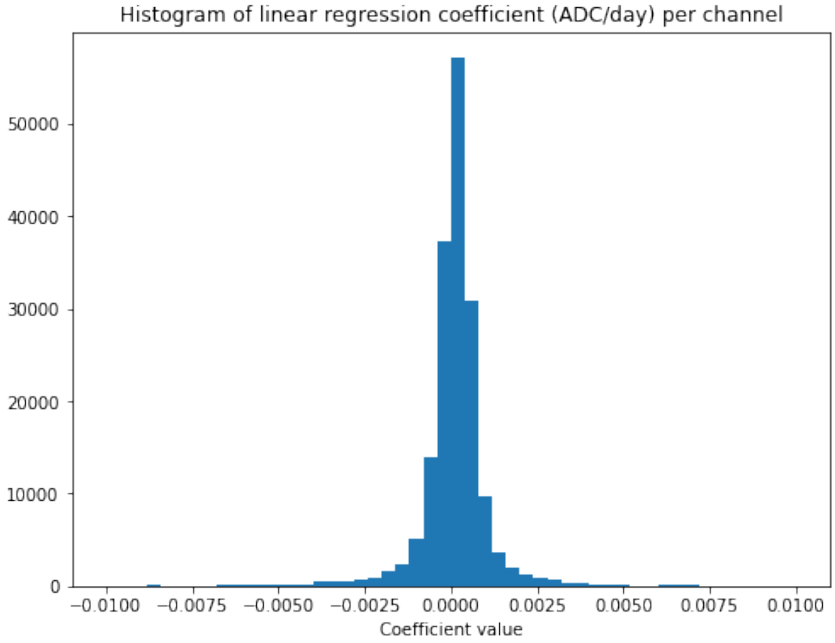


Figure 4.1.3: Histogram of linear coefficient of the pedestal trend per channel. Some of the outlyingcoefficient values have been trimmed, and are outsidethe scope of this plot.

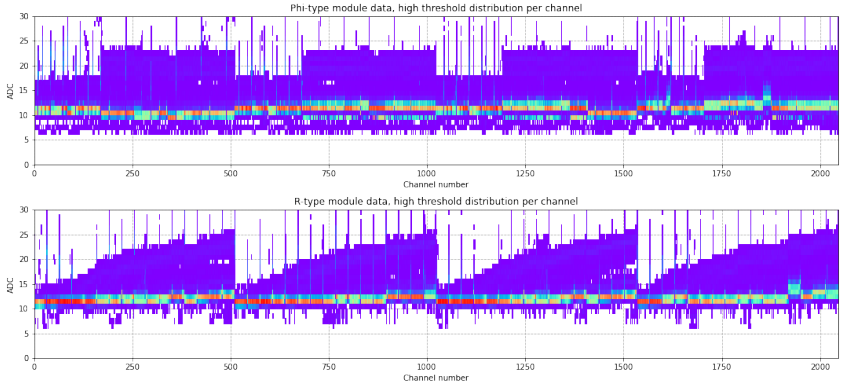


Figure 4.1.4: High threshold distribution with header cross-talk. $H_t(Ch, \varphi, \#, T)$ above and $H_t(Ch, R, \#, T)$ below.

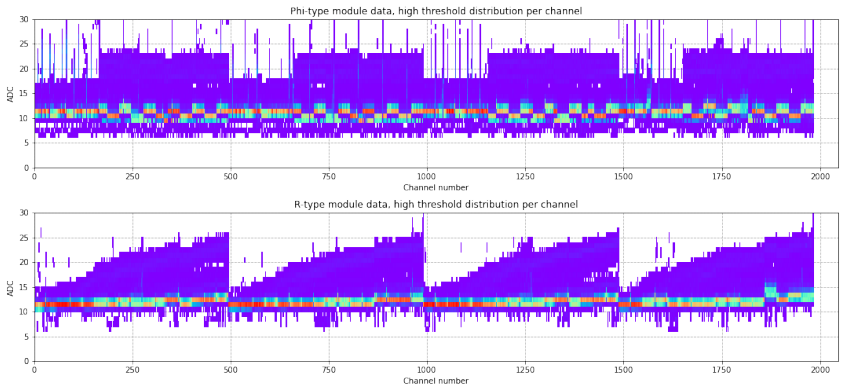


Figure 4.1.5: High threshold distribution without header cross-talk. $H_t(Ch*, \varphi, \#, T)$ above and $H_t(Ch*, R, \#, T)$ below.

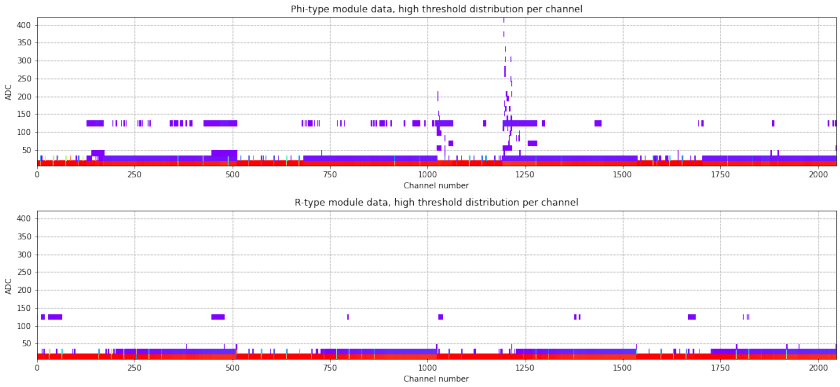


Figure 4.1.6: High threshold distribution of all of the calibrations, across all sensors. $H_t(Ch, \varphi, \#, T)$ is above, and $H_t(Ch, \varphi, \#, T)$ is the plot below.

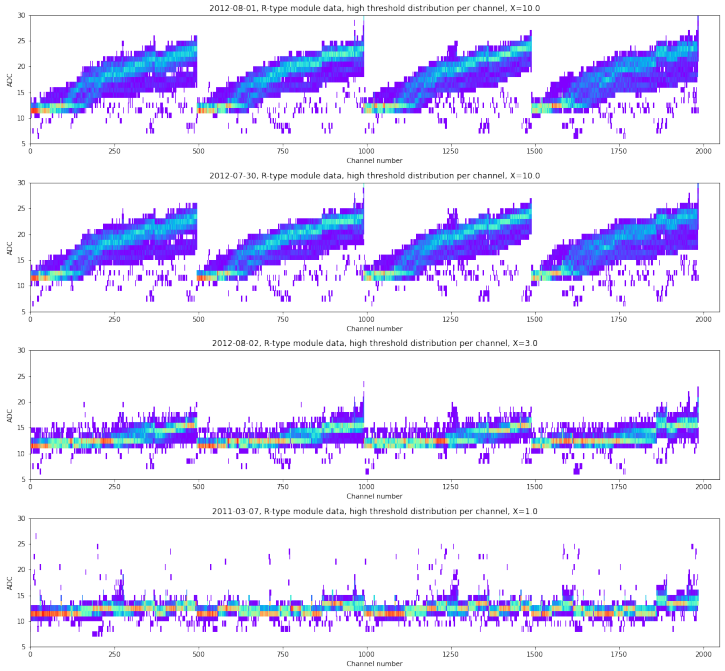


Figure 4.1.7: Various R type calibration dates, with assigned value of outlierness.

dates '2012-07-30', '2012-08-01'. Those two dates are imperfect due to the power break occuring on those dates. This imperfect calibration was influencing the sensitivity of the detector, and overall data taking. Additionally for the purpose of outlierness analysis two calibration dates also being are recognised as imperfect 2012-08-02 and 2011-03-07. Those occurances of the miscalibration are a direct motivation for detecting anomalies in the calibration in section 4.2.

4.1.4 Masked Values and others

As mentioned previously, the $H_t = 127ADC$ means that the channel is being masked. Setting the threshold so high, means that no hit will be registered in a given channel. The masks were given to the channels that exhibited exceptional noise levels. Their occurances are depicted in Figs 4.1.10 and 4.1.11. In 4.1.10 the channels from all calibration are plotted as number of blocked channels per sensor in time domain. The important insight is that some sensors are more prone to masking channels than others,

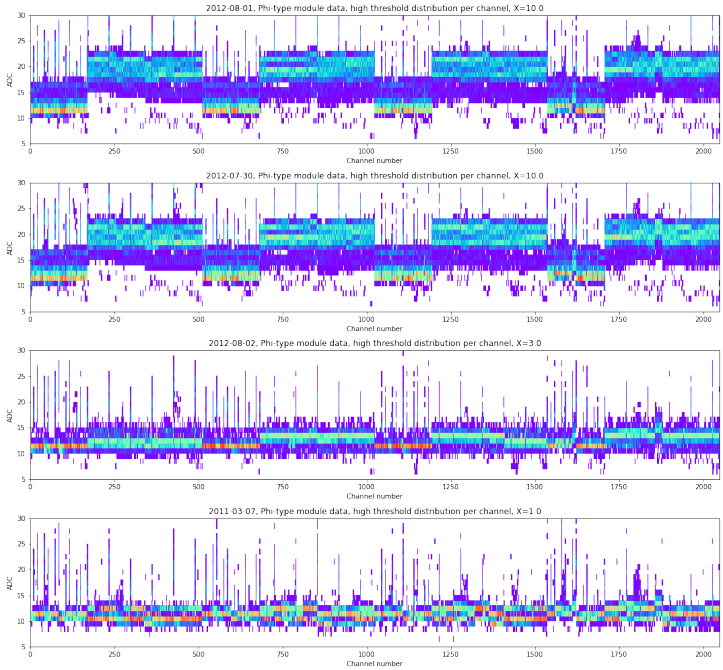
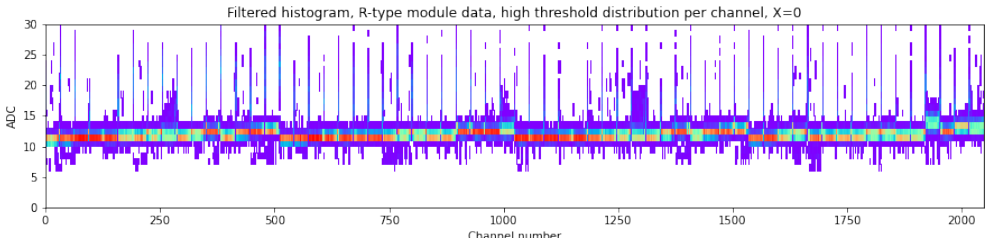
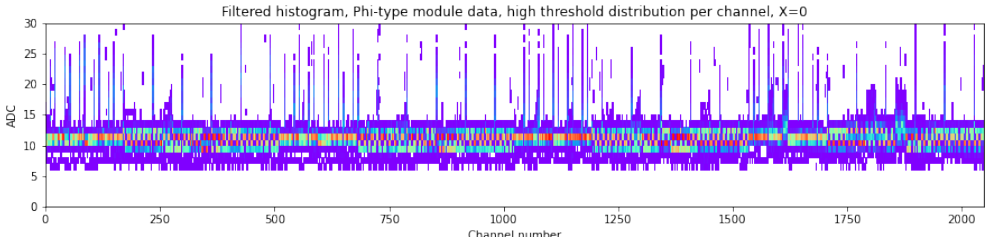


Figure 4.1.8: Various phi type calibration dates, with assigned value of outlierness.



(a) .



(b) .

Figure 4.1.9: All of the calibrations, with reduced dimentionality using autoencoder and PCA.

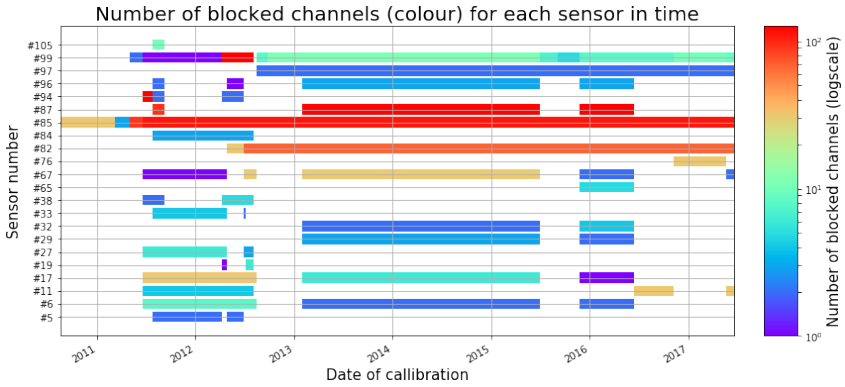


Figure 4.1.10: Distribution of blocked channels per sensor in time.

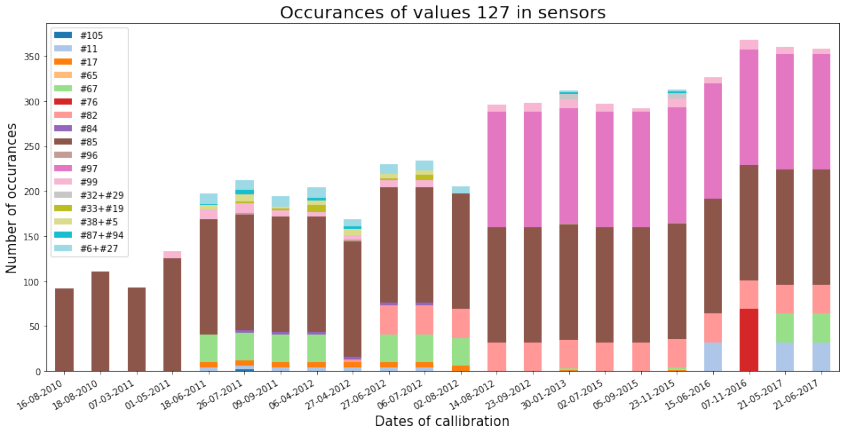


Figure 4.1.11: Total number of occurances of masked channels versus time.

and that during the ammount of masked channels changes. In some cases the masked channels can go away and come back into the sensor. The figure 4.1.11 depicts number of masked channels in time domain, with a sensor split. It is clearly visible that the total number of masked channels grows with time. (@TODO show that this goes in hand with total luminosity). This is expected as with time the negative radiation effects accumulate.

Apart from the masked values, the usual and outlying calibration, there is still a part of the data which doesn't fit any of those categories. This part of data can be characterised as the $H_t > 50ADC \wedge H_t \neq 127ADC$. In the Fig 4.1.12 depicts those other outliers. Number of occurances of these outliers change from calibration to calibration, but overall is limited to three sensors: #85 #67 and #94.

4.2 Outlierness with probabilistic programming

4.2.1 Dataset creation

The analysis of the calibration data in the previous section yielded a usefule insight into the properties of the high threshold H_t. A standard calibration for R of φ sensor is usually close to certain value, independant of the calibration. It is assumed that the distribution of the threshold value is given by the gaussian distribution. The analysis of the bad (@TODO add asterisk here) calibrations shows that actually that the distribution H_t is alaso dependant on the channel (e.g the regularity repeating every 500 channels).

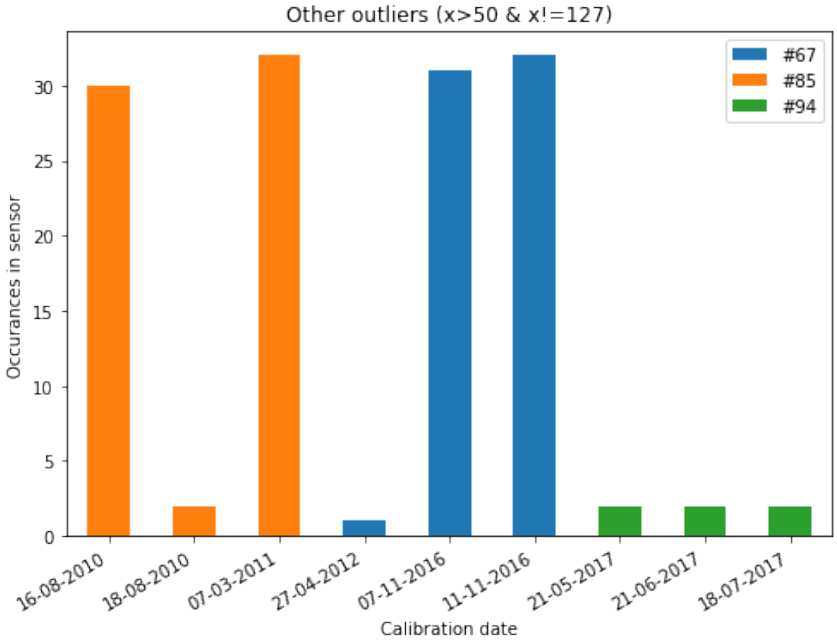


Figure 4.1.12: Outlying values other than masked channels.

The "bad" calibrations were assigned an "outlierness" value X. This "outlierness" measure is purely artificial and subjectine, and also expresses a strenght of belief that a given calibration is an outlier. The bigger the number, the stronger the belief that calibration is outlier. There is no scale or limit on the outlierness number, other than that the 0 represents a normal calibration. The actual assigned values can be seen in the Table 4.2.1.

Table 4.2.1: Outlierness calibration dataset values.

Calibration date	X
2011-03-07	1
2012-08-02	3
2012-07-30	10
2012-08-01	10
all others	0

4.2.2 Model

For the purpose of simplicity this subsection, we will be using only the R-typed sensor data, and decribe a notation (@TODO linebreaks in the equation):

$$T_n = H_n(n, R, \#, T*) \wedge T\prime_n = H_n(n, R, \#, T) \tag{4.1}$$

Where n stands for a channel number across all sensors in time. As mentioned previously, it is assumend that the distribution of the threshold in particular channel is gaussian.

$$T_n \sim Gaussian(\mu = \mu_n, \sigma = \sigma_n) \tag{4.2}$$

As visible in 4.1.9, the value of the normal calibration for the R sensor oscillates around $12ADC$. Therefore the parameters μ_n and σ_n can be calculated by fitting the channels histogram to gaussian distribution. In this case the probabilistic programming paradigm is used to achieve that. We use the dataset containing only "good" calibrations to calculate that.

In order to extend the model to the "bad" calibrations, we add additional terms to the model:

$$T\prime_n \sim Gaussian(\mu = X * \mu\prime_n + \mu_n, \sigma = X * \sigma\prime_n + \sigma_n) \tag{4.3}$$

The X stands for the outlierness of the calibration, and $\mu\prime$ and $\sigma\prime$ are coefficients of the linear dependency on the X. These parameters, as previously are intrinsic property of n channel. The X value is the same across given calibration date. Notice that when the $X = 0$, then, the model in Eq. 4.3 is equivelent to 4.2. This allows to use the parameters μ_n and σ_n calculated for previous model, to be used in the extended one. Given the X from the 4.2.1, we are able to calculated $\mu\prime_n$ and $\sigma\prime_n$.

4.2.3 Training

(@TODO) add details about pymc training here

4.2.4 Results

The model presented in this subsection is a machine learning model, but is quite different from the models used frequently (such those based on neural networks).

It is not a black-box model, as for each of the channels there are only 4 parameters that are needed. Such low complexity of the model also allows for no train-test split (a usual practice in mahcine learning). This approach has additional benefits such as:

- Partial dimentional independence - This model can be used to extract the value of the outlierness not only for entire calibration, but also for a specific sensor, or module, or a group of channels, or even just specific channels. When analysing the calibration this allows for a more detailed insight as it can bring focus to a specific part of a detector that exhibits more unpredicted behaviour.

- Generation of artificial data - Because the probabilistic programing is based upon the statistical inference, and distributions, simply by calculating the μ and σ parameters of gaussian distribution we are able to generate an artificial sample of the data.

- Interpolation and Extrapolation - This model was trained only on four different values of X, and yet, it is capable to asses the outlierness value not only in the range between those four values (e.g. $X = 5$), but also through generation of artificial data, in can show what a calibration that this outside the scope of the data (e.g. $X = 12$) might look like.

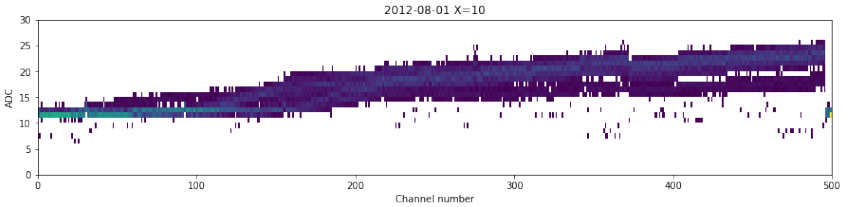


Figure 4.2.1: 500 channels from a single calibration date marked with X=10

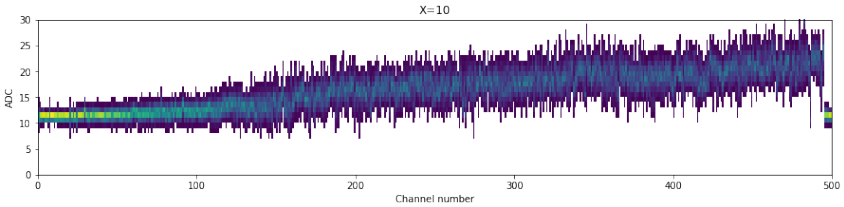


Figure 4.2.2: 500 channels from a single generated calibration at X=10

In Fig. 4.2.1 there is a sample of the data (500 chanels) that was taken of the "bad calibrations". It can be noted that the range of the H_t values is on par with thos visible in the plot of the generated dataset 4.2.2. (@TODO add another figure with artificial data of 12, and reference it here) Additionaly Fig there is a plot of calibration with value $X = 12$, which was not present in the dataset and is actually outside the scope of the dataset.

This model was actually introduced to the Lovel monitoring system in the second half of year 2018, and was used to monitor the oncoming calibrations. The Figure 4.2.3 depicts a screenshot from the monitoring system, and shows outlierness levels for the sensor # 21. It is noticable that the value of the outlierness was slightly eleveted in 2017, which is attributed to some changes in the cooling system.

4.3 Dimentionality reduction

The Velo detector in it's strip version, has 172 032 strips. Many of the parameters needed for it's calibration are calculated per strip. This is a huge amount of parameters, which is not easilly understable for human operators. In it's pixel version, the number of such parameters will grow to 41 million (exactly

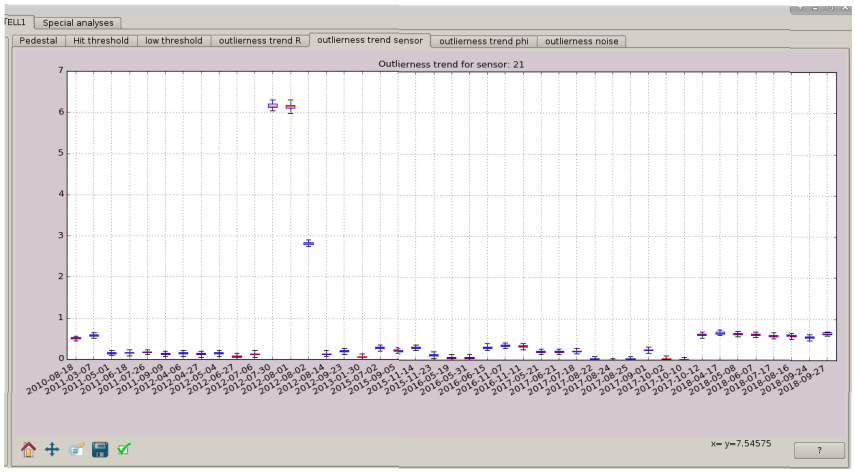


Figure 4.2.3: Screenshot from the outlierness monitoring in Lovell software.

$256 * 256 * 12 * 52 = 40894464$)[1]. Thus, an early tests of dimentionality reduction techniques will be very useful for the future. In this section we will only use high threshold parameter. There are mutliple ways of applying PCA or autoencoder to the dataset (using different set's of values - or dimentions).

4.3.1 Methods

4.3.2 Pedestals dimentionality reduction

For the purpose of looking at what differentiates the pedestal parameter in some sensors frome the others. For this purpose we used PCA reduction, by reducing each channels time progression (a single dimentionality reduction for each of the sensor's channel). The combined result is visible on Fig. 4.3.1. Because of the high number of individual sensors, the split into several plots gives more clarity (Fig. 4.3.2).

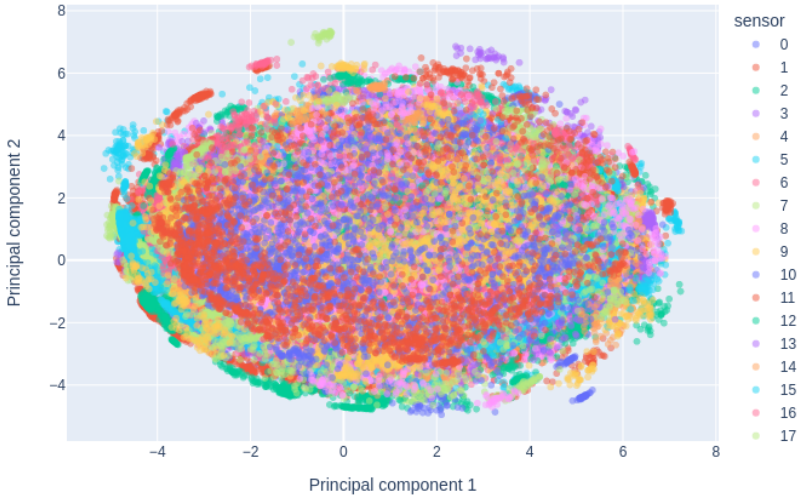
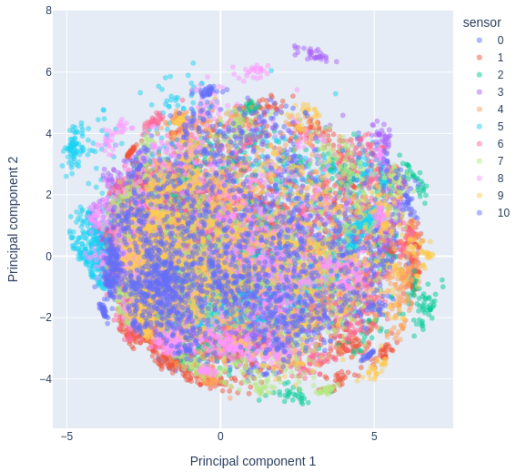
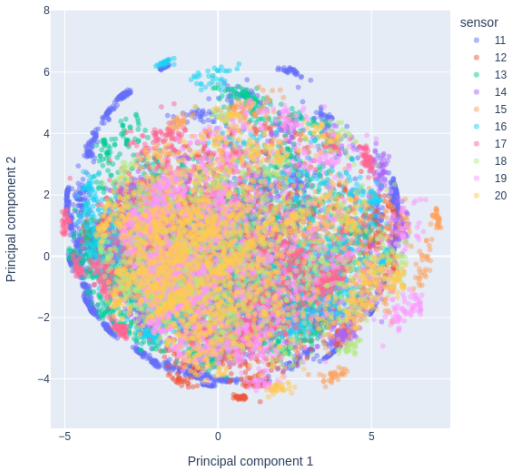


Figure 4.3.1

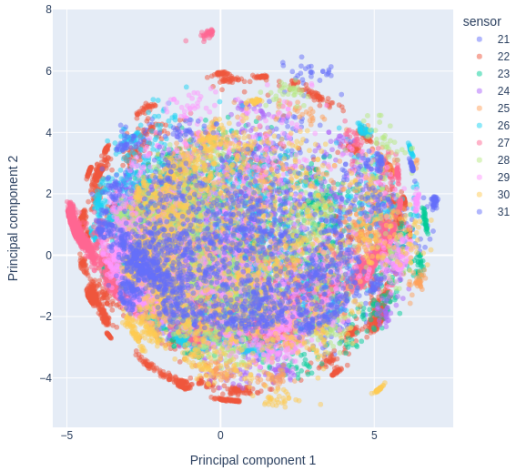
The reduced dataset shows a 2D gaussian distribution. It can be seen that some of the channels from a particular sensor can form clusters, or be distributed on a ring on the border of the dataset. What actually happens with a single datapoint with the tranformation can be explained by examining the particular channels. In the Fig. TODO there are 3 selected channels from a plot of the reduced data in sensor 11. Those points are also plotted at the Fig. TODO. We can see that the sensors on the opposing edges of the circle formed by the data, represent different trends in the pedestals value. Channel's 1899 pedestal value grows over time, but 543's decreases. Values in channel 322 stays high and stable. Therefore we can say that that essentially the PCA reduction separates the channels based on their trend. The 2D gaussian distribution structure makes it consistent with the trend calculation in previous sections, as the distribution centers around 0. This means that there is no overall trend (no growth, or decrease).



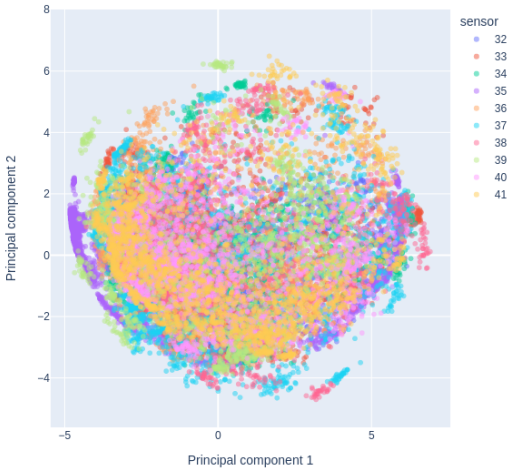
(a) PCA, R sensor type



(b) PCA, phi sensor type

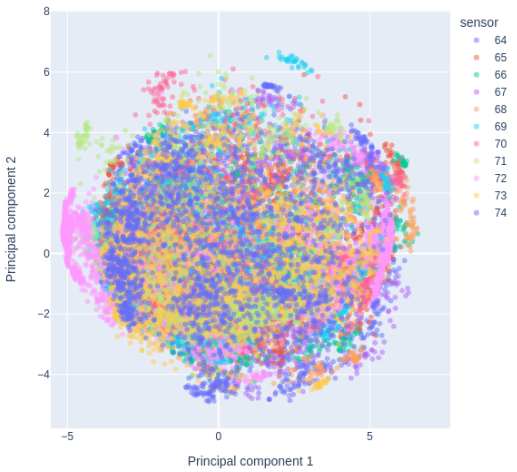


(c) autoencoder, R sensor type

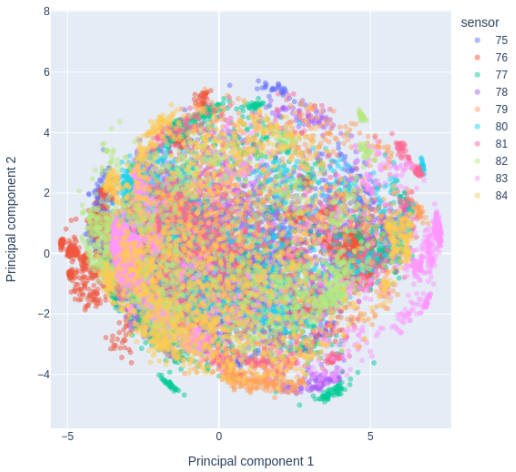


(d) autoencoder, phi sensor type

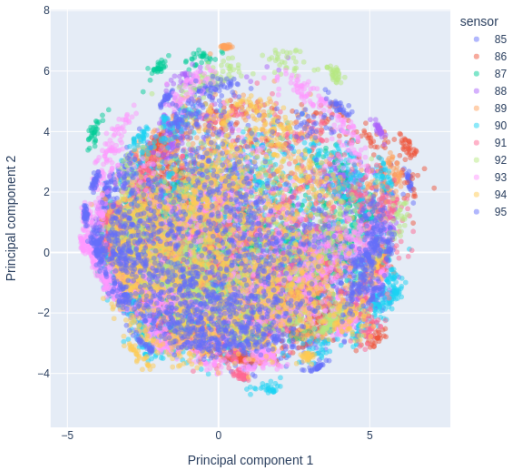
Figure 4.3.2: All of the calibrations, with reduced dimentionality using autoencoder and PCA.



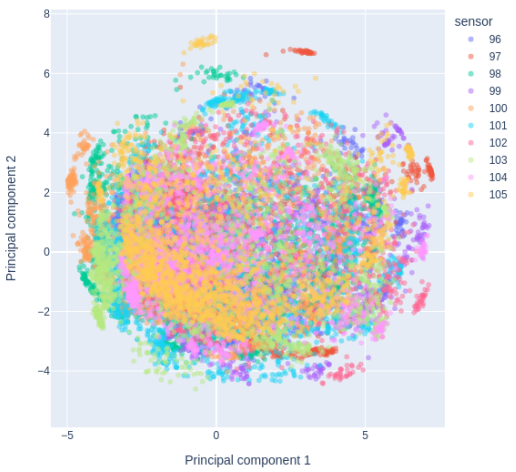
(a) PCA, R sensor type



(b) PCA, phi sensor type

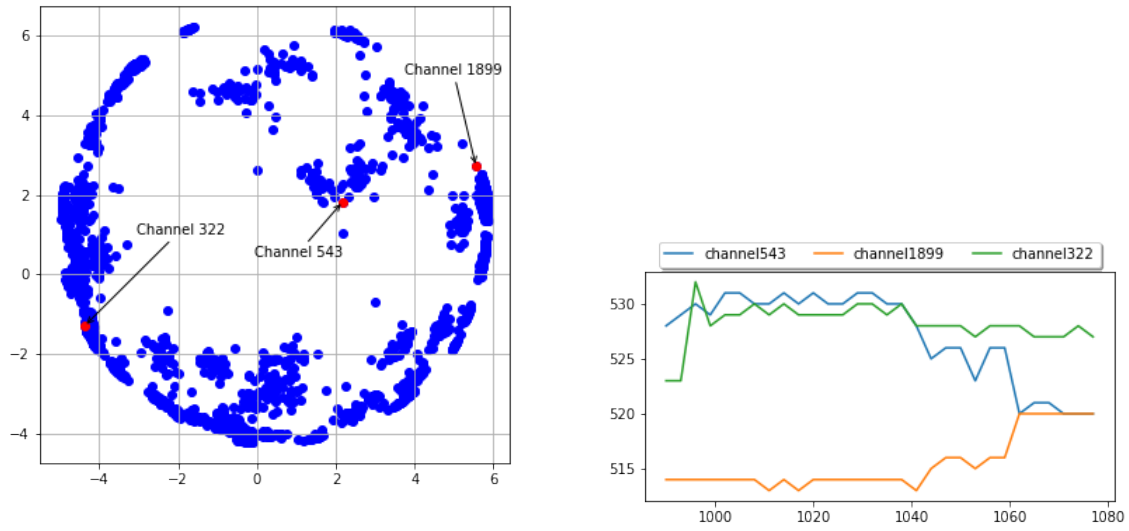


(c) autoencoder, R sensor type



(d) autoencoder, phi sensor type

Figure 4.3.3: All of the calibrations, with reduced dimentionality using autoencoder and PCA.



(a) .

(b) .

Figure 4.3.4: All of the calibrations, with reduced dimentionality using autoencoder and PCA.

4.3.3 Threshold dimentionality reduction

The most usefult is feeding the algorithm with each of the sensors, with separation for the R and phi types of sensors. This allows us to easilly see changes common for all of the sensor at once. In Figures 4.3.6-4.3.8, you can see ten consecutive calibrations plotted after dimentionality reduction of a single sensor ($n_{dim} = 2048$) to a 2D ($n_{dim} = 2$), and plotted on a plane. The color represents the number of a sensor. In both of the plots made using PCA and autoencoders you can see that two outlying dates (2012-07-30 and 2012-08-01) stand out significantly. In contrast with autoencoder, the PCA is deterministic, and does not require random initialisation. Because of this, there may be some variance in autoencoder model results, even when using exactly the same setup.

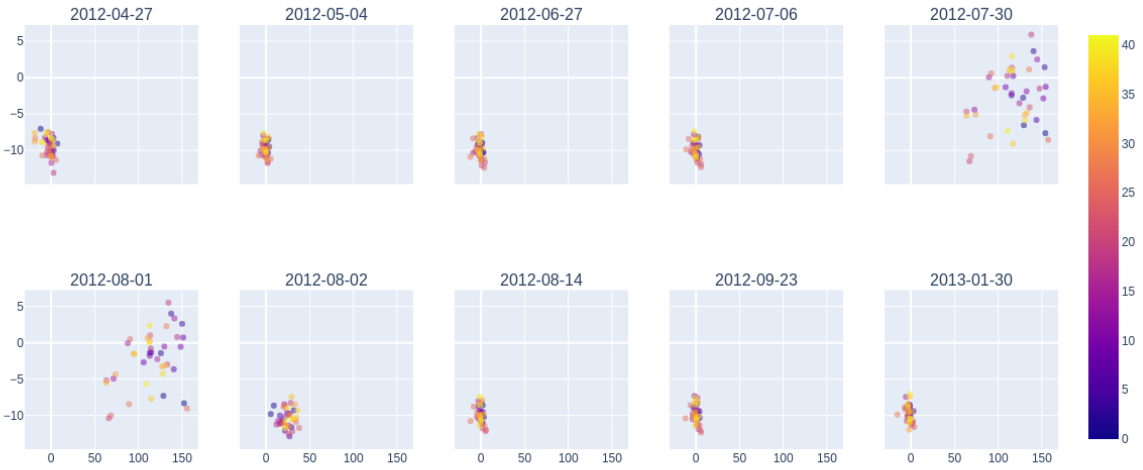


Figure 4.3.5: Time progression of selected section of calibration dates, with reduced dimentionality using PCA, only for R sensors.

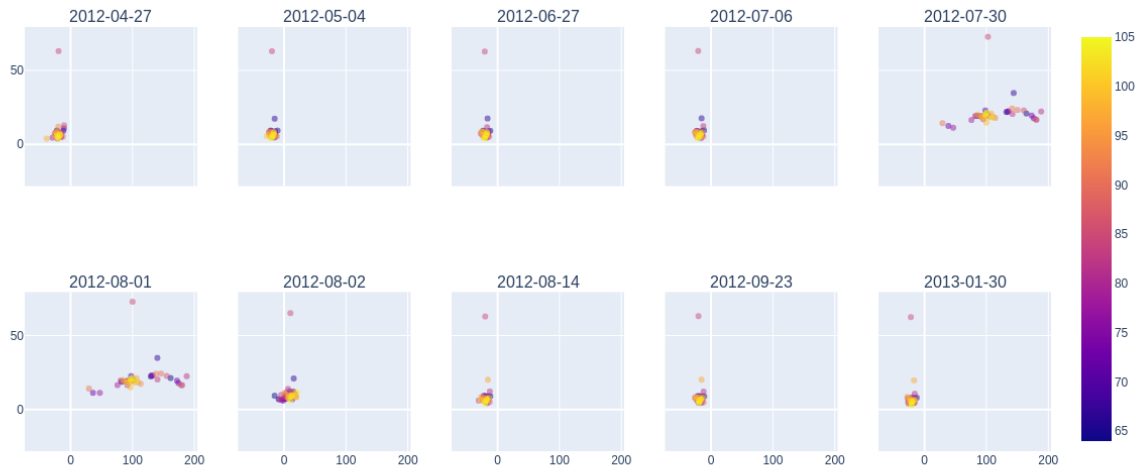


Figure 4.3.6: Time progression of selected section of calibration dates, with reduced dimentionality using PCA, only for phi sensors.

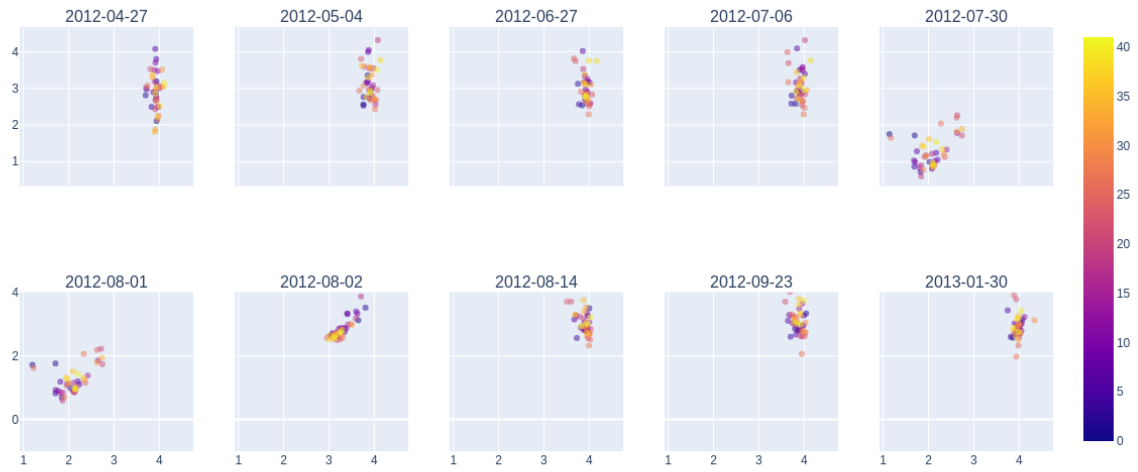


Figure 4.3.7: Time progression of selected section of calibration dates, with reduced dimentionality using autoencoder, only for R sensors.

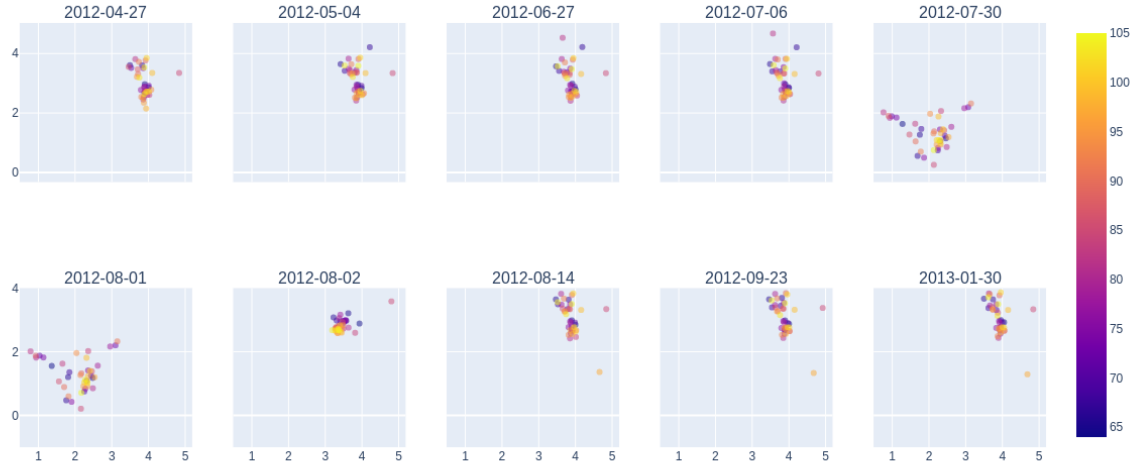
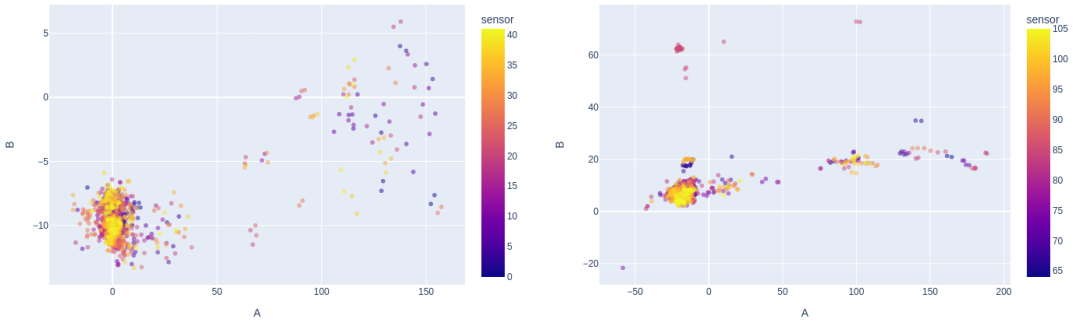


Figure 4.3.8: Time progression of selected section of calibration dates, with reduced dimentionality using autoencoder, only for phi sensors.

The Figures 4.3.9a-4.3.9d represent all calibration dates on the same plot, splitted into different sensors, and created with different techniques. Notice that all of the plots have different scales of the axes, as the methods used to reduce dimentionality do not retain the scale. The most useful insight to those plots is that the relative change represents outlying calibrations.

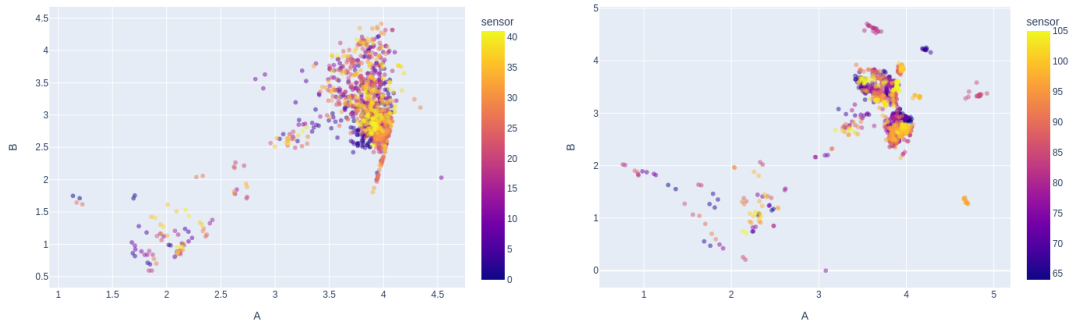
4.4 Time to calibration forecasting

The calibration of the detector is essential for it's proper use. It set's all of the parameters of the detector that will be needed during the data taking. In Velo runs 1 and 2 of the LHC it was possible that if the



(a) PCA, R sensor type

(b) PCA, phi sensor type



(c) autoencoder, R sensor type

(d) autoencoder, phi sensor type

Figure 4.3.9: All of the calibrations, with reduced dimentionality using autoencoder and PCA.

channel thresholds were set too high, useful data could be lost. When a electrical signal coming from a single strip would not exceed the threshold, it would be discarded. Therefore it is crutial for the detector operations to have as frequent calibrations as possible. But the calibration process requires a noise data recorded when there is no beam present. Ideally, the longer the time that it takes to record the noise, the better the callibration accuracy (gaussian noise mean calculation. This also means that calibration requires a special time slot in between data taking, when no beam is present. This motivates the studies of the forecasting of the need for calibration in Velo.

4.5 Dataset

In the data availible to the analysis we can distinguish two kinds of data: raw data and calibration data. The raw data is the data that is recorded during the beam conditions, and the calibration data is the data taken during no-beam. The actual way of calculating the values present in the data does not differ. The ammount of datapoints is significantly greater for the raw data, as it was taken in any single data run, which can last from a few seconds, to about an hour. Calibration data on the other hand was taken every few weeks. In both kinds of the data there is a constant dimentionality of the parameters, as each of the values was taken for each of the 2048 channels of the sensor individually, in total giving a 172 032 values of a single parameter for the whole detector. The analysis of the trends of the trends has produced a candidate for a parameter that could be used to identify a need for a calibration. The *pedestal subtracted*

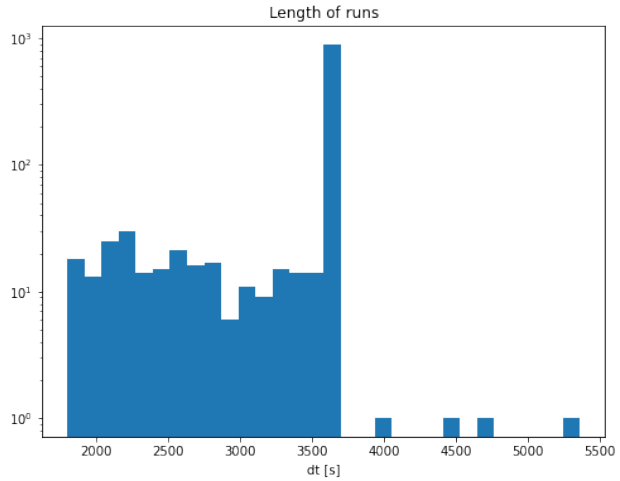


Figure 4.5.1: Histogram of lengths of runs in the dataset, expressed in seconds. Significant ammount of runs lasted about 3600 seconds (1 hour).

$\Delta\mu$ parameter is the value of the mean of the noise calculated as follows:

$$\Delta\mu = \mu_{current} - \mu_{calibration} \tag{4.4}$$

Where $\mu_{current}$ is mean of the signal coming from the current run. The analysis of this parameter has shown that a value of the standard distribution of this paremeter $\sigma(\Delta\mu)$ within the detector grows after the calibration, and drops after one. An example of this is present at the **??**.

The data used for the forecasting model comes from 2018. The calibration process and raw data in 2018 is a best candidate because of the frequently occuring runs, and calibrations. The exact timespan of the data is from 2018-05-08 to 2018-09-24. This constitues exactly 4 calibration periods. The used data comes in a form of $\sigma(\Delta\mu)_t$ where t stands for a given run time. The time information is translated to delta time $(dt_n = t_n - t_{n-1})$. The continious time series of data is processed to a windowed time-series, of 100 data points, beginning with the first data point after calibration, and padded with zeroes.

$$X_{it} = \begin{bmatrix} std_{it} \\ dt_t \end{bmatrix} \tag{4.5}$$

As it is a case of supervised learning, the additional component Y_{it} is set to be a number of days left until next calibration. We have chosen the number of days as the most suitable time range for this purpose, as there can be many runs per day,

4.6 WTTE-RNN for Velo

The WTTE-RNN as discussed in Sec ... is a one layer capable of outputing the parameters of the weibull distribution. The actual type of a problem in machine learning is called survival analysis, architecture of the neural network used for forecasting is present at the Table 4.6.1. It contains 6 layers in total. The LSTM layer is necessary for the time series that is used as input data.

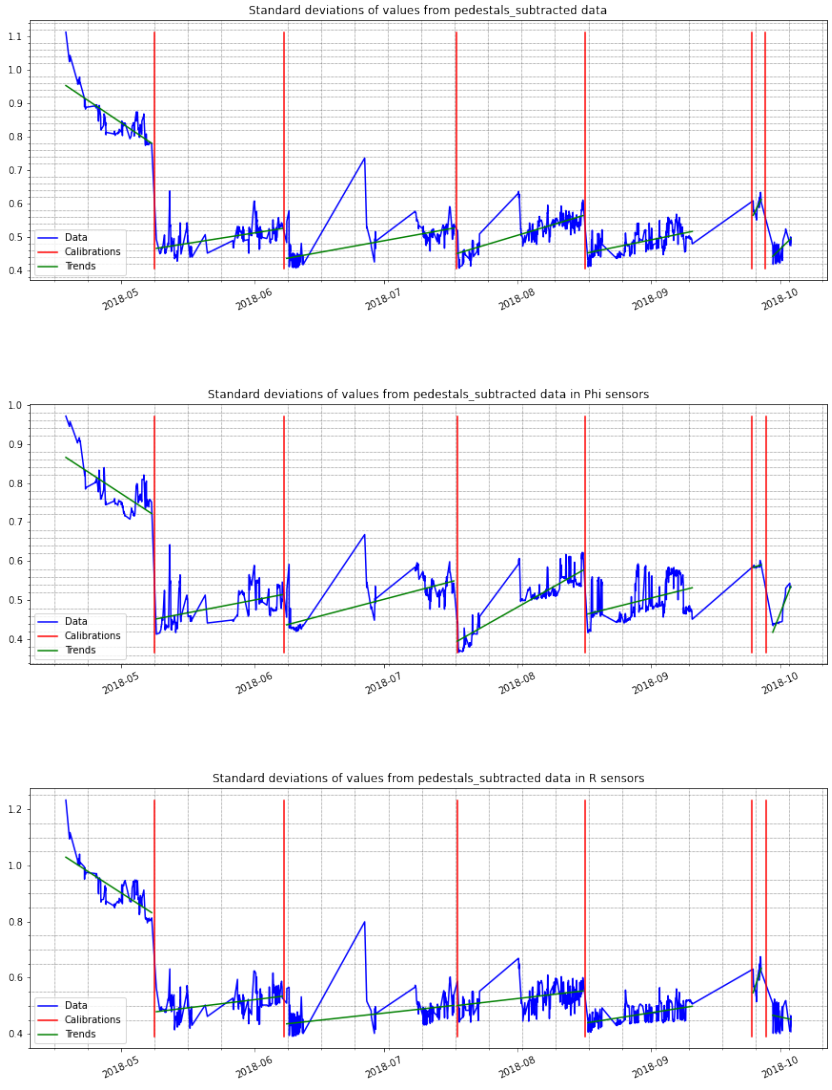


Figure 4.5.2: Standard deviation of the pedestal subtracted parameters in the whole sensor (top), pedestal subtracted in Phi sensors (middle), and in R sensors (bottom). Calibration dates are in red, and the green lines present trends calculated between calibrations.

Layer No.	Layer type	Layer size
1	Dense	43
2	Dropout	
3	LSTM	20
4	Activation - Tanh	
5	Dense	2
6	WTTE-Activation	

Table 4.6.1: The neural network layers used for the WTTE-RNN model

4.7 Mask clustering

The VeloPix masks flags are veru useful things. They help to spot if there is something wrong with the detector.

The cluster masks can pose a threat to the reconstruction algorithm, and the radial reconstruction resolution. As such they have to be monitored.

4.7.1 Mask simulation

Due to the VeloPix being still under developement and comissioning during these studies, the experience gained with the tests of VeloPix was used to create a simulation of occuring masks.

In order to make a realistic simulation we have used a process consisting of three types of intrusions:

- Random uniform changes P, in which a random portion of pixels changes their state to masked
- Linear cluster L where a linear portion of pixels are masked.
- Blob cluster G with masks created by a 2D isotropic gaussian distribution.

These intrusions are added to an initially empty matrix with a different probabilities. Each of the intrusions is created on a empty matrix with the same size as the sensor, and is simply added to the simulated matrix M. One crutial step of this simulation is random cleaning of the matrix, in which random 80% of the pixels are unset to a non-mask state. The pseudocode can be found in listing 1. This process creates a continious simulation of a VeloPix matrix masks, as the new masks and clusters are generated,

and the old ones slowly dissappear.

Data: $n \geq 0$

Result: $y = x^n$

$N \leftarrow n$;

while $N \neq 0$ **do**

  if *random()* $<$ *blob_prob* **then**

    $M \leftarrow M + G$

  else if *random()* $<$ *line_prob* **then**

    $M \leftarrow M + L$

  else

    $M \leftarrow M + P$

  end

  $M \leftarrow M + Pu$

  $N \leftarrow N - 1$

end

Algorithm 1: An algorithm with caption

4.7.2 Clustering

There are many clustering algorithms present in the field of machine learning. Although they all come under the term "clustering" there are actually multiple different goals that can be achieved by different algorithms. The most common density search clustering algorithms are DBSCAN and OPTICS. The details of these algorithms are described in section 3.6. In the case of application of the clustering algorithm towards masks in calibration, the desired algorithm should be able to

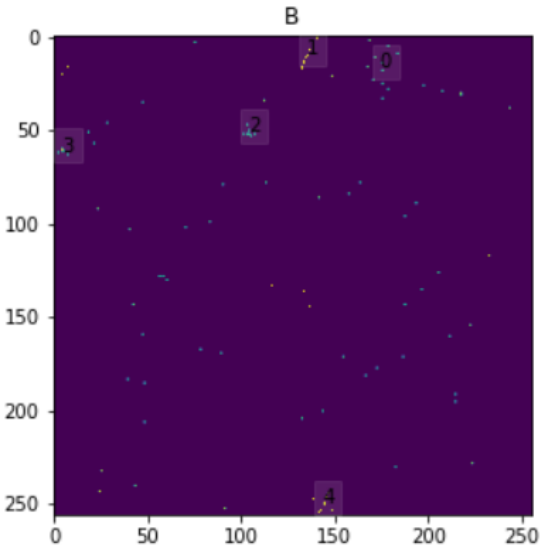


Figure 4.7.1: And examplary clusterisation using DBSCAN ($\varepsilon = 10$, MinPts $= 4$) on the binary pixel-map. Clusters are numbered 1-5.

4.7.3 Cluster features

After finding the individual clusters on one calibration (a one set of masks), the clusters of masks can be characterised by a set of features.

1. **Position** $p_k = (\bar{x_k}, \bar{y_k})$, where $\bar{x_k} = \frac{\sum_i^{N^k} x_k^i}{N_k}, \bar{y_k} = \frac{\sum_i^{N^k} y_k^i}{N_k}$.

2. **Size** $s_k = \{n_k, d_k\}$ where n_k is the number of pixels in a cluster divided by the mean number of the pixels in the given sensor's clusters.

3. **Shape** $h_k = \{a_k, c_k\}$, where a_k is the directional coefficient of the cluster measured by the fit to the line $y^k(x) = a_k * x + b_k$. The c_k is the roundness of the cluster calculated as it's Pearson Coefficient.

With those metrics, we can then define the spacial charecteristic vector as $v_k = [s_k; h_k]$. Then we define cluster as a set of unique features $cluster_k = p_k, v_k$.

4.7.4 Cluster tracking

This set of unique features is used to characterise every cluster found in the calibration. The $cluster_k$ features are then used to identify cluster k from timestep t_n as te same in the timestep t_{n-1}. It is performed by calculating a cosine similarity matrix M. The M matrix is calculated in the following manner:

$$M_{i,j} = \Phi_{i,j} * V_{i,j} \tag{4.6}$$

Where Φ is defined as:

$$\Phi_{i,j} = \frac{1}{d_{min}} * \max(d_{min} - D_{i,j}, 0) \tag{4.7}$$

and $D_{i,j}$ is a distance matrix:

$$D_{i,j} = ||p_j - p_i|| \tag{4.8}$$

The value d_{min} is set to be a limiting factor. If the distance of centroids of the clusters stays the same, the value is 1., but as the distance approaches d_{min} the value approaches 0. In the tests of the cluster tracking $d_{min} = 10$ was used. Exemplary pairing of the clusters in consecutive simulation steps is visible in the Fig. 4.7.3 with it's simmilarity matrix calculation in 4.7.2.

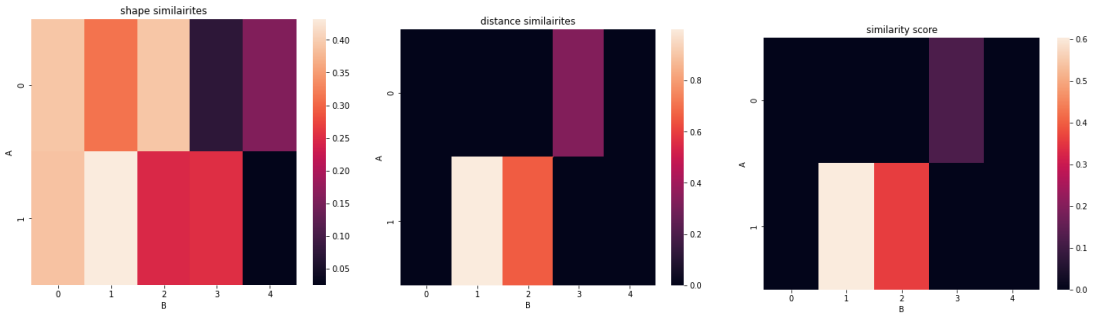


Figure 4.7.2: Rows represent clusters on image A in Figure 4.7.3, columns represent clusters on image B in Figure 4.7.3. Values indicate the Spacial Characteristics Similarity Measure V (left plot), and Positional Similarity Measure Φ (middle plot). The M matrix is the rightmost plot.

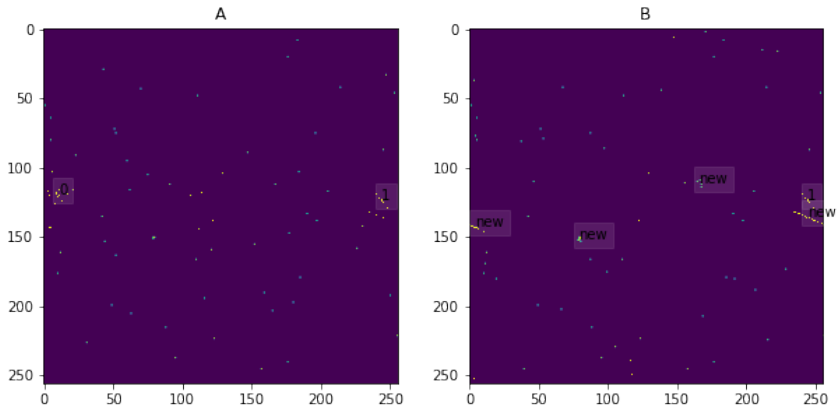


Figure 4.7.3: Clusters labelled with the same integer are chosen by the algorithm as the consecutive generations of the same cluster. Clusters labeled as 'new' are clusters on the time step t_n that were absent on time step t_{n-1}.

4.7.5 Results

The DBSCAN and OPTICS algorithms were tested on the 3000 consecutive steps of simulation of a single sensor. The ground truth in context of clustering of the masks is not something that can v determined in the real world data. But the simulation along with the simulation steps, can attribute any generated mask pixel to it's source (random uniform, linear, gaussian blob), thus providing information that can be used as ground truth. It is assumed that the generated pixel should be identified as belonging to a cluster after up to 8 timesteps. The Table 4.7.1 presents a confusion matrix calculated using that ground truth information. The OPTICS algorithm has three more times as many false positives as the DBSCAN. On the other hand it has almost twice more true positives.

Table 4.7.1: Confusion matrix values in 3000 consecutive simulation steps.

Value	OPTICS	DBSCAN
True Negative	7608	16207
False Positive	11680	3081
False Negative	1156	3167
True Positive	5665	3654
Accuracy	0.51	0.76
Precision	0.33	0.54

Both algorithms have been tested towards the ability to track the consecutive cluster's pixels through the steps of the simulation. Fig. 4.7.4 depicts the number of the masks assosiated with any cluster starting at the timestep of introduction of said masked pixel. It is clearly visible that The OPTICS algorithm recognises more pixels as belonging to clusters, and for extended ammount of time, whereas the DBSCAN very quickly drops it's attention to the pixels.

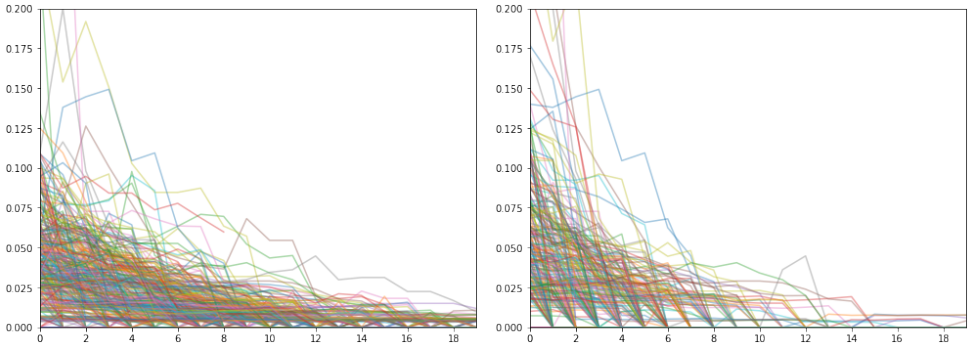


Figure 4.7.4: The fraction of pixels categorised as belonging to any clusters (Y axis) in next consecutive calibrations (X axis), since the cluster introduction to calibration (number of timesteps $n = 300$). The number of detected pixels slowly decreases with time. The OPTICS algorithm (left plot) recognises the pixels of the clusters as belonging to a cluster (not necessarily the same one) for a longer amount of time. The DBSCAN is more strict in distinguishing the pixels that belong to clusters.

The important test is the influence of the clustering algorithm towards the cluster tracking ability. The Fig. 4.7.5 shows total number of clusters in given timestep of a simulation, being recognised as new, or retaining from previous timestep. It is clear that the cluster tracking with OPTICS detects much more clusters as new, and looses more clusters from callibration to callibration, whereas the DBSCAN, if finding less new clusters, and retaining old ones.

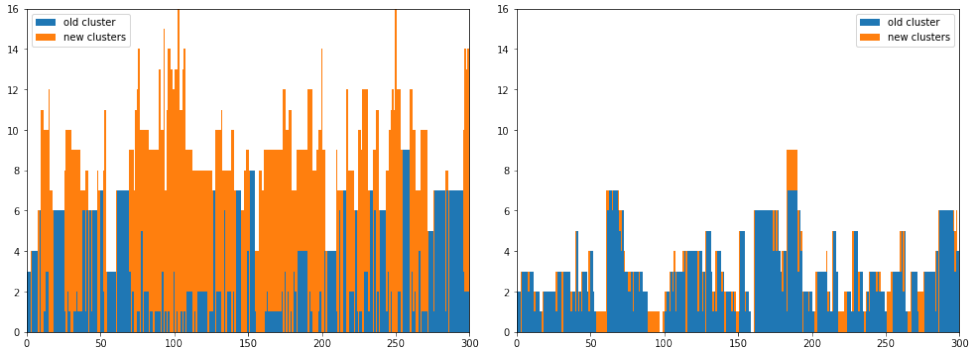


Figure 4.7.5: The number of clusters classified as being "old" - same as in previous calibration (blue colour), and a number of clusters marked as "new" - not being the same clusters as in previous calibration (orange color). The left plot belongs to OPTICS, and the one on the right to DBSCAN.

4.8 Studies of surrogate function in VeloPix

(@TODO add something here)

4.8.1 Surrogate function

(@TODO add something about how the surrogate functions works, that there is somne charge created at the sensors to calibrate) The surrogate function is the function that relates the TOT count to the charge gathered in the VeloPix pixel[7]. It has the following structur:

$$ToT(q) = p_0 + p_1 \dot{q} - \frac{c}{q - t} \tag{4.9}$$

It has four free parameters: p_0, p_1, c, t, and is essentially a convolution of linear and hyperbolic functions. There is existings underlying assumption that $q > 0$. An examplary surrogate function is visible on 4.8.1 (@TODO explain that plot better)

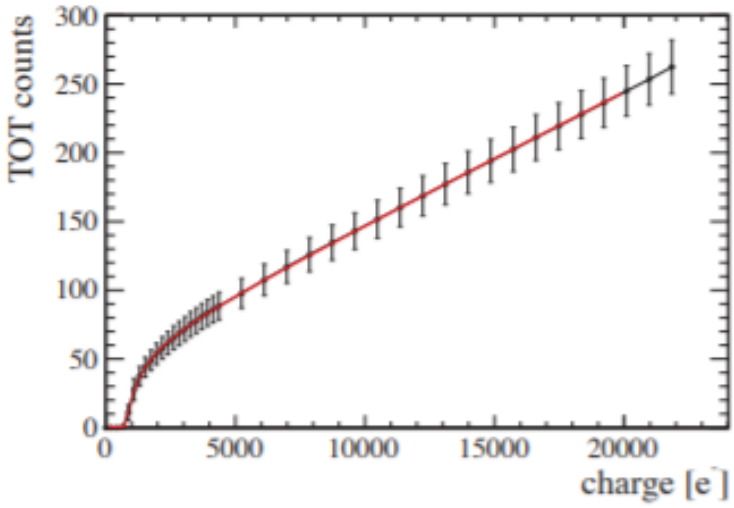


Figure 4.8.1: An examplary plot of surrogate funciton.

4.8.2 Dataset

This analysis uses the dataset gathered during the testbeam phase of the Velopix. (@TODO explaine the testbeam here) (@TODO explain the sensor markings here)

4.8.3 Cross sensor study

The test beam data contained multiple different types of sensors, with different types of irradiation. Unfortunately, in the entire datsaet, there is only one sensor that is suitable for the studies of surrogates in function of fluence. This is due to the irradiation profile of the sensors. Sensor s8 was the only one that had a clear, non-uniform profile of the irradiation, that allowed for a binning of the sensor based on the ammount of irradiation that the sensors were exposed to.

nonetheless we investigate other sensors and their distribution of the parameters in the Fig. **??** and **??**. The important insight is that for most of the sensors, the overall distribution post irradiation moved to the left for the $p0$ and $p1$. For the parameters c and t the trend is not as obvious, and we will explain the further analysis.

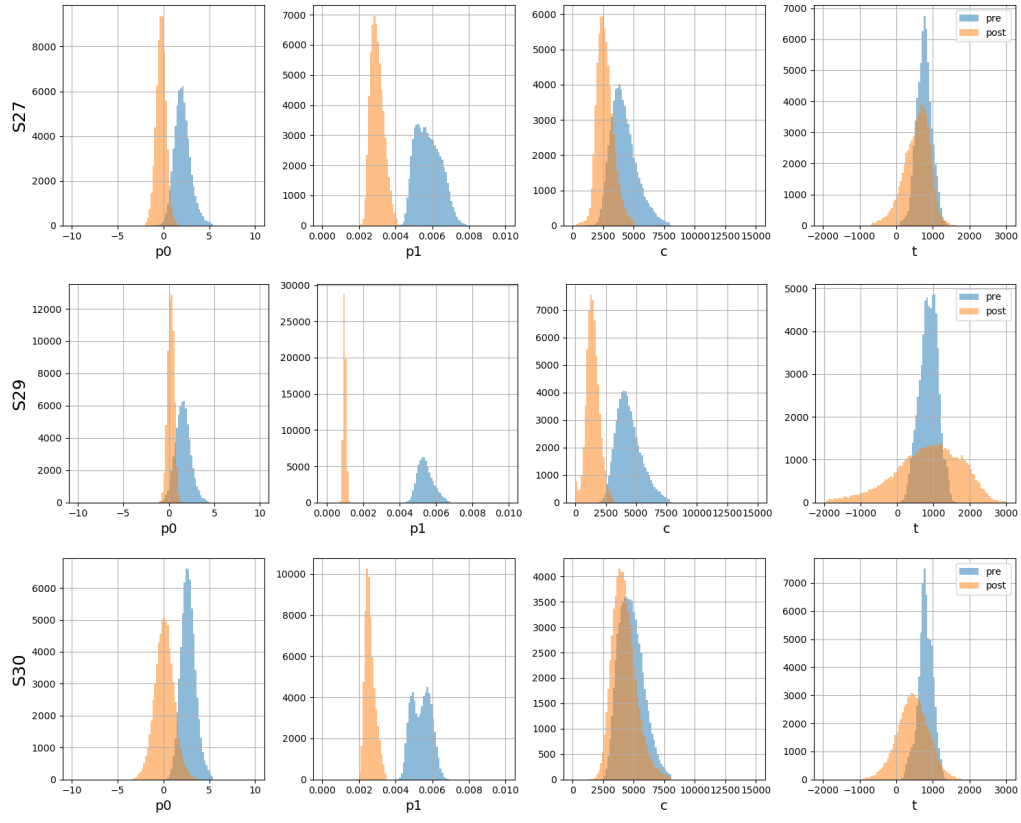


Figure 4.8.3: Part 2 of the distributions of surrogates parameters.

4.8.4 The fluence

As mentioned previously, the sensor S8 (@TODO maybe use different name here), was the only one that had a well documented non-uniform irradiation. The irradiation profile is visible in the plot (@TODO) add the actual irradiation plot here. For the purpose of this irradiation profile, we devise a binning visible in the Figure 4.8.4. This binning allows for assigning values of fluence per each bin and analysis of

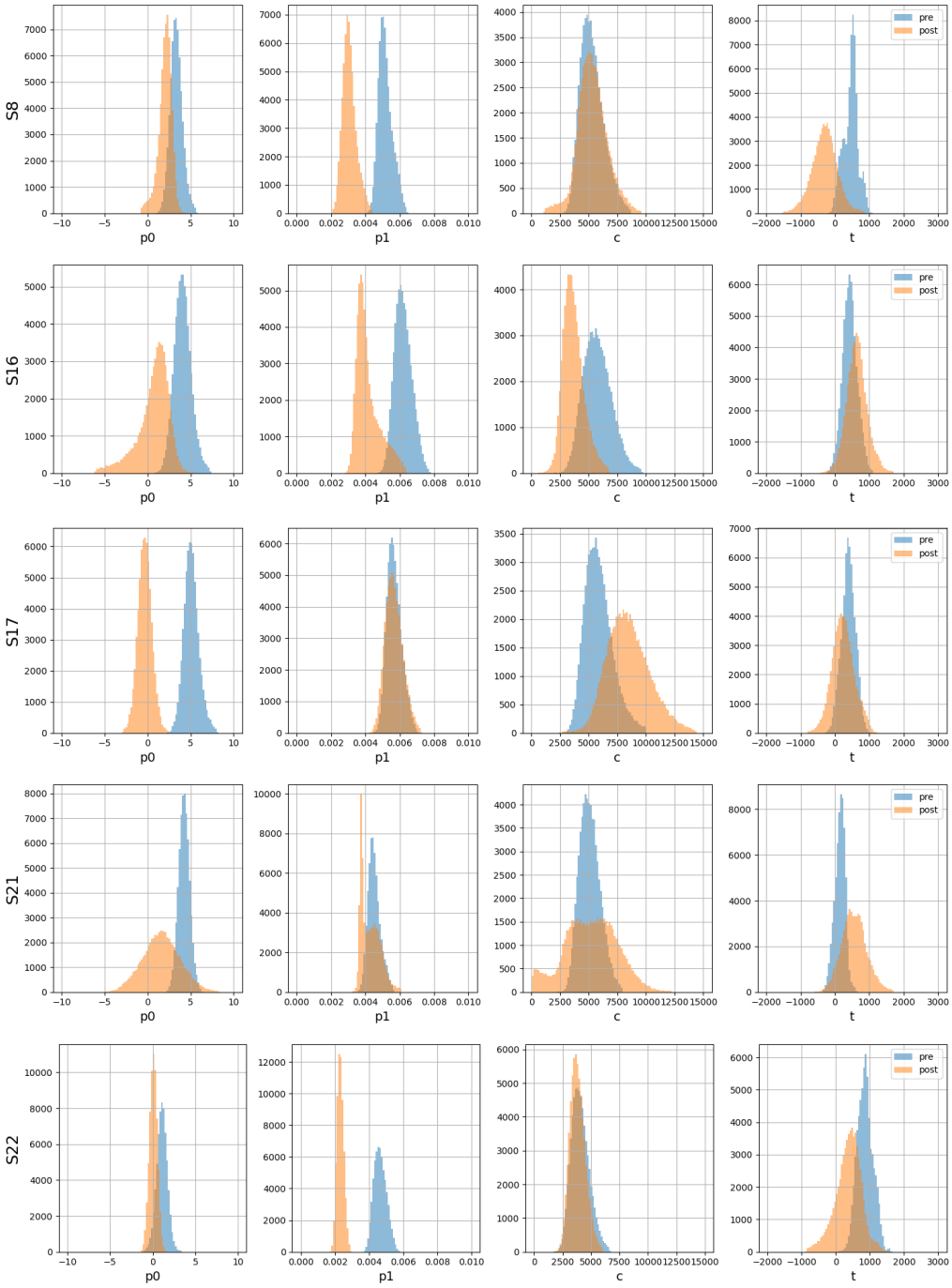


Figure 4.8.2: Part 1 of the distribution of the surrogates parameters in sensors. The rows of plot represent sensors, each of the columns of the plots reperesent a parameter of the surrogate (p0, p1, c, t). The color blue (pre) denotes the distribution before irradiation of a given parameter, the orange color is the distribution after the irradiation of the sensor.

the pixels groups in the binnning. The fluence per bin is as seen in the table 4.8.1, the innermost bin has the highest fluence level. This mean that we expect that the effects of the radiation will be most visible in the center of the pixel matrix.

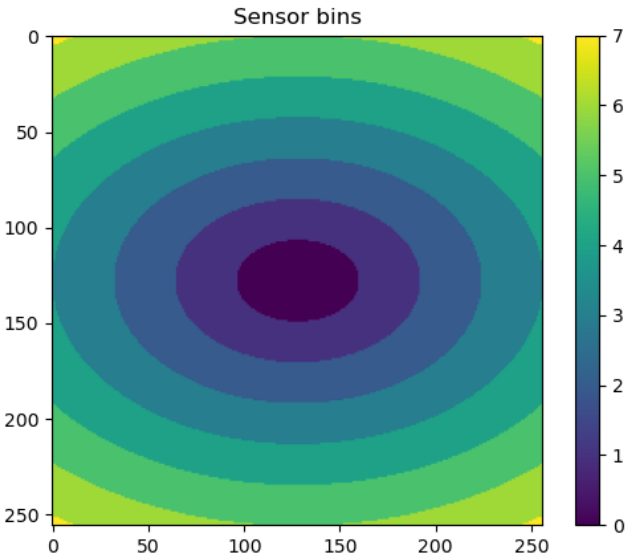


Figure 4.8.4: A heatmap of the binning used in the analysis of the fluence. Each color represents a different elliptical binning. The binning starts from 0 in the center, and the outermost bin is labeled as 7.

(@TODO this tab, is mean of accumulated?) (@TODO change the look of this tab) (@TODO add the area[cm2])

Bin label	Fluence $[n_{eq}/cm^2 \dot{1}e15]$
0	7.504
1	7.187
2	6.595
3	5.760
4	4.766
5	3.655
6	2.681
7	1.779

Table 4.8.1: Table of fluence per bin

The visible shifts in parameters $p0$ and $p1$ distribution visible across the sensors are complimentary to the shift visible in the sensor S8, when comparing the distributions of these parameters per bin. In the Fig. **??** the parameters distributions are depicted as boxplots, per bin, pre and post irradiation. The detailed explanation of used boxplot is availible in the A. One can notice that the $p0$ and $p1$ exhibit lowered values in comparison to the pre iradiated bins. The $p0$ case is the most noticable, and $p1$ although in total is lower, the trend doesn't keep up with the fluency per bins. The behaviour of $p1$ could be explained by the initial inequal distribution of this parameter pre iradiation. The c parameter exhibits

greater variance in parameter distribution with slightly eleveted values. T parameter also shows greater variance, with visible overall decrease.

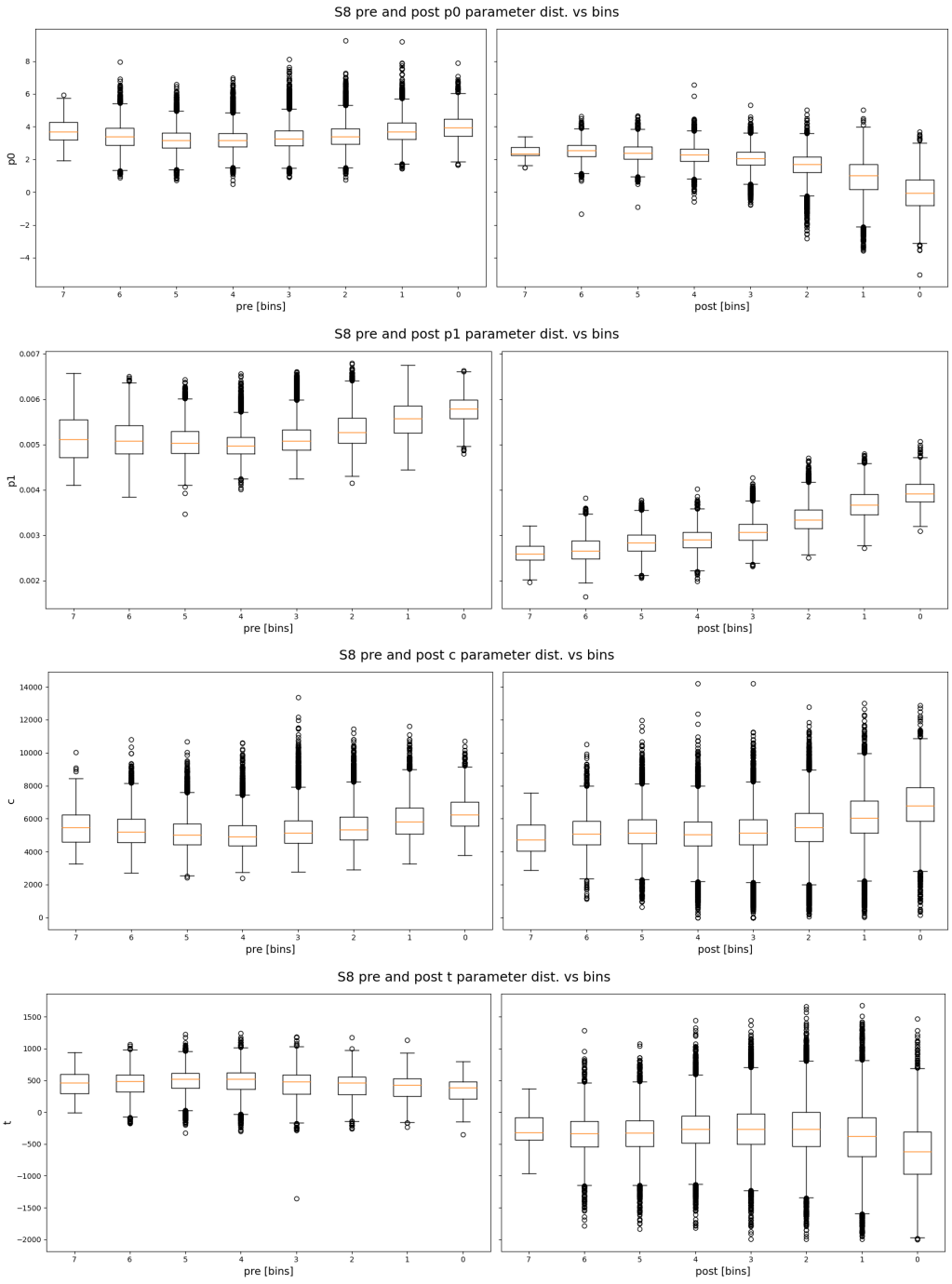


Figure 4.8.5: Parameters distributions as boxplots per bin, pre and post irradiation.

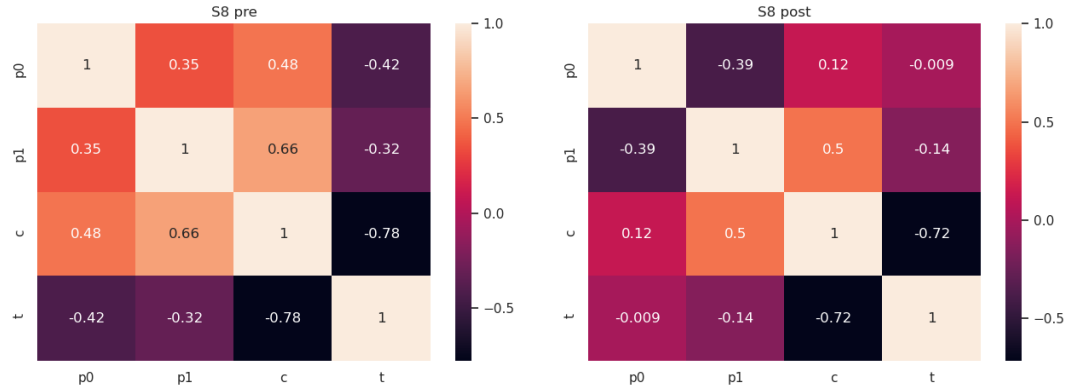


Figure 4.8.6: Pearson corelation coefficients between the surrogate parameters in pre and post irradiation.

When analysing the parameters it is important to note their corelation, as visible It is important to note the corelation of the parameters. The pearson corelation parameters in teh Fig. **??** show that there is a high corelation between c and t pair, as well as c and $p1$. We do not explore the reasons for such corelation, but acknowledge it in further studies.

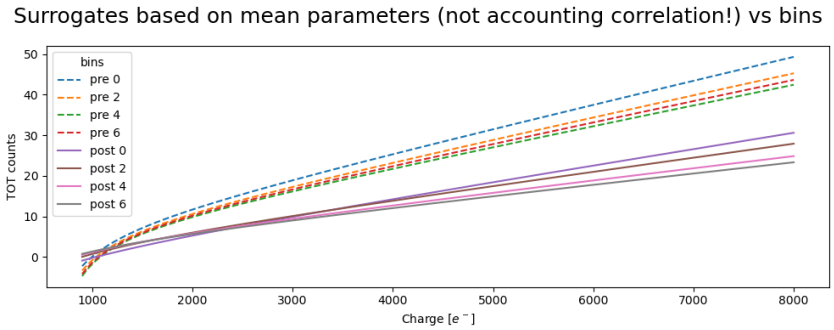


Figure 4.8.7: Examplary surrogates

To better understand what is the changing in the surrogate function we are presenting some examplary surrogates, plotted ussing mean of the parameters present in the bin. Those examplary surrogates in the **??** are just approximations, since we are not decorelating the parameters, just using a simple mean. The examplary surrogates show that for the most part (for more than $1500e^-$ charge) the pre irradiated sensors expressed more TOT counts for the same amount of charge, than after the irradiation. Also, although it doesn't have a physical meaning, the plot shows some part of the surrogates below $0TOT$. This is just to show that the surrogate function changes also in other way; the minimum charge required to create a TOT signal is lower. But for the majority of the charge range it means that the sensitivity of the pixels has decreased, which is something that should be expected of the silicon sensors after irradiation.

Given this change in the sensitivity, we can inverse the problem expressed by the test beam data. Measuring the change in the surrogates expression pre and post irradiation, it might be possible to calculate the fluence based solemly on the data coming from the sensor.

4.8.5 Modelling the surrogates in fluence domain

We propose a following model of surrogate for a given bin.

(@TODO change this list to the nice flowchart if possible)

1. Rescaling
2. PCA

Because the parameters have different ranges of values, the scaling is needed to bring them to the same range. We are using a standard scaler in form of $(z = \frac{x-u}{s})$. Next, because the pearson coefficient indicate that the parameters are corelated, we will use PCA to decorelate. The underlying distributions are not gaussian, and in fact the crystallball distribution would be a better fit. Since it is easy to decoralte the variables using the PCA, which assumes that the variables are of normal (gaussian) distribtion, we will assume gaussian distribution of the surrogate parameters. The PCA internally fits the dimensions of the variables to a 1D gaussian, and it's transformation matrix is actually holding the width and mean information. The PCA decorelation allows then for sampling of the parameters decoralted space using 4 random gaussian generators and the following procedure;

1. Inverse PCA
2. Inverse scaling (x = (z*s) + u)

This process is usefull for checking the loss of the quality of the parameters resulting from using normal distribution instead of crystall ball.

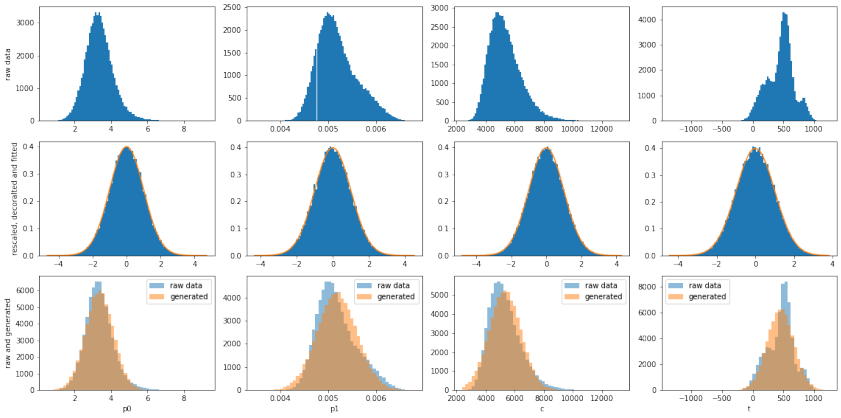
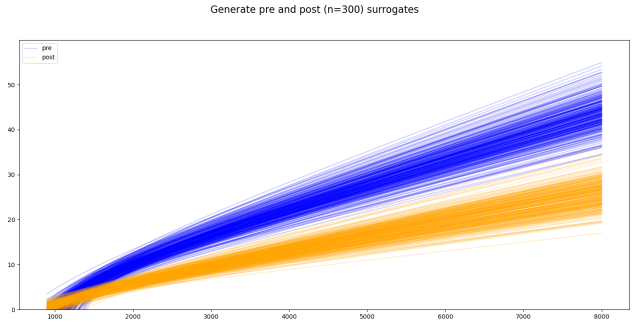
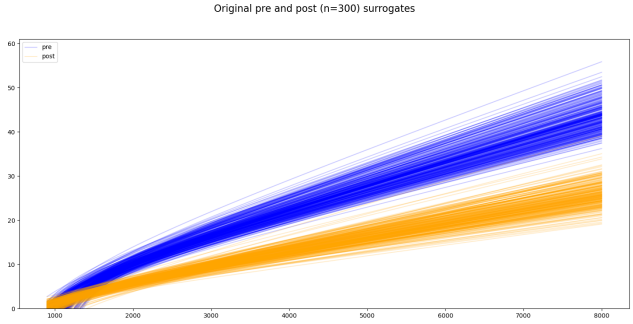


Figure 4.8.8: Examplary case of surrogate model on the entire sensor. Each column of the plots representes different parameter of surrogate function. First row of plots is the total distribution of parameterrs in the data. The middle row shows the distribution after scaling and PCA, with gaussian function outline in orange. The bottom row shows the same actual distribution as in first row in blue, and a distribtion of generated parameters using the inverse model procedure and generating samples from underlying gaussian distributuion.

(@TODO make proper distribution (non 0 and 1 mi std))

Fig. 4.8.8 depicts the model of the surrogates applied to the entire sensor (including all bins). It is visible that the range of the parameters is roughly covered with the generated distribution. By separating this model on per bin basis we can connect the parameters distribution with the fluence.





Additional test for the model is in the Fig **??**. The plot depicts the combination of parameters into surrogate function.

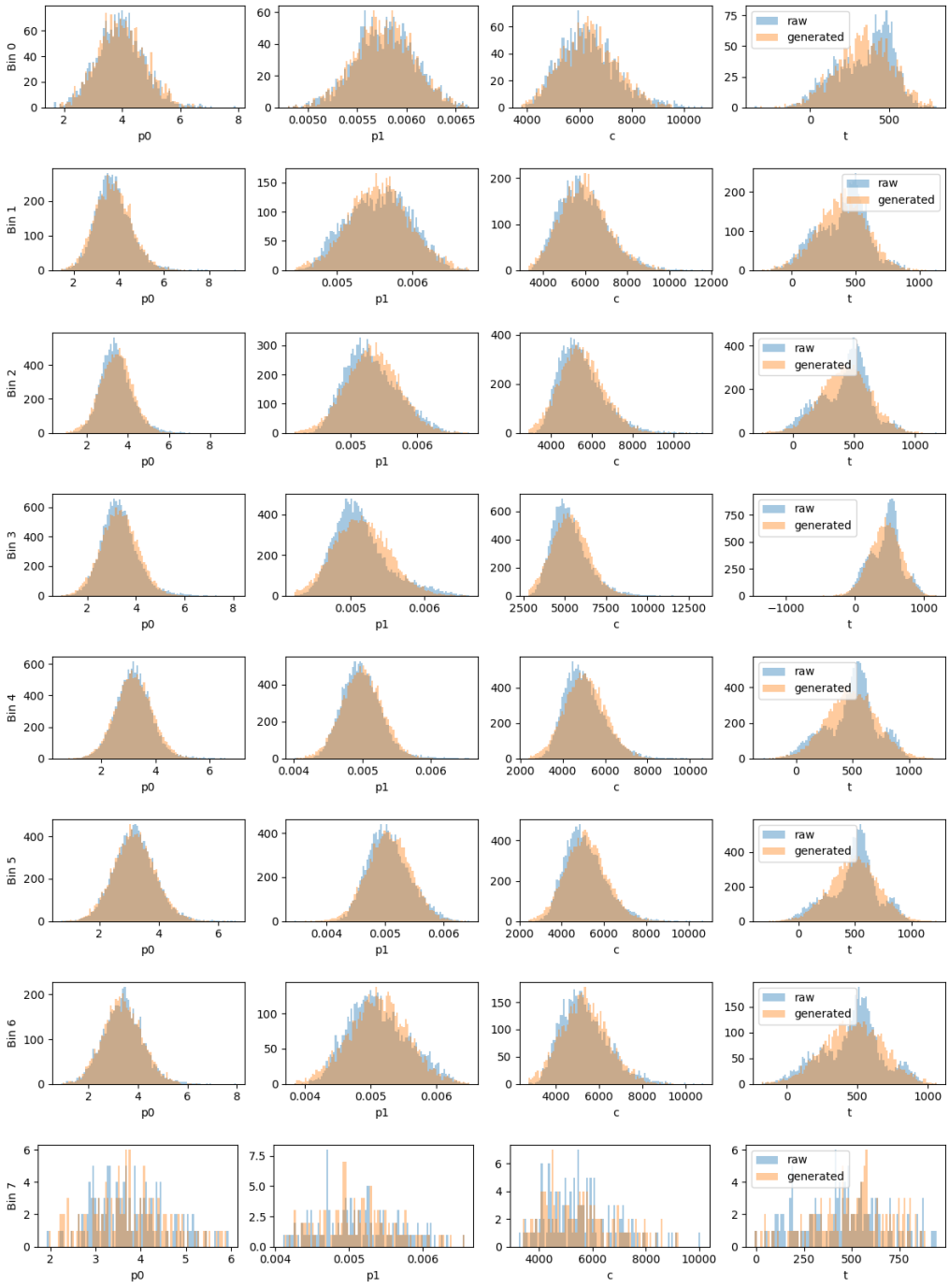


Figure 4.8.10: Actual distribution of parameter in given bin, with an overlay with a distribution generated with inverse model in orange.

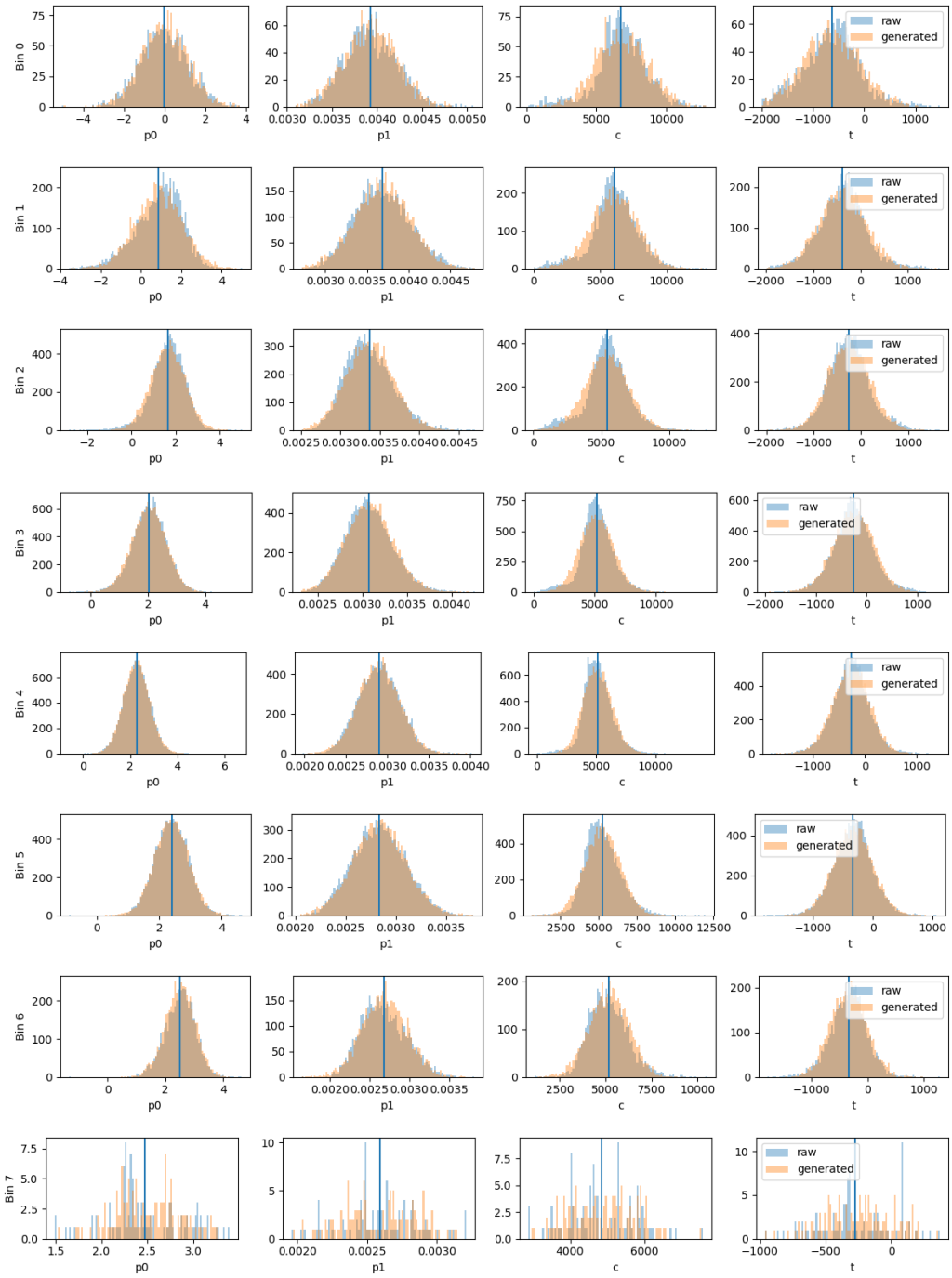


Figure 4.8.11: Actual distribution of parameter in given bin, with an overlay with a distribution generated with inverse model in orange.

In the Figures **??** and **??** the model is split between each of the bins. It is visible that in both pre and post irradiation cases the distributions roughly cover the same scope as in the actual data.

4.8.6 Model of fluence

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo.

Quoteauthor Lastname

5 Software for VELO

(@TODO add why this was needed)

5.1 Storck

Storck stands for D**ata** **Stor**age and Tra**ck**ing .It is a database system, created for the purpose of storing Velo calibration data. It was developed using python and django framework. (@TODO too reference that) Ath the time of writing, it is being adapted ad CERN for use in the comissioning of VELO. This section will present it's design. The details of the implementation can be reffered to in the CERN's gitlab repository link in the bibiliography**?**

5.1.1 Motivation

The Scientific workflow when preparing or managing a high energy particle detector requires taking vast ammounts of different types of data. Apart from the immidiate data coming from the device other data types, such as conditions of the test or the detector (such as temperature, date, duration of the test) are needed as well. The developement process might be messy, and evolve in time. This is also true for the monitoring tasks when using the detector for taking measurements. On the other hand, relational databases require predefined structure, that is not easy to be modified when the data have been filled. Relational and non-relational databases tend to be not the best solution when used to store binary files. For those reasons, the Storck project was created. It utilises relational database with non-relational elements with filesystem storage. It is not oriented towards a one type of files, nor does it expect any kind of predefined structure. Storck allows to share data between it's users, and track the changes in the data.

5.1.2 Software Technology

5.1.2.1 Django and Django rest api

Python language provides many web frameworks. One of the most popular of them is Django. (@TODO add reference to website https://www.djangoproject.com/) Django is mature web design framework, based on the Model-View-Controller (MVC) design pattern. Django is splitted between Model, View and Template, thoigh tt's important to note that functionally Django View corresponds to Controller, and Template to View parts of MVC. This design pattern allows for logical separation the functional responsibility of the code. Furtherly, Django's design consist of apps. Each of the apps implements their own version of MVC, though it is possible to use any of the component of the MVC design from any other app. This framework also utilises object–relational mapping, which is freeing the framework user from writing queries using SQL language, and allows for high level object oriented usage of the model.

5.1.2.2 PostgresSQL

PostgresSQL is popular implementation of relational database. It is SQL compliant. Importantly, it also has non-relational capability. It i

5.1.2.3 Docker

Docker provides OS-level virtualisation. It uses images and containers. Docker image is a software package that contains everything that is necessarry to run an app. Images are defined in special files called Dockerfiles. Image definition is done by using Dockerfile commands. Images are usually based on operating system's, like Ubuntu Linux or Alpine Linux. The former is by far the most popular choice, as the base image from alpine linux needs only 8 MB of disk space. When Images are being mounted to the docker daemon, they are reffered to as containers.

5.1.3 Service Design Overview

The design of the system at heart uses REST API communication for managing the access to the database. Users have have multiples ways of interacting with Storck; but all of them internally use REST API. Users can use the web interface, and manually input or download the data. For the purposes of the ease of use, there also exists a python wrapper to the REST API, so users can easilly create scripts that will interact with Storck. When the web server receives commands via REST API, it interacts with PostgresSQL in order to validate the request, and responds using HTTP request, either by providing requested information (like list of files, file details) or by sending the file. Because http servers may be a bottleneck when serving large ammounts of data, we implemented possibility to download the data using direct files access. When storing files, storck doesnt write them directly to the database, but it saves it on the disk. By the design of the CERN's computing ecosystem, this disk space is accessible by ssh connection with the CERN's lxplus servers. The Fig. 5.1.1 contains schematic diagram for the data flow in the Storck.

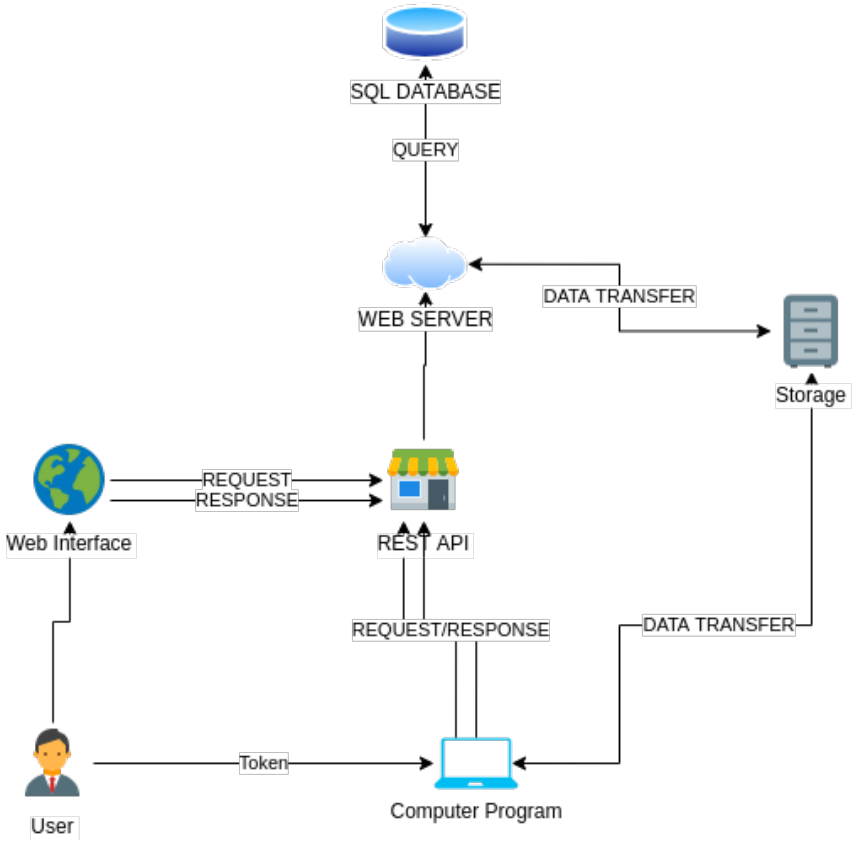


Figure 5.1.1

5.1.4 Implementation

The overall implementation spans over a few django apps. We will not present here parts of the code, as we feel this would be excessive. Yet we provide an overview over the most important parts of the implementation of the service

5.1.4.1 Files

The most important element of the Storck is the StorckFile model/database. It records the files uploaded, and stores proper information. The *id* field contains an unique id that identifies each of the files in the database. (@TODO make name cursive) Next, the file, which is of django's File field type, contains a path to locally saved file. This property is not implicitly created as the request to file with its contents comes in, but instead it is created with the deduplication procedure described in Sec **??**. Hash is the md5sum of the contents of the file, calculated upon receiving. Workspace and user contains a foreign key connection to the list of all workspaces and users. User field contains user id of the user who uploaded this file (or its duplicate). previous_version points to the record containing the previous version of the given file. Here "new version" means a new file in given workspace with the same "fake_path" attribute. (@TODO check if this is really coverd in the code) duplicate_of contains a reference to the files duplicate record. fake_path is the logical path of the file. This property has no physical sense for the Storck database, as the files are not stored using this path. meta_open is the the metada property. It is JsonField, which makes it flexible, and allows to create arbitrary structure of the data inside of the

Property	Django model type
id	AutoField
file	FileField
hash	TextField
workspace	ForeignKey
user	ForeignKey
previous_version	ForeignKey
duplicate_of	ForeignKey
date	DateTimeField
fake_path	TextField
meta_open	JsonField
meta_closed	TextField
hide	BooleanField

Table 5.1.1: Table of contents of the StorckFile model.

field. Because we don't want to show older versions of files in the web view, we use flag hide to decide whether the file is shown or not.

5.1.4.2 Authentication

The CERN's ecosystem allows for authentication of CERN users*. It uses (OIDC) OpenID Connect system, whichi was build on top of OAuth 2.0 framewrok. OIDC system allows for third-party identification. (@TODO refernece OIDC, OAuth) What that means is that instead of requiring a proprietary login password in Storck, we can use redirect the login to CERN SSO (CERN single sign on) service. There, user can use their CERN Account password and login, the same that is being used for other services at CERN to log in. Automatically, in the background, CERN SSO communicates to the web server and confirms the identity of the user. On the administrator side, setting up the authentication with OIDC requires setting proper tokens. Those can be generated using the CERN application portal. Although in the section we refer to CERN's procedures of authentication, for anyone willing to reuse Storck other purposes, it is possible to do that with any authentication system that implements the OIDC. None of the parts of this process is specifically dedicated only to the CERN ecosystem.

*each person at CERN can use CERN account to authenticate or log in to multiple services.

5.1.4.3 Deduplication

The deduplication process is used when the file is about to be saved in the storck. The contents of the file must be received by the server, and when they are, server calulates the md5sum hash value. (@TODO reference the md5sum) The database is then checked for the hash matching the file. Is the same hash exists in the database, then the record containig this hash is the duplicate of the incoming file. If the file has no duplicate, it is processed and saved normally. But if the duplicate exists, the received file content is discarded. The entry to the file database is progressed normally, and the file path to the incoming file is set to be the same as for the already existing file. Addiitonaly the attribute duplicate_of is also set to point to the id of the entry of the existing file record.

5.1.5 REST API

The REST API makes for a perfect interface for project like this. (@TODO add reference to REST AO) Most of the modern programming languages are capable of making HTTP request be themselves, and network connection is usually necessarry for any operation. REST API is agnostic of an operating system, and programing language. In my previous experience with LHCb monitoring, one of the biggest challenges was compatibility of the database tools, which only interfaced by C++ library. The design of STORCK is free of this problem, as the client of the system doesn't need to update every time there is a change in the Storck service.

method	path	params
GET	/api/workspaces	
POST	/api/workspace	name (body)
POST	/api/workspace/user	workspacetoken (body)
		userid (body)

method	path	params
GET	/api/files	hidden (query)
		token (query)
GET	/api/file	info (query)
		id (query)
		token (query)
POST	/api/file	token (query)
		file (body)
		path (body)
		meta (body)

(@TODO consult this subsection with the code) The Tab. tab:workspace-api, tab:file-api, present the listing of the REST API methods.

The rest API consists of two endpoints

5.1.6 Deployment

The novel deployment process standard in the IT industry is using Dockerisation. And so, Storck in it's deployment depends on it.

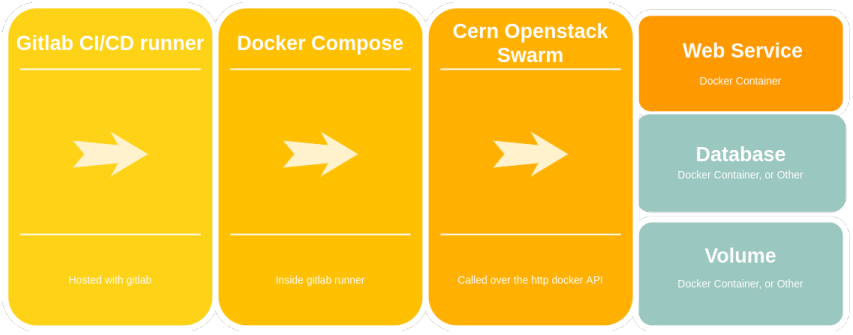


Figure 5.1.2: The schematic of Storck deployment process. The light green color means optional components.

The Fig. 5.1.2 depicts the flow of the deployment for storck. The lowest level of the deployment consists of 3 docker images, Web App, Database and Volume. The last two are optional as in fact, it is possible to use external (not dockerised) Volume, and external Database service. Database can be runned simply using a predefined docker image. Web App is creating using custom Dockerfile. The base image is the Alpine Linux image with python 3.8. During build of the image, it set's up necessary environment variables, dependencies, migrations, and in the end, runs the uwsgi server. (@TODO reference uwsgi server)

5.1.7 Gitlab CI

As visible in the Fig. 5.1.2, the Gitlab CI running environemnt is the first part of the deployment process. Although gitlab started purey as a service that allows for hosting and browsing the git repositories, like many other git services, it provides mechanism for Continous Integration / Continuous Deployment. What it means is that ther is no longer a need for a dedicated server that will be running test environment, and a manuall triggering of some deplyment or testing jobs. It is now perfectly possible to do those things from the gitlab web interface.

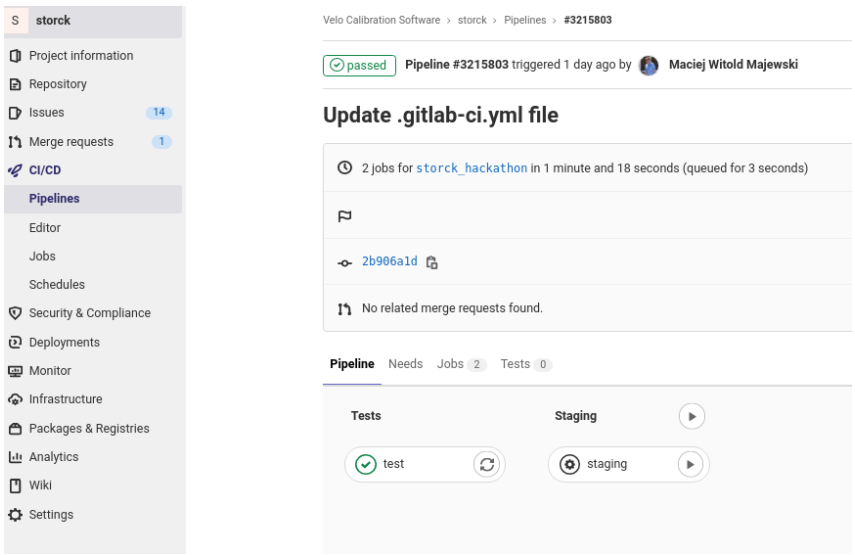


Figure 5.1.3: view of a gitlab pipeline

Automatic deployment can be set in various ways. In case of Storck, the pipeline it was set up so that

it would run unit tests on every git commit pushed to any branch on gitlab. Then, if the test did not fail, it is possible to use a button to deploy Storck to staging server. Only on the master branch, after succesfull deployment of staging it is possible to deploy to the production server. Fig. 5.1.3 shows a view of a single pipeline in gitlab.

5.1.8 Storck web interface

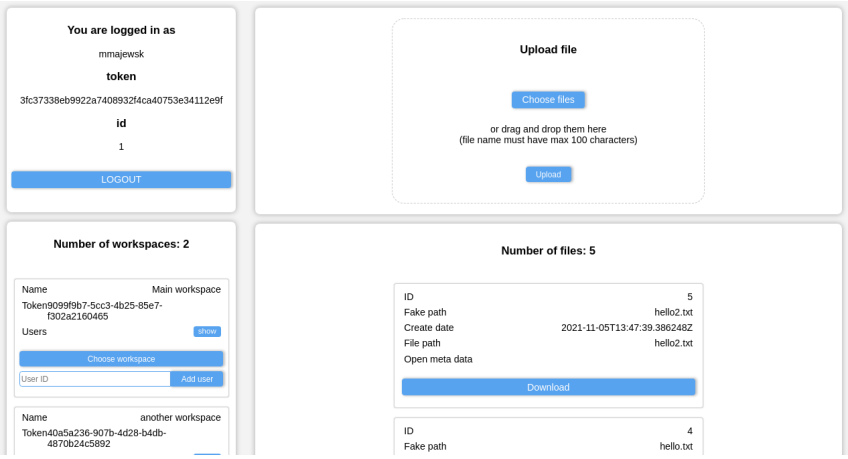


Figure 5.1.4: Storcks web interface.

(@TODO maybe add more in the description)

Storcks web interface is very basic, as it is not meant to be the main way of interacting with storck. In the Fig. 5.1.4, on the left side there is a space that contains users username, id and user token. Below that space there is a list of avialible workspaces. Each of the workspaces shows it's workspace token, and also allows to add users to the workspace. The two areas on the right side of the screen are devoted to files. The top area can be used as drag-and-drop area for the files to be uploaded to storck. The lower area lists all of the fiels contained in the workspace, and button for downloading.

5.1.9 Storck python client

Storck python client was created as a separated project and repository to the main storck project. To use it, python users can simply install it as a package using a command pip install git+https://gitlab.cern.ch/velo-calibration-software/storck (@TODO make this install line look better). It is a very small and simple package. The very basic usage is as follows:

from storck client import StorckClient

client = StorckClinet(api$_host$ = ..., *user_token* = ..., *workspace_token* = ...)*print*(*client.get_files*())

(@TODO) ake it a code snippet

The StorckClient class can be used to make an instance of the client. It accepts three arguments, which are as follows: the http adress to storck server, user token and workspace token. Thise are optional as instead an environmental variable can be used. The client has several methods which we present in form of a table. For a more detailed description of all of the methods we refer the reader to the source code.

Method	description
auth_verify	verifies if the credentials are correct
set_workspace$_{t}oken$	sets the workspace token for the object
create$_{w}orkspace$	creates new workspace
get_files	outputs list of files currently present in the workspace
get_file	returns all information about given file
upload_file	uploads the contents of the file and its information
download_file	downloads the contents of the file
add$_{u}ser_to_{w}orkspace$	adds user to workspace

Table 5.1.2: Table of available methods of storck client

5.2 Titania

Titania is a monitoring framework built with python, Qt and flask. (@TODO refernce those) As well as Storck it is being adapted at the Velo group for the purposes of the next runs at LHCb. This sections will present the design and examplary uses of the framework.

5.2.1 Motivation

Drawing from the experience of working with monitoring in Run 1 and Run 2, we have proposed a new project for handling the monitoring tasks. The previous software for the monitoring, although useful, was getting difficult in the developement due to the ammounts of different uncoordinated changes. The source code of the platform was not well structured, so adding new plots was usually quite the challenge. This is what drove us to developing a new platform for monitoring VELO as a python framework.

5.2.2 Software Technology

Python is one of the most popular tools for plotting and visualistaion in the scientific community. Famously, the first image of the black hole was created with python, and even in the footage from the first flight on mars the use of matplotlib can be seen. (@TODO check and reference the sources) Titania was designed with the desktop GUI in mind first (it also contains some web interface capabilities). So naturally, we used the PyQt library, which interfaces the QtGUI library to python. QtGUI in short provides easy way to create cross-platform GUI applications. It is object oriented library, and impements system of sockets and connections (@TODO check if correct words are used)

5.2.3 Framework Design

As previously stated, one of the key goals of the new system is extensibility, and orderliness. Inspired by django's concept of implementing MVC in the framework, we define a 3 key components to any monitoring tasks

1. Data
2. Plots

3. Exploration

Those are a fundamental blocks of any monitoring tasks.

Data here is understood in terms of source of data. Anytime we wan't to show some data we need to access them. There is multitude of eays that this can be done; reading files from a disk, reading a stream of data, receiving data as http request and others. This also means a format of the data. Data can be text file of csv structure, binary file, image or sound recording.

Plots are the graphical representation of the data. This means linear plots, or scatter or histogram and others. Each of the ways of f representing the data has it's own custom options of placement and colours and other things.

Exploration - one must be able to choose between diferent data source, and different plot types. If a monitoring service shows only one plot from a one data file, it can hardly be a system.

5.2.3.1 Web Interface

5.2.4 API

5.2.5 Implementation

5.3 Conclusions

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo.

Quoteauthor Lastname

6

LARTPC detector

6.1 Neutrino experiments

6.2 LARTPC structure

6.3 LARTPC simulation

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo.

Quoteauthor Lastname

7

Reinforcement learning for LARTPC detector

8

Conclusion

Lorem ipsum

A

Appendix A - boxplot

The explanation of statistical meaning of the boxplot is at Fig A.0.1

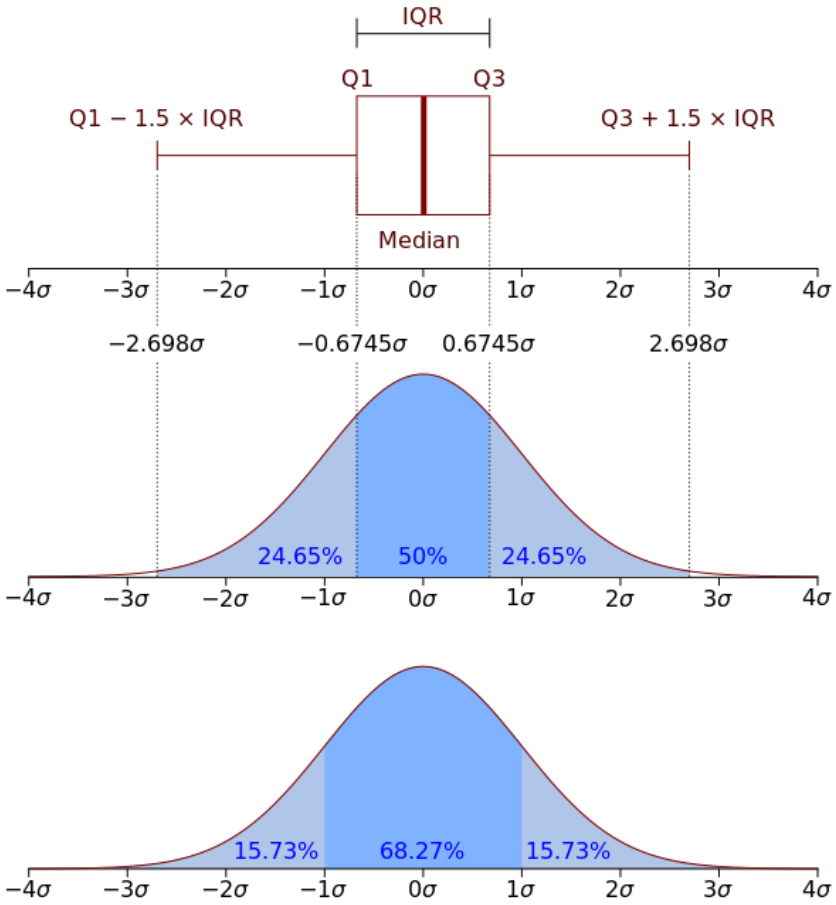


Figure A.0.1: A plot explaining the scope of the boxplot (@TODO add this to appendix). The top plot represent the boxplot, with median in the middle. The range of the box spans from Quartile 1 to Quartile 3. The interquartile region (IQR) is the width of the box. The whiskers span from the edges of the box to length of 1.5 IQR to either side of the axis. The two bottom plots explain the ranges in terms of percentages and width of the distribution. Additionaly, an outlier is a value lying outside the both whiskers range.

References

[1] Collaboration, L. (2013). *LHCb VELO Upgrade Technical Design Report*. Technical report.

[2] F.R.S., P. E. R. (1911). Lxxix. the scattering of α and β particles by matter and the structure of the atom. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 21(125), 669–688.

[3] Heisenberg, W. (1932). Über den Bau der Atomkerne. I. *Zeitschrift fur Physik*, 77(1-2), 1–11.

[4] Iwanenko, D. (1932). The neutron hypothesis. *Nature*, 129, 798–798.

[] Majewski, M. (2021). Storck - cern repository. https://gitlab.cern.ch/velo-calibration-software/storck. [Online; accessed 10-Nov-2021].

[5] TAYLOR, C. (1999). *The Atomists: Leucippus and Democritus: Fragments*. University of Toronto Press.

[6] Thomson, J. & Institution, S. (1901). *On Bodies Smaller Than Atoms*. Annual report - Smithsonian Institution, 1901-1902. U.S. Government Printing Office.

[7] Tsopelas, P. C. (2016). *A Silicon Pixel Detector for LHCb*. PhD thesis, Vrije U., Amsterdam.