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

Abstract (Polish Version)

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.

Abstract (English Version)

This work brings together multiple projects revolving around particle detectors and novel applications of computational intelligence for many of their problems. As an expert and enthusiast of AI, my main focus of the study is the application of machine learning methods for particle detectors. The core of this document relates to the LHCb spectrometer and its Vertex Locator detector. The LHCb is part of the Large Hadron Collider system of detectors', and its discoveries contributed to the global understanding of physics. The two data-taking periods, Run 1 (2011-2012) and Run 2 (2015-2018), sum to the integrated luminosity of around $9fb^{-1}$. Technical stop (after Run 2) brought and upgraded the detector,

migrating to the pixel-based matrices. The unique requirements and conditions in which the Velo detector is required to perform require individual research for the methods of monitoring and analysis of the state of the sensor, which is presented in this work.

The beginning chapters, up to and including Chapter 3, contain the necessary theoretical and technical introduction to the details of the physics goals, detector composition and analysis methods. The chapters touching on the unique analysis of the calibration of the Velo detector in Runs 1 and 2 are contained in Chapter 4, and the study of the upgraded detector is contained in Chapter 5.

The first portion of the presented studies investigates two calibration parameters derived from the noise signal picked up by the detector's silicon sensors, pedestal (mean of noise) and threshold (standard deviation of the noise). Those parameters are investigated in views of different dimensionalities. The analysis of the trend of the pedestals has shown no persisting effects, which confirms that the adaptation of the voltage fed to the detector was in acceptable ranges. The analysis of the threshold parameters revealed the effects of the harmful radiation on the sensors. It inspired the creation of a novel machine learning-based algorithm for the assessment of the calibration itself. I created and implemented an algortihm called "outlierness" and successfully introduced it to the monitoring of the Velo detector in 2018.

The high dimentionality of the problem (vast amount of unique sensor strips) led to the tests of dimentionality reduction algorithms (PCA and autoencoder) and showed that both algorithms could show outliers in the functioning of the detector. The subtle shifts in the relation of the calibration to the noise found in the detector during data taking led us to leverage that effect and use a recurrent neural network to find a method for predicting the need for the calibration of the detector.

The new Velo detector in the upcoming run 3 uses pixel matrices for the data taking and increases the number of individual sensors even further. I participated in the creation of a novel method for finding and tracking clusters of masked pixels in its calibration, which will allow for a more informed decision about detector maintenance.

While most of the helpful methods for analysing the calibration of the detector can only be used while the detector is already fully commissioned and functioning, the test data of the sensors can be used for studying the effects of the radiation. The new VeloPix sensor registers a time-over-threshold signal, which can be linked with the energy deposited in the sensor expressed in eV via surrogate function. The change of surrogate parameters is linked with the amount of total radiation delivered to the sensor. I

studied this effect, which led to the development of the breakthrough intelligent method for assessing the fluence of the sensor.

The experience of working with Velo has led to developing the necessary and novel solutions to practical problems presented in Chapter 6. The experience of working with the calibration data has led to the creation web-based database service called Storck, which is undergoing the introduction to the LHCb systems during the commissioning of the detector. Along with the database system, an open source framework for efficient creation of the monitoring tools - Titania is presented. Storck and Titania are generic open-source tools that can be used independently in any experiment. I led the development of both projects, along with several developers.

The applications of the machine learning methods for the High Energy Physics detectors usually show high specificity and customisation to the undergoing physics search and design of the detector. While it is usually necessary to develop solutions to particular problems, the problem of tracking and particle identification can be thought of as more universal. The work in generalising machine learning algorithms for particle physics experiments has been largely untouched. The large and robust open dataset created by DeepLearnPhysics has created an opportunity for early search for a more general AI capable of understanding the rules of our universe. The opportunity was leveraged by an ambitious attempt to research the reinforcement learning method's application for particle tracking and identification in the LARTPC dataset. This research is presented in Chapter 8, with a theoretical introduction in Chapter 7.

Additionally to the research work, my duties included conducting AGH course Python in the Enterprise. Throughout the years, I taught a total sum of around 250 students how to write clean, good, reliable pythonic code and provided guidance for their course-related projects. The students also learned how to use the GIT version control system and how to collaborate with other students on projects using Agile software development, Scrum, Continuous Integration and Deployment tools, as well as how to apply machine learning to real-world engineering problems.

Below is the list of the contributed papers, as well as conference and schools participation.

Publications

- **title**,

  text

- **Analysis, and machine learning anomaly detection of the VELO-LHCb calibration**,

Journal of Physics: Conference Series, 2nd Trans-Siberian School on High Energy Physics, 1-5 April 2019 (proceedings)

- **Automatised data quality monitoring of the LHCb Vertex Locator**,
  Journal of Physics: Conference Series (2017) 2017 J. Phys.: Conf. Ser. 898 092046
- **Development of the LHCb VELO monitoring software platform**,
  Acta Phys. Pol. B 48, 1061 (2017)
- **Mapping the material in the LHCb vertex locator using secondary hadronic interactions**,
  2018 JINST 13 P06008
- **Phase I Upgrade of the Readout System of the Vertex Detector at the LHCb Experiment**,
  IEEE Trans. Nucl. Sci. 67 (2020) 732-739
- **The upgrade I of LHCb VELO—towards an intelligent monitoring platform**,
  2020 JINST 15 C06009
- **Software Platform for the Monitoring and Calibration of the LHCb Upgrade I Silicon Detectors**,
  Acta Phys. Pol. B 50, 1087 (2019)
- **Simulation and Optimization Studies of the LHCb Beetle Readout ASIC and Machine Learning Approach for Pulse Shape Reconstruction**,
  Sensors 2021, 21(18), 6075;
- **Readout Firmware of the Vertex Locator for LHCb Run 3 and Beyond**,
  IEEE Trans. Nucl. Sci. 68 (2021)
- **Radiation Damage Effects and Operation of the LHCb Vertex Locator**,
  IEEE Trans. Nucl. Sci. 65 (2018)
- **Phase I Upgrade of the Readout System of the Vertex Detector at the Experiment**,
  IEEE Trans. Nucl. Sci. 67 (2020)

Conferences

- **Development of the LHCb VELO monitoring software platform**,

  23rd Cracow Epiphany Conference, Kraków, 9 - 12 Jan 2017 (proceedings);

- **A TCAD based model of double metal layer effects and a review of the radiation damage and monitoring of LHCb Velo**,

  13th "Trento" workshop on Advanced silicon radiation detectors, 19-21 February 2018, Munich;

- **Machine Learning in High Energy Physics**,

  Data Science Summit, Warsaw, 14 June 2019

- **Machine Learning in LHCb**,

  ML in PL, Warsaw, 22–24 November 2019 (poster)

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

  Symposium Artificial Intelligence for Science, Industry and Society, Mexico, 20-25 October 2019

- **LHCb Velo upgrade software platform**,

  10th LHC students poster session, Geneva, CERN, 19 Febuary 2020

- **Unsupervised learning for pixel mask clustering and cluster tracking in LHCb's Velopix sensor calibration** - awarded best talk in young researchers session

  4th Jagiellonian Symposium on Advances in Particle Physics and Medicine, Kraków, 10-15 July 2022 (proceedings)

Schools

- **Machine Learning in High Energy Physics Summer School**,

  Oxford, 5–12 August 2018

- **2nd Trans-Siberian School on High Energy Physics**,

  Tomsk, 1-5 April 2019

- **42nd CERN School of Computing**,

  Cluj-Napoca, Romania, 15-28 September, 2019

- **5th International Winter School on Big Data**,

  Cambridge, 7-11 January 2019

Grants

- **Intelligent monitoring software and tests of silicon pixel sensors for the modernized peak detector in the LHCb experiment.** - principal investigator,

  Preludium 14, National Science Center (NCN), 2018-2020

Acknowledgments

I acknowledge support from National Science Centre (Poland), UMO-2017/27/N/ST2/01880.
This research was supported in part by ACK CYFRONET AGH (PLGrid Infrastructure).

One of the key insights from one of my favourite books is that, no matter how small your stride is, as long as you take enough steps, you'll be able to travel further than you ever imagined. Saying this, I would like to express my gratitude to everyone who has helped, accompanied or made any of the steps towards finishing this work more enjoyable.

First, I would like to thank my supervisor Tomasz, without whom this journey wouldn't be possible. Thank you for the countless times you have helped me, offered me guidance, believed in my abilities, and gave me space for growth.

I would also like to thank my friend from room 121. Thank you, Bartek, for helping, especially with the VeloPix section of this work, and for often being someone I confide in my problems with.

This journey would undoubtedly be significantly less enjoyable and more difficult if it weren't for Wojtek, Paweł and other members of the LHCb-AGH and KOIDC. Thank you for being so helpful with answering many of my questions and with formal issues, and also for being excellent travel companions. I wish you good luck in your future careers.

The majority of the scope of this work touches on the LHCb experiment at CERN, so the thanks are in place for the entirety of the LHCb collaboration, especially to Victor and Kazu. Being part of this collaboration has been the highlight of my scientific career. It always will be an experience that I cherish and am proud of.

Additionally, I would like to thank all of the students that partially contributed to this work (in no particular order): Jakub, Dominik, Mateusz, Julian, Patryk.

Now I wish to thank all the people that influenced the years of my PhD studies on a more personal level.

I want to thank my friends from CSC: Andrzej, Kimmo, Oliver, and Willem. Thank you for making my experience with the CERN community so much unique. Big thanks to Artur and all of the members of the Pykonik and Pydata Kraków community for being inspiriation for constant improvement of my CS skills. Thanks to Jery for support in both good and challenging times.

The vast amounts of screen time spent on this work would be unbearable without the anchoring in reality, so I would like to thank all of the friends from Hawiarska Koliba for being that anchor. Especially thanks to those with which I could share my thoughts about the PhD experience (Kasia and Dr Czox).

Being a PhD student is a choice of way of life, and so I would like to thank my extended family for being supportive of this choice, and always rooting for me.

There is one person without whom I would have never even thought about picking up a PhD in physics. My highshcool teacher Mr. Andrzej. Thank you for showing me that the passion, enthusiasm and sheer joy of the work are far more critical than having top grades.

I would also like to thank my first teachers: mom and dad. Thank you, dad, for sparking my interest in science and technology, and thank you, mom, for equiping me with ambition and willpower to make great use of it.

Thank you, sister, for often giving me a quiet place to study, and thank you for always having my back, even when on the other side of the continent.

And last but by far not least, thank you, Ola, for your constant, unyielding, unrelenting, rock-solid support throughout the entire course of my PhD, and especially during the last months.

Contents

In the beginning there was nothing, which exploded.

Terry Pratchett

0 Introduction

In a brief moment, a universe happened. After the Big Bang, the matter (leptons and quarks that form protons and neutrons) came to life following a brief period known as inflation. The currently known physics says that an equal amount of antimatter should be produced as well. Yet, we do not observe the universe to be symmetrical, cosmic yin-yang; equal parts matter and antimatter. 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 asymmetry.

The grand promise of the modern physics is the theory of everything; a universe boiled down to a single theory, perhaps even to a single equation. This dream is partially realised by the Standard Model, the theory that describes three fundamental forces (interactions) in physics. Weak nuclear, strong nuclear, and electromagnetic force, come together into a single equation. To some, it may be surprising that all of the mechanics in the known universe can be rooted just in a single theory based on the principle of local gauge invariance. But the emergence of complex mechanisms from simple rules is prevalent in science. Game of Life - a cellular automata simulation, although containing a small set of rules can make increasingly complex structures. Most of the known chemistry that guides the biological processes, can be explained almost exclusively ”just” using the electrical potential and probability.

As particle physicists, we see this emergence of complexity even at a very basic level. By colliding two protons, we see a multitude of emergent possibilities, many different particles, and a vast sea of probable combinations of fundamental building blocks. Studying those combinations of possibilities is the bread and butter of modern particle physics. One of the misconceptions about High Energy Physics is that it is like a grandiose hunt through a murky and empty Forrest for one big prey.

In reality, it is more like being an explorer of new land. Every day we check new jungle parts, expecting to find the trees and maybe some new species of the forest flora, and we see precisely that. On some

days, we revisit some parts to see if we notice anything extraordinary. But still, we hope that we will find an ancient city of unknown civilisation with unimaginable treasure that will change everything that we know. And we can't deny it does not exist until we search through all of the continent. For particle physicists, this treasure is physics beyond the standard model.

This steady, monotone and patient work of checking all of the branches of physics is facilitated by the international community of scientists, of which CERN scientists constitute by far the largest group, with the biggest particle accelerator - LHC (Large Hadron Collider). The LHC is a circular accelerator located in the underground tunnel under the Swiss-French border. A large portion of the physicist working at CERN are parts of four main experiments Alice, Atlas, CMS and LHCb. During my PhD studies, I was privileged to be one of the collaborators in the LHCb experiment. The LHCb experiment discovered so far 54 new particles while analysing the collected data samples. Such extraordinary science done with the LHCb spectrometer couldn't be possible without years of preparation and without overcoming technical challenges posed by the physics and the hardware. Cooling systems, electronics, mechanic design, software, tracking, particle identification, all of the subsystems, and parts of the detector spawn new problems to be solved by the collaboration and the greater scientific community. I am glad to introduce the reader to my thesis, describing some solutions to those problems.

There is no unique picture of reality.

Stephen Hawking

1 High Energy Physics

The idea of atoms - building blocks of nature stretches back as far as ancient Greece, to Leucippus and Democritus**?** , and prevailed through the ages until the beginning of the 20th century, when it was solidified. In 1897 J.J.Thompson discovered an electron using a cathode ray and noticed its negative charge**?** . Later in 1911, Rutherford discovered the nucleus using the gold foil scattering experiment**?** . Shortly after that, in 1932, the model of the nucleus containing neutron and proton was developed**? ?** . These events, along with the development of quantum mechanics, laid a foundation for the development of modern physics. The deeper we wanted to look inside the matter, and the deeper it was probed by the physics, the more different particles were found. This "particle zoo" was later explained with Quantum Field Theory and a small number of fundamental particles. Those fundamental particles and the rules that guide their interactions are combined into Standard Model - the best theory developed by our civilisation so far.

1.1 Standard Model

Standard model (schematically depicted in Fig. 1.1.1) consists of two types of particles: bosons and fermions. Bosons are responsible for the interactions, and fermions constitute matter (Neutrons, Protons, Atoms). Furtherly, the fermions consist of three generations of leptons and quarks. There are six quarks: up, down, charm, strange, top, and beauty. Two up quarks and one down quark combine into a proton, and one up quark and two down quarks combine into a neutron. The combinations of quarks create the particle zoo. In fact, the quarks have never been observed as free particles, which is explained by the colour confinement. The most popular combinations of quarks consist of two (mesons

*Source https://en.wikipedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg

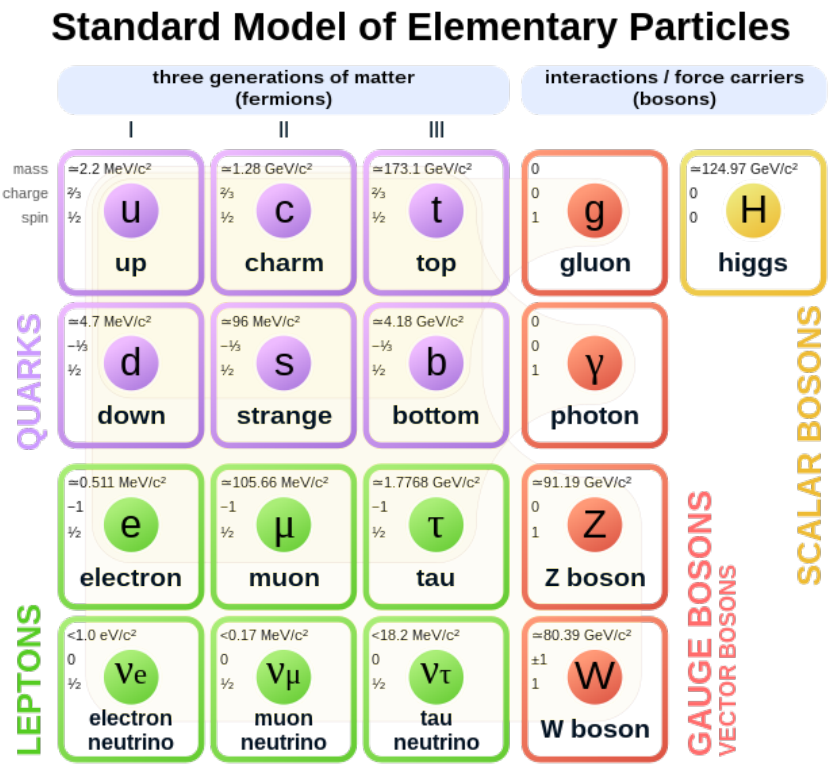


Figure 1.1.1: Standard model of Elementary Particles *

- quark and antiquark pair) and three quarks, with other combinations like pentaquarks much less probable but still possible. Half of the fermions are leptons and have been observed as free particles, unlike quarks. The three charged leptons are electron, muon, and tau; all have an electric charge of -1. The other three are three flavours of neutrinos: electron neutrino, muon neutrino, tau neutrino. The neutrinos are charge-less and have a negligibly small mass. Due to that, they are very elusive and hard to study.

The Bosons are the particles that carry the interactions. There are twelve gauge vector bosons and one scalar boson. The gauge bosons consist of a photon, eight gluons, two charged W bosons and one neutral Z boson. The photon is a carrier of electromagnetic force and is massless and chargeless. The strong nuclear force is propagated by gluons, which, similarly to the photon, are mass and charge-less, but unlike a photon, they can't exist as a free particle and carry the colour charge. The W and Z bosons are responsible for the weak nuclear force. Their exchange can change the flavour of the interacting quark or lepton. The one scalar boson - the Higgs boson, is famously the newest addition to Standard Model. The Higgs boson is responsible for attributing mass to W and Z bosons and fermions.

Additionally to the particles described above, each fermion has a corresponding anti-matter partner. In case of bosons we assume, according to Standard Model, that they are their own antiparicles (apart from W bosons).

The Large Hadron Collider and its accompanying detectors were created to study various implications of Standard Model. A few of the LHCb experiment's primary goals are presented below.

1.2 CP symmetry violation

One of the cosmology's greatest mysteries is the severe inequality between matter and anti-matter in the observable universe. The simple assumption would be that there should be equal amounts of matter and anti-matter if the rules of physics are the same for matter and anti-matter. The most probable cause for such a state is that soon after the Big Bang, the C-, P- and CP-symmetries (**C** for the charge parity operation, **P** for the space parity operation) were broken what led to creation of significantly more matter than antimatter. The CP violation study is conducted to explain this phenomenon.

The parity inversion operator, **P**, can be defined as a mirror inversion:

$$P\psi(r) = \psi(-r)$$

There used to be a belief that physics is invariant under this operation, which was verified in strong and electromagnetic interactions but was not verified in weak interactions**?** . This led to experiments showing that some of the reactions in beta decay do not occur as frequently as their mirror image. The next idea was that although parity itself may not be conserved, the conjugation of the charge parity operator combined with the spatial parity one constitute an exact symmetry. The charge parity operator can be defined as an operator which changes a particle into its anti-particle by changing the sign of its additive quantum numbers (e.g., the charge):

$$C\,|\psi\rangle = |\bar{\psi}\rangle.$$

The combined *CP* symmetry is also broken**?** in weak interactions. The LHCb reported a *CP* violation in B meson**?** and charm decays**?** . The beauty decays can have the CP-asymmetry at the level of 80%.

In the search for the invariant law of symmetry, an additional combination of *CPT* symmetry (*T* is the time-reversal operator) was proposed, which seems to be conserved in all known processes in Standard Model.

Although the studies have shown that there exist symmetry breaking processes in weak interactions of the quark sector, they do not explain the imbalance in matter and anti-matter in the universe. An additional search for symmetry breaking in the lepton sector is also undergoing.

1.3 Tree-level determination of the unitarity triangle angle γ - CKM matrix

One of the most interesting hints of physics beyond the standard model comes from the weak force and its calculation of the CKM matrix values. The CKM (Cabibo-Kobayashi-Maskawa) matrix elements describe the strength of the flavour change in weak interactions. I.e. V_{ud} is the probability of the down quark transforming into an up quark.

$$\begin{bmatrix} d' \\ s' \\ b' \end{bmatrix} = \begin{bmatrix} V_{\mathrm{ud}} & V_{\mathrm{us}} & V_{\mathrm{ub}} \\ V_{\mathrm{cd}} & V_{\mathrm{cs}} & V_{\mathrm{cb}} \\ V_{\mathrm{td}} & V_{\mathrm{ts}} & V_{\mathrm{tb}} \end{bmatrix} \begin{bmatrix} d \\ s \\ b \end{bmatrix}$$

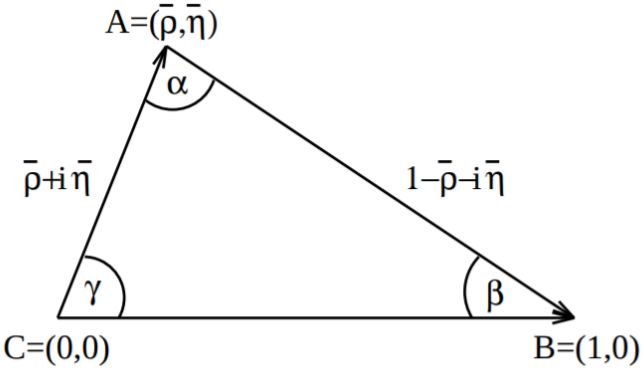


Figure 1.3.1: A unitary triangle (source:?)

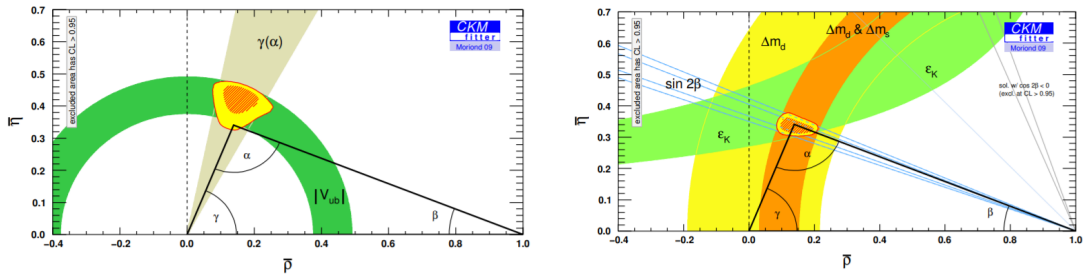
The most recent approximation of the CKM matrix values? , is as follows:

$$\begin{bmatrix} |V_{ud}| & |V_{us}| & |V_{ub}| \\ |V_{cd}| & |V_{cs}| & |V_{cb}| \\ |V_{td}| & |V_{ts}| & |V_{tb}| \end{bmatrix} = \begin{bmatrix} 0.97370 \pm 0.00014 & 0.2245 \pm 0.0008 & 0.00382 \pm 0.00024 \\ 0.221 \pm 0.004 & 0.987 \pm 0.011 & 0.0410 \pm 0.0014 \\ 0.0080 \pm 0.0003 & 0.0388 \pm 0.0011 & 1.013 \pm 0.030 \end{bmatrix}.$$

The CKM matrix can be understood as a rotation matrix in 3D (SU(3)) and, therefore,e can be parametrised by independent angles. It is helpful to use it with Wolfenstein parametrisation? , which allows parametrising the matrix in four independent values λ, A, ρ , and ν.

Such parametrisation allows for representation of the unitarity condition of the CKM $V_{ud}V*_{ub}+V_{cd}V*_{cb} + * V_{td}V*_{tb} = 0$ as a triangle:

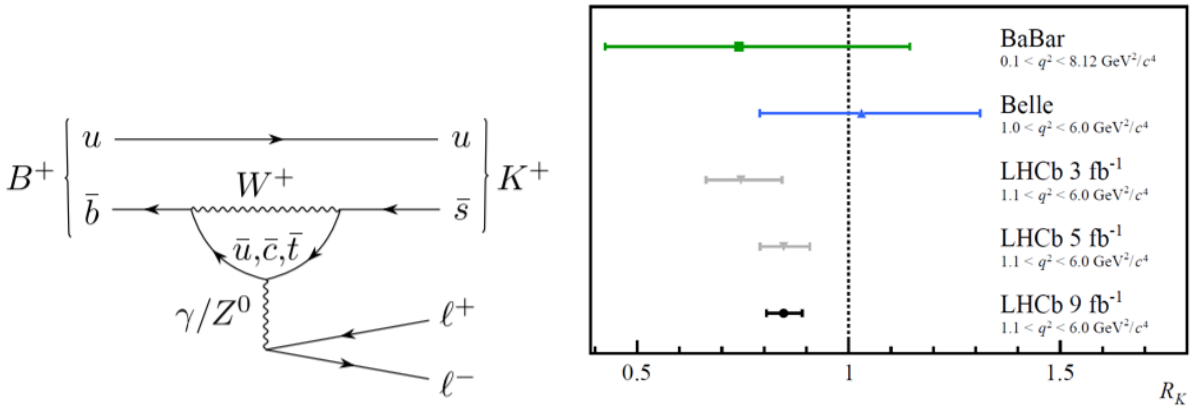
Studies of various processes, mainly tree-level and loop processes, allow for calculating the constraints on the value of the γ angle in the unitary triangle.



(a) **(b)** Target (Y)

Figure 1.3.2: Constraints on the Unitarity Triangle from (a) tree and (b) loop processes, from 2009.

As can be seen in the 1.3.2, the constraints on the tree-level before the LHC data taking periods? Run 1 and Rub 2 studies were much broader than for the loop processes, which established the need for research in this direction. The angle calculated by the LHCb for the tree-level decays as of 2020 is $\gamma = (67 \pm 4)^\circ$? .



(a) A decay of B^+ into kaon and two leptons. **(b)** Experimental results of the R_k ration calculation.

Figure 1.4.1

1.4 Test for the lepton universality.

The coupling of the leptons to the weak gauge bosons is expected to be identical. This is one of the consequences of the Standard Model. This means that there should be similar decay ratios except for the different masses of the leptons.

The Lhcb collaboration tested this assumption using a $B^+ \to K^+\ell\ell$ decay.

As can be seen in the 1.4.1 process includes a change of the meson B^+ into kaon by weak force and change of $\bar{b}$ int $\bar{s}$. The assumption coming from the Standard Model is that the ratios of electrons and muons produced in this process (leptons) should be close to 1. This ratio is called R_K and is calculated as $R_K = BR(B^+ \to K^+\mu^+\mu^-)/BR(B^+ \to e^+e^-)$

The result of the studies at LHCb is $0.846^{+0.044}_{-0.041}$**?** which is 3.1 standard deviations away from the Standard Model prediction, which hints at the breaking of the lepton universality. If this result were confirmed with greater confidence, it would suggest new physics beyond the standard model.

[..] to unite people from all over the world to push the frontiers of science and technology for the benefit of all.

Cern Mission Statement

2

The Large Hadron Collider beauty experiment

One of the most famous equations of the modern science: $E = mc^2$ states that the energy equals mass (multiplied by the speed of light squared). The true meaning behind that small equation is immense. All matter can be changed into energy, and energy can be transformed into matter. In studies of the elementary particles, this rule is, well, elementary. If we want all of the different particles of the universe, we can make them, "just" by using immense energy. This energy can take a form of a simple particle (like a single proton), being sped up to great velocities. If we speed up the same particle but in the opposite direction and subjugate them to collide with each other, the total accessible energy is even greater. Those collisions produce a vast number of possible particles. Of course, the more energy there is in a system, the higher the probabilities of creating different states of matter. This is the reason for the existence of the experimental High Energy Physics as a branch of science. The Large Hadron Collider was created as a result of a race of reaching ever higher energies. Furthermore, the Large Hadron Collider beauty**?** (LHCb) experiment is one of the main experiments that benefit from LHC's particle beam.

2.1 Large Hadron Collider

The LHC (Large Hadron Collider) is probably the biggest machine ever built by humans. It lies in a circular tunnel beneath the border of France and Switzerland (Geneva). It is a circular particle accelerator, 27 km in circumference. The tunnel depth ranges from 60 to 100 meters.

The LHC was created in the same tunnel, which housed a Large Electron-Positron Collider. The accelerator contains two beams of particles, the particle used for the beams are usually protons, but heavy ions (i.e. lead ions) are used for some periods of time. Its actual work was divided into years of runs or data taking periods. Runs 1 and 2 were undergoing in 2009-2013 and 2015-2018. The planned run three is supposed to start in 2022. There are four main detectors positioned along LHC; Alice, Atlas,

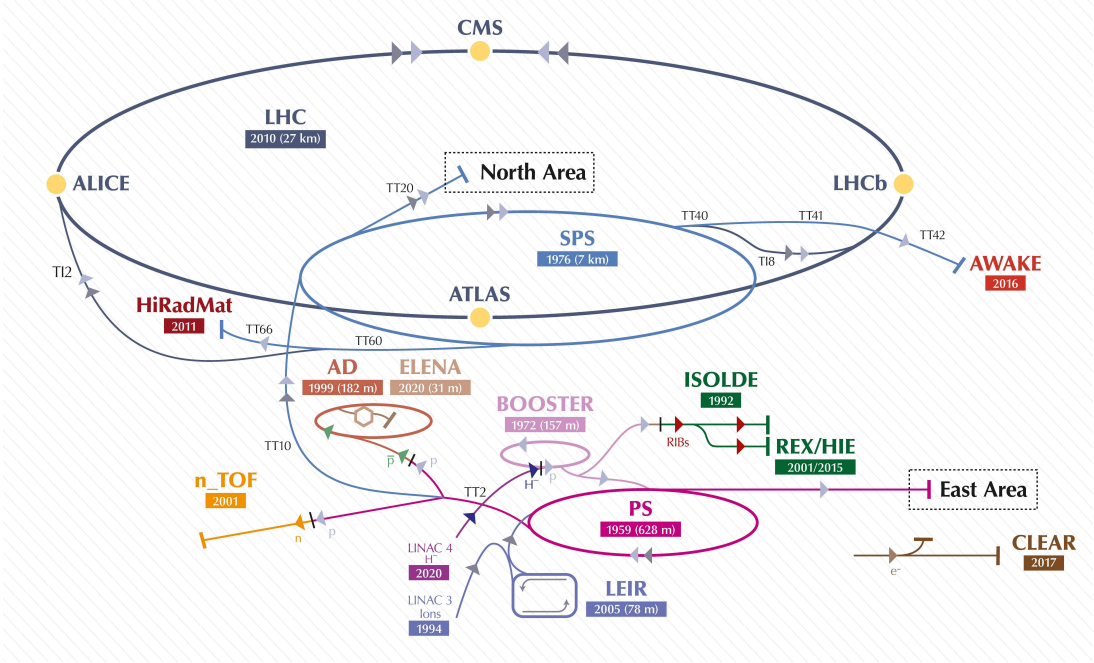


Figure 2.1.1: The entirety of the CERN accelerator complex. (source:?)

CMS, and LHCb. The entire accelerator complex is depicted in the Fig. 2.1.1. The maximal energy of the particles is dependent on the capability of the dipole magnets that are used to guide the particles on the circular path. As of Run 1 and 2 of the LHC, the dipole magnets could generate 8.33 T? for the 7 TeV upper limit energy level of particles. The maximum energy reached in LHC for a single proton was 6.5TeV. The other set of quadruple magnets is used to create a lensing effect on the particles, which allows to focus the beam correctly. The beam itself is comprised of bunches of particles. In order to create collisions, these bunches cross at an approximately 200μrad angle at LHCb? , depending on the energy and conditions. As seen in the 2.1.1, the LHC is part of the CERN accelerator complex. Before the particles make it to the final LHC ring, they are accelerated in stages in pre-accelerators. For the protons, the first system is a linear particle accelerator LINAC 4, then the Proton Synchrotron Booster (BOOSTER), next the Proton Synchrotron (PS), and finally the last pre-accelerator which is the Super Proton Synchrotron (SPS), which then directly feed the LHC ring.

2.2 LHCb spectrometer

The LHCb? experiment is the main focus of this thesis. It is a single-arm forward spectrometer. Unlike Atlas or CMS, it is not a general-purpose detector although it is sometimes called a general-purpose forward experiment. The initial design focused only on the beauty physics sector, which means studies of the hadrons containing the b-quark. Hence, the "beauty" part of the name stands for b quark. In time, its excellent tracking system and flexible high-level trigger allowed to expand significantly the initial physics programme. First, the studies of charmed particles were included, next a number of spectromety and exotic states searches were added (54 new particles, including penta- and tetra-quarks states, were discovered by LHCb), finally ion and fixed target measurements were performed. The unprecedented precision of obtained results made the LHCb also one of the best place to search for New Physics phe-

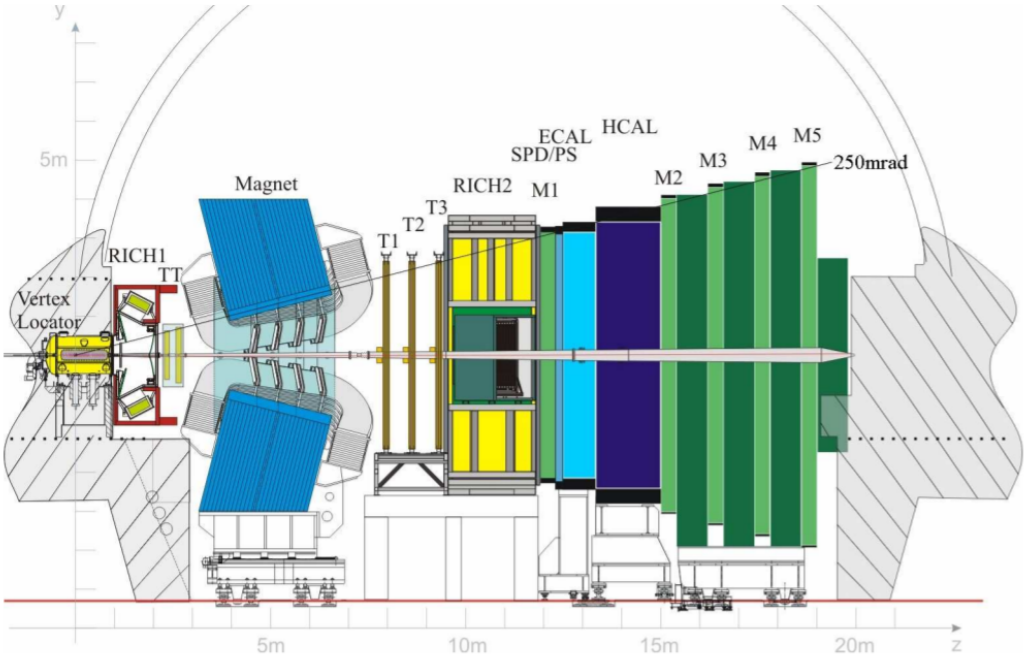


Figure 2.2.1: A cross-section of the LHCb spectrometer in runs 1 and 2, the Velo detector, is visible in the far left of the image.

nomena. The LHCb detector is located underground, in the LHC tunnel, near the french-swiss border on the French side.

The LHCb detector has been undergoing a deep upgrade**??**, and its composition for the future runs (3 and above) will be different than for runs 1 and 2. As the scope of this work touches both versions of the detector, the text will mark which detector it refers to, either by date, run or using the word "upgrade" to denote the new version of the detector. Overall, the detector's pseudo-rapidity range is $2 < \eta < 5$, and it's momentum resolution: $\Delta p/p = 0.5\%$at low momentum to 1.0% at $200 GeV/c$**?** which is the best at LHC.

2.3 LHCb subsystems

This section will briefly discuss LHCb subsystems, apart from the Velo detector, which is discussed in detail in the next section. The order of the presentation of the subsystems follows their position along the beam axis (see Fig. 2.2.1), that is also the z-axis of the LHCb global coordinate system.

2.3.1 Rich

Ring Imaging Cherenkov detectors (RICH-1 and RICH-2) uses the Cherenkov effect to identify charged hadrons. RICH-1 is closer to the interaction point and detects lower momentum particles ($1-60 GeV/c$). RICH-2 is downstream after the magnet and is optimised to detect hadrons with momenta in the range ($15 - 100 GeV/c$) Both detectors are using different radiation mediums for Cherenkov effect. RICH-1 is using aerogel and C_4F_{10}, RICH-2 is using CF_4. The different media and different Cherenkov angles produced by the particles, with a combination of the momentum information of particle, allow to dis-

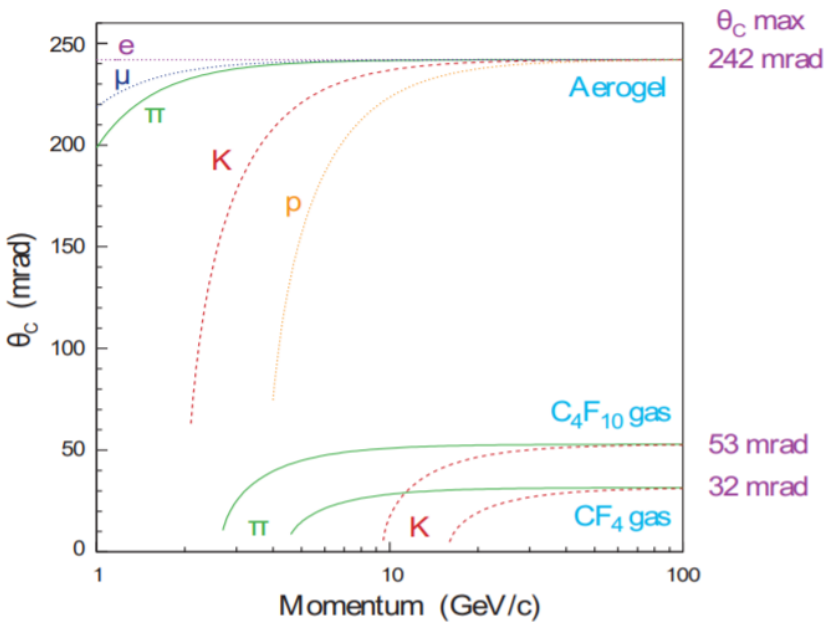


Figure 2.3.1: Different media used in RICH with different Cherenkov angles versus their momentum.

tinguish protons, kaons and pions with high precision.

2.3.2 Trackers

Apart from the Velo, the tracking subsystem can be divided into three parts: Trigger Tracker (TT), Inner Tracker (IT) and Outer Tracker (OT)**??**. The IT and TT make up a Silicon Tracker (ST), which utilises the silicon microstrip technology, similar to Velo. The TT is located upstream and is a 150 cm wide and 130 cm high planar detector between the magnet and the RICH-1 (see Fig. 2.3.2b), and the IT station covers a 120 cm wide and 40 high cross-shaped region in the centre of the OT. The ST was designed with a single hit resolution of 50 μm Each of the trackers has four layers which utilise "xuvx" topology (two inner layers are rotated by a stereo angle of $+/-5^o$, as presented in Fig. 2.3.2a). The OT is a detector comprising of gas-tight straw-tube modules, with the 4.9 mm inner diameter of the straws. The inner gas is a mixture of Argon (70%) and CO_2 (30%). The gas composition was chosen to guarantee fast drift time (less than 50 ns) and good drift-coordinate resolution (200 μm). It comprises three stations, marked as T1, T2, and T3 in Fig. 2.2.1. Each of the stations consists of 4 layers employing a similar topology $x-u-v-x$ of the straws, with $+/-5^o$ stereo angle. All of the stations consist of two retractable halves, but unlike the Velo, they are only retracted when servicing. These halves consist of short modules (S-type) and long modules (F-type). The F-type modules contain a staggered layout of 256 straws, and the S-type modules containing 128 single straws. The S-type modules are located closest to the beam and are half of the length of the F-type to compensate for the space for the beam. In total, there are 168 F-type and 96 S-type modules,

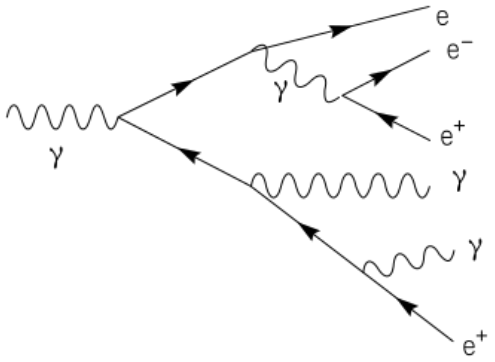
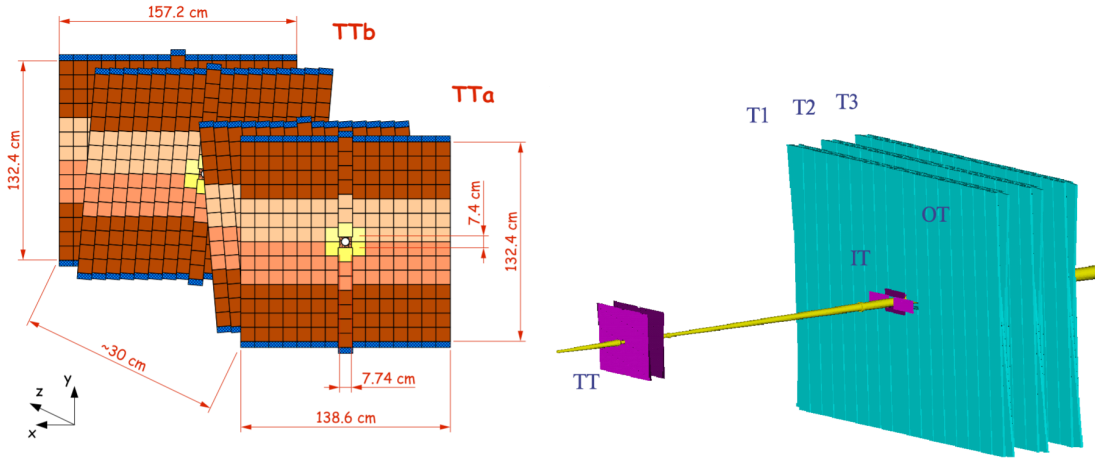


Figure 2.3.3: Electromagnetic particle shower*.



(a) TT stations positioning[?] **(b)** Silicon tracker and outer tracker[?]

2.3.3 Magnet

There is a massive warm, dipole magnet of 1600 tonnes of weight between the tracking stations. Its purpose is to bend the charged particles' trajectory to determine their charge and momentum. High performance LHCb tracking system requires that the field generated by the magnet has to be carefully mapped with the precision better than 0.1 ‰.

2.3.4 Calorimeters

The calorimeter system at LHCb consists of three components: PS/SPD (preshower detector/scintillator pad detector), ECAL (electromagnetic calorimeter) and HCAL (hadronic calorimeter). There are two common types of showers in the calorimetry world: electromagnetic showers and hadronic showers. When the electron or positron enters scintillating active material, it emits a photon. A photon, in turn, dissociates to an electron-positron pair, which creates a positive feedback loop that creates more and more particles. This is known as an electromagnetic particle shower and is depicted in Fig 2.3.3.

The hadronic particle shower is more complicated and involves interaction via the strong force. The electromagnetic calorimeter was created for the purpose of capturing electromagnetic showers. The

*Image source: https://commons.wikimedia.org/wiki/File:Schematic_of_a_particle_shower.svg

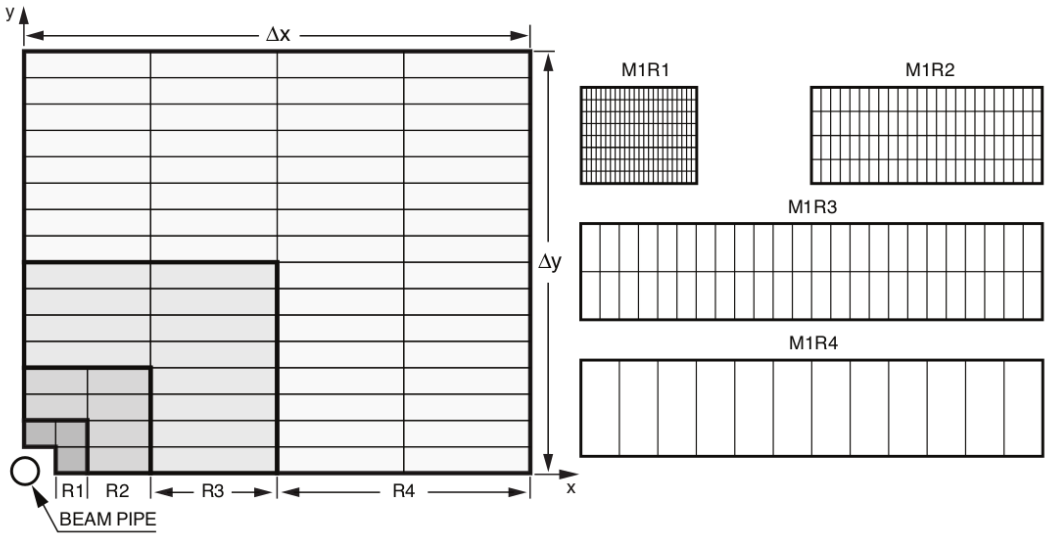


Figure 2.3.4: Left: front view of a quater part of the muon station, each part reperesents a single region (R1, R2, R3, R4). Right: Division of the each of four region chambers into logical pads.

rejection of the high-energy charged pion showers in the ECAL requires a PS detector placed in front of the ECAL. Furthermore, neutral pion showers also must be rejected. For this reason, the SPD detector was positioned in front of the PS detector, separated by a lead converter.

The electromagnetic shower is induced more easily and is shorter than the hadronic one. The HCAL is much denser and bigger to allow for complete deposition of the energy of the hadronic shower.

2.3.5 Muon station

The muons are about 200 times heavier particles than the electrons and exhibit much less energy deposition in matter. In a typical case muon detectors are positioned at the outermost part of the setup (this is the case for LHCb spectrometer). Muons, thanks to their properties constitute a very convenient objects to trigger on.

In LHCb, the muons system consists of five stations (M1-M5). It is a gas detector consisting of a total of 1380 chambers. The stations M2-M5 are positioned downstream, and between each station, there is an iron absorber 80 cm thick. All of the stations utilise MWPC (multi-wire proportional chambers) technology, except for the inner region of the M1 station, which uses triple-GEM (gas electron multiplier) detector chambers. All stations are divided into four regions, R1, R2, R3, and R4, corresponding with the 1:2:4:8 ratio of length from the beam (as seen at 2.3.4). The inner space of MWPC is filled with a mixture of $Ar/CO_2/CF_4$ $(40:55:5)$, and contains a wire plane of 2 mm spacing, symmetrically placed in a 5mm gap. This allows for 5ns time resolution. Each chamber consists of four gaps with wires.

2.4 Velo

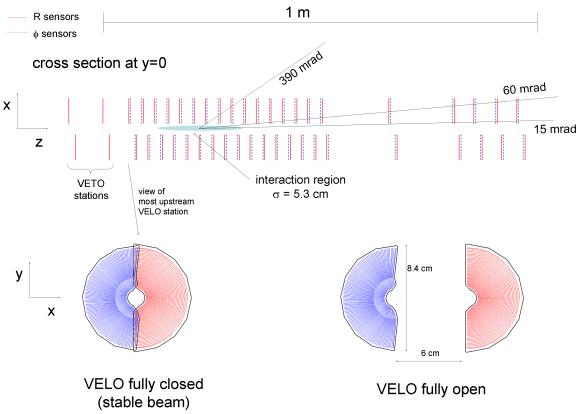
The Velo detector (Vertex Locator) is the heart of the LHCb spectrometer. Its main goal is to record the charged particles' trajectory and reconstruct the collision point with extreme precision. As mentioned in the previous section, the composition of the LHCb spectrometer is changed in upcoming Run 3. The

Velo detector is (as of the time of writing) being upgraded from a silicon micro-strip detector to a silicon pixel matrix detector. Because of the possible confusion I assumed the following naming convention throughout my Thesis. I will refer to the upgraded detector as "VeloPix", and to the detector used in Run 1 and Run 2 as "Velo Strip" or simply "Velo". The readout chip is called VeloPix ASIC or VeloPix chip.

2.4.1 Strip Velo in Run 1 and 2

The Velo in the Run 1 and 2 consisted of 42 modules. Each of the modules comprised 2 sensors: R and φ-type. Each of the sensor sides was a silicone microstrip sensor. The individual sides correspond to different polar coordinates, as depicted at 2.4.2. Each sensor had 2048 silicon strips (physical channels). In total, it gives more than 170 000 strips in the entire Velo detector.

(a) Strip Velo during assembly*.



(b) Overview diagram showing the spacing of modules along Z and positions open and closed.

Velo modules are mounted on two retractable halves, which enclose the beam. The movable construction is necessary to protect the LHC beam during the injection stage, before the stable beam condition is achieved. One of the halves is visible in the Fig. 2.4.1a. The alignment of the modules on the halves is staggered as depicted in Fig. 2.4.1b. Additionally to 42 two-sided modules, four VETO stations are positioned on the other side of the interaction point, consisting of only R-type sensors. The VETO stations proved very useful to provide information to the trigger system and measure the rapidity gap.

The Velo detector is enclosed in its own secondary vacuum tank, which is separated from the primary beam vacuum. The geometry of the inner gap of sensors is 8mm in radius, and the outer radius of the sensor is about 42 mm. When the stable proton beam is decalred, the halves of the detector are being retracted (so called closing procedure), which requires the precise position of the collision point to be calculated. After that, the halves of the Velo detector are moved closer gradually towards the beams using stepper motors. Sensors get as close as 7 mm to the interacting beams.**?** . The sensors themselves are created using oxygenated n^+-on-n technology, consisting of n^+-type implant on a n-type bulk with a back-plane of p^+-type implant. With an exception for two most upstream modules that where produced using the n^+-on-p silicon type.

*Image source: https://lbtwiki.cern.ch/bin/view/VELO/VELOConferencePlots

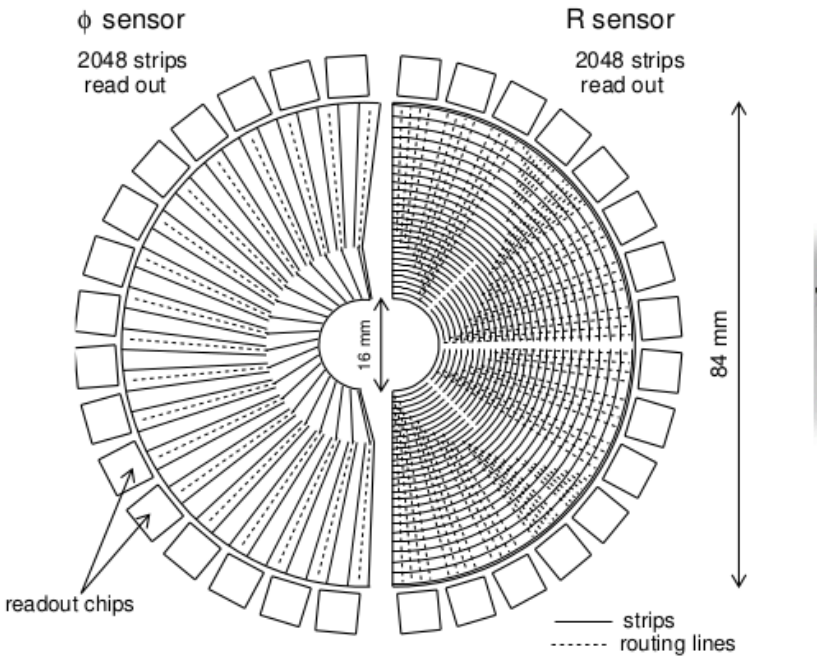


Figure 2.4.2: A depiction of Velo sensors and their routing lines. This is just the schematic representation as it does not reflect the actual number of strips and routing lines. The Φ-type sensors' outermost strips are directly connected to the readout chips without a need for long routing lines, but the innermost part has routing lanes lying parallel to the outermost strips. The *R*-type sensor routing is divided into sections of routes with different lengths. In reality, both sensors are divided into 512 long sections of repeated routing lines lengths.

The sensors were 300 μm thick, and their spatial separation (pitch) varied as a function of radius**?** as depicted on the 2.4.3. The way that physical sensor strips were arranged, with their routing lines, influences the ordering of the data and the calibration procedure.

As seen in the Fig. 2.4.2, the ordering of the R-type sensor routing lines is divided into four sectors and varies in the distance of the strip from the centre (proton beam). What is not visible in the Fig. 2.4.2 (because of the simplification of the plot) is that the Φ-type sensor is also divided into two parts (as seen in the 2.4.3).

The silicon strips were connected to the Beetle front-end ASICs that were mounted on the Velo module board. Each Beetle chip had 128 indivudually instrumented readout channels. In the case of Velo the signals read from the detector were only conditioned by the analogue part of the chip and send via 60 m long copper cables for digital processing to the electronics Tell1 boards. The strips were sampled with 40 MHz frequency and internally queued**?** in an analogue pipeline. The analogue data were brought off-chip at 1.1 MHz, with 32 channels multiplexed over four lines.

As mentioned above, the data readout from the Beetle chips was passed to Tell1 boards. The Tell1 boards have been standardised for the entirety of the LHCb detector. They accept either optical or analogue receiving cards. The information was then processed on the on-board FPGAs with custom processing algorithms. From the Tell1 electronics, the data was transferred to higher-level systems like ECS (Experimental Control System) and triggers.

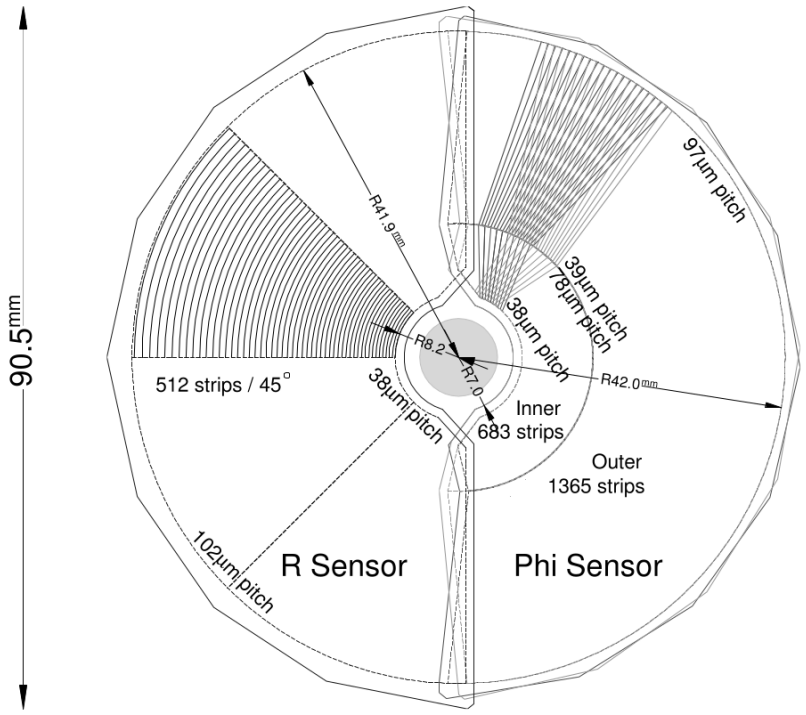


Figure 2.4.3: A detailed view of the Velo sensors with their according pitch. The R sensor's pitch starts at 38μm and ends on 102μm. The Φ-type sensors' pitch modulation is divided in two. The most inner 683 strips have from 38 μm to 78 μm pitch, and the outer rest has a pitch ranging from 39 μm to 97 μm.

2.4.1.1 Calibration procedure for Velo

In the absence of proton-proton collisions, the signal observed on each Velo readout channel is approximately normally distributed. In the first place, the calibration procedure aims to shift the central value of each of these distributions to zero (so called base-line correction). As shown in the top plot in Fig. 2.4.4 this offset (also called the pedestal) is, in general, different for each individual channel and needs to be evaluated by a dedicated algorithm using RAW Non-Zero Suppressed Velo data taken without colliding beams. The equalised Non-Zero Suppressed data are shown at the bottom of Fig. 2.4.4. After the offset correction is applied (called the pedestal P subtraction, **?**) the noise can be estimated in each channel as the width of the signal distribution.

The noise evaluated during the calibration procedure is vital for the hit detection and cluster reconstruction algorithm since it is used to suppress the channels without high ADC counts (Zero Suppression procedure). Here, we assume the correlation between charged particles passing through sensors and depositing the energy in their active volume and the signal amplitude measured on each readout strip. The performance of the cluster reconstruction algorithm is of critical importance for the Velo data quality. The algorithm is comprised of two stages hit detection and cluster building. The former step relied on so-called high thresholds evaluated for each channel using the measured noise. For each channel, the high threshold, H_t, was set to be five times higher than the noise measured on this channel. Taking into account the approximate Gaussian nature of the signal distribution without particle hits, it

*Where ADC is Analogue to Digital Count that represents the amplitude of the signal after the digitisation.

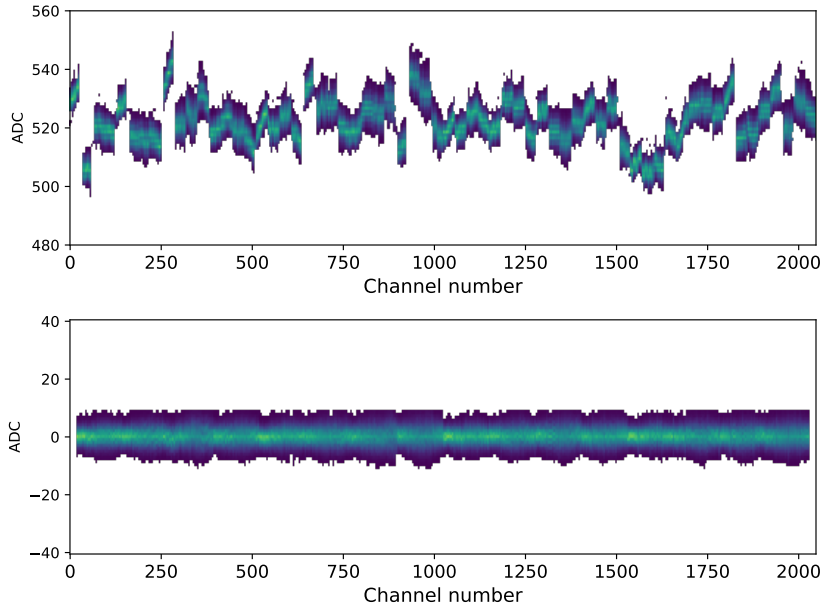


Figure 2.4.4: Typical RAW Non-Zero Suppressed Velo data before (top) and after the digital processing (bottom). The response of each channel was equalised, and a common flat baseline centred on 0 ADC*achieved. The spread of the data points about the baseline is attributed to the total noise present in the system.

corresponds to the probability of accepting a noise signal as a real hit at the level of 10^{-7}. The cluster building step used the selected channels that passed the hit detection algorithm as seeds. Clusters were formed by searching for the secondary signal channels using so-called, low hit thresholds. Up to four channels were used to form a single cluster.

In general, the clustering algorithm works as follows**?** :

1. If the measured signal exceeded the high threshold H_t on strip S_n, then the channel was tagged as the cluster seed.

2. In the next step, the signal measured on the adjacent strips, with respect to a seeding strip, were compared to the low thresholds. All strips that passed the low threshold cuts were tagged, along with the seeding strip, as a cluster candidate

3. In the last step, a cluster (hit) was built. A global cut on the cluster size was set to four strips. In case when a cluster candidate is comprised of more strips, the algorithm creates multiple adjacent clusters. Finally, the local position of each created cluster was evaluated using a centre of gravity formula.

A visualisation of the algorithm can be found in the 2.4.5.

2.4.1.2 Header Cross-Talk

The Beetle chips in Velo are connected to 128 physical strips and sends data on four analogue links, with 32 channels of analogue data. There is a pseudo-binary information header preceding the analogue data. This information consists of four bits and encodes the status of the front-end chips and the pipeline

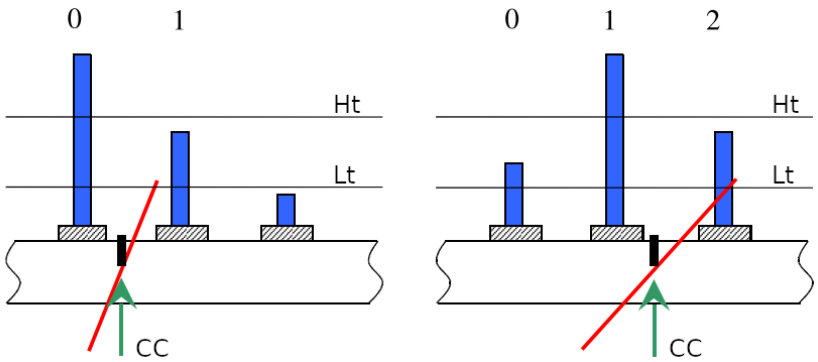


Figure 2.4.5: Visualisation of the clustering algorithm. Two examples of Velo clusters are shown schematically. Blue rectangles depict the charges collected on the strips. Black lines represent high threshold H_t and low threshold L_t cuts, the reconstructed cluster centre (CC) is indicated by arrows and the particle trajectory is given by the red sloping lines.

column number. Unfortunately, this caused a problem in the data transfer, and the four preceding bits induced an additional noise in the first upcoming data channel. In total, this contributed to the increased noise in the following channels in Velo sensor: 0, 32, 64, 96, 159, 191, 223, 255, 287, 319, 351, 383, 415, 447, 479, 511, 543, 575, 607, 639, 640, 672, 704, 736, 768, 800, 832, 864, 896, 928, 960, 992, 1024, 1056, 1088, 1120, 1183, 1215, 1247, 1279, 1311, 1343, 1375, 1407, 1439, 1471, 1503, 1535, 1567, 1599, 1631, 1663, 1664?, 1696, 1728, 1760, 1792, 1824, 1856, 1888, 1920, 1952, 1984, 2016. This effect is known as header cross-talk**?** . The initial commissioning studies, performed before Run 1, showed that this can produce a significant stream of fake clusters increasing the volume of the Velo data transmitted to the high-level trigger. A dedicated correction algorithm has been developed in order to alliviate the problem.

2.4.1.3 Voltage and luminosity

Each of the strips in Velo is, in essence, a semiconductor diode. It is an n^+-type implant in an n-type bulk with a back p-type implant**?** . The physical process of exciting the matter electromagnetically inside the sensor bulk creates a cloud of electrons and holes. For the charge to be able to cross the depletion region, there needs to be additional bias voltage applied. This is also a way of mitigating the radiation damage, as the multiple negative factors disrupt the flow of the generated charge. The Fig. 2.4.6, shows that the bias voltage necessary for the optimal operation of the Velo detector has been steadily growing along with the delivered integrated luminosity.

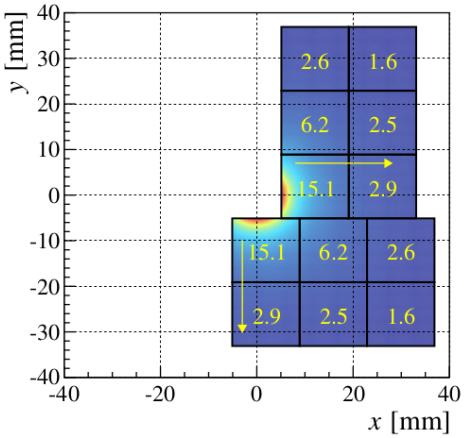
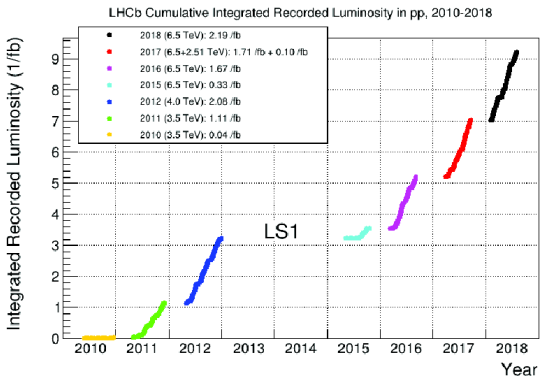
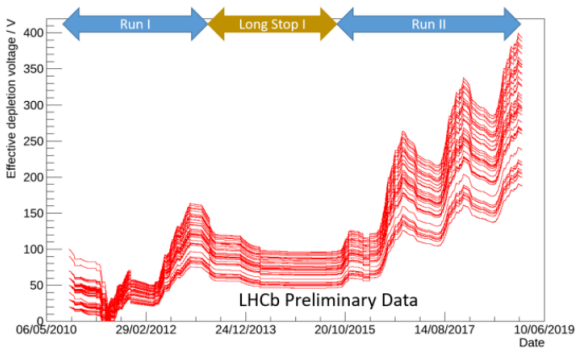


Figure 2.4.7: The projected data rate for each ASIC in Velo module (in Gbit/s).



(a) Integrated luminosity for the years 2010-2018[?] .



(b) Projected Velo bias voltage.

Figure 2.4.6: Radiation damage effects on the LHCb Velo.

2.4.2 VELOPIX

The Large Hadron Collider and its detectors were planned to be incrementally improved over time. In LHC Run 3, there will be a significant increase in the instantaneous luminosity, and the Velo must adapt to that change. For this purpose the Velo group proposed to change the sensor technology from planar strip sensors to pixel ones read-out by the VeloPix ASIC that was adapted from the TimePix chip.

Some of the requirements for the upgraded detector include:

- **Increased rate of data** output to 40 MHz with hits/event $5.2cm^{-2} \times R^{1.9}$ (R is radius in cm). This means that the VeloPix ASICs must be able to output up to 15.1 GB/s. The peak total data rate of upgraded Velo can reach up to 2.85 Tbit/s.

- **Irradiation** the upgraded detector is predicted to experience an accumulated the flux of fast hadrons $8 \times 10^{15} n_{eq}/cm^2$, and when this is reached, it is expected that the bias voltage and the current necessary for operating the sensors at the pixel closest to the beam will be $1000V$ and $200\mu A/cm^2$ at -20C. Such a high dose of radiation and high currents will make a significant obstacle in protecting the sensor from thermal runaway.

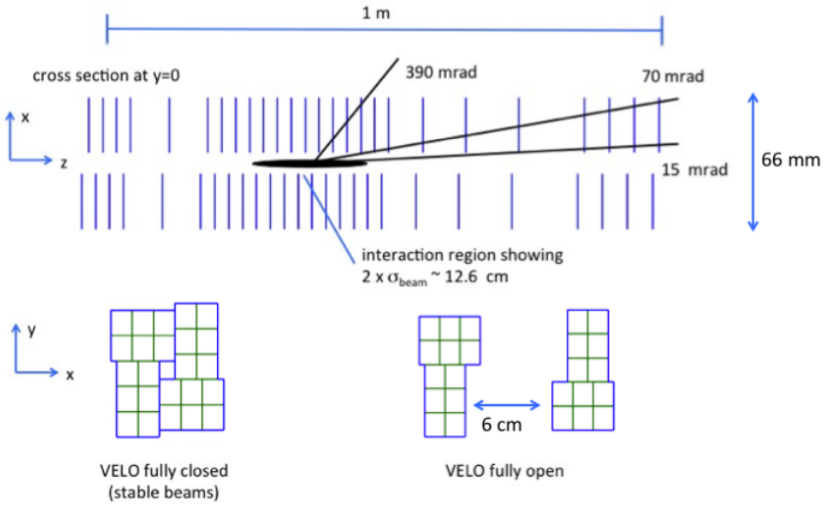


Figure 2.4.8

The new upgraded Velo detector that will collect data during Run 3 and Run 4 has a modular design, similar to the previous strip device. Each module contains 4 sensors, each sensor is readout by 3 VeloPix ASICs. VeloPix chips are rectangular in shape and can readout matrices of 255x255 pixels. New Velo sensors feature square $55x55\mu m^2$ pixels that provides excellent spatial resolution. There are 52 modules arranged along the z-axis (i.e., along the LHC beam). It is worth noticing that significant portion of the Velo strip detector's infrastructure remained, for instance the secondary vacuum tank. A simplified visualisation of the Velo modules z-positions and the pixel sensors geometry is shown in Fig. 2.4.8.

In order to process the data more effectively and save the transmission bandwidth individual pixels are grouped on the logical level in, so called, super-pixels. Super-pixels consist of eight pixels arranged as 2 by 4 matrices. Sending data using the super-pixels instead of pixels allows for a reduction of the total required bandwidth by about 30%**?** . When there is a hit in the super-pixel, a 9-bit timestamp is created and stored. Each super-pixel can store up to two timestamps. Readout from the super-pixels is done via a 23-bit bus running at 13.3 MHz. This means that there is a data transfer only when there is a hit information. In other words, the new VeloPix detector is a trigger-less system.

2.4.2.1 Velopix masking and calibration

A critical functionality of both vertex systems is a possibility to deactivate a problematic readout channels, also called "bad" channels. Typical reasons for deactivation can be related with sensor issues (for instance physical damage), problems with bonds (broken bond or partially connected bond) or readout lines. The best strategy to identify bad channels is to measure and trend noise. The procedure of collecting the noise data (without proton-proton collisions) and identifying problematic channels is called masking. During the Run 1 and Run 2 operation a significant experience was gained related to this important issue. There was a fraction of bad channels that were considered "fixed" and they do not changed over time, however, a majority of them did not show permanent problems and evolved in time. Quick tagging of bad channels was essential for keeping the high data quality. For instance, a large number of noisy channels will create a stress in trigger readout system by producing spurious fake hits. This

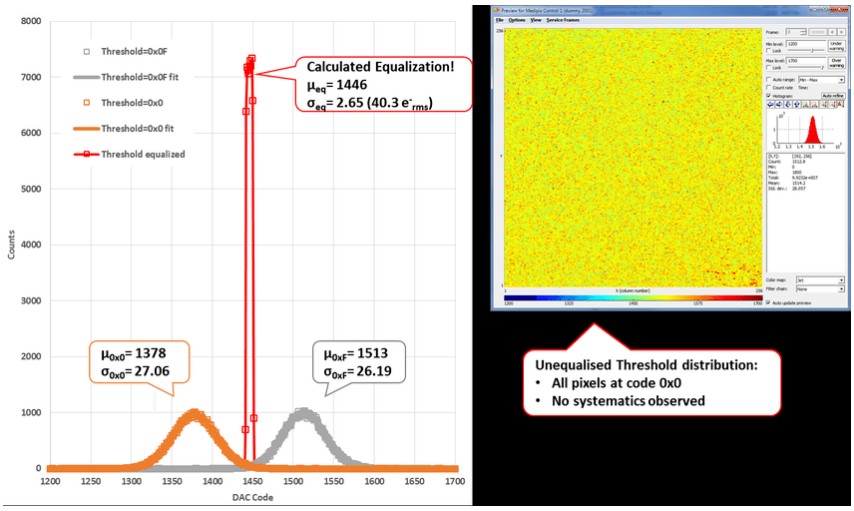


Figure 2.4.9: Exemplary threshold spread of the pixel matrix.

may degrade the tracking and increase processing time due to larger number of hits. In case of the pixel detector, that operates in trigger-less mode, large number of unmasked noisy channels can completely saturate the readout lines, since the fake hits will be sent constantly (in case of the strip detector it would be just one fake cluster per one trigger). Evaluation of the set of pixels to be masked will be performed during the, so called, equalisation that is the core of the VeloPix calibration procedure. The number and distribution of the masked channels must be carefully study and monitor during the data taking period. A short description of the equalisation is given below.

Similarly to strip Velo, VeloPix will collect the noise calibration data**?** . In the case of the VeloPix, each pixel has its own individual voltage threshold. If this threshold is exceeded by signal induced by a passing charged particle, we register a hit. There are two types of values that can be set in a readout chip to define individual thresholds: Global Threshold (GT) and Trim. The former is evaluated for each sensor matrix (255 x 255 pixels), whilst, the latter are defined per individual pixel, they are 4-bit numbers (16 different values). Trim is a readout DAC voltage that is individually added to the signal in the discriminator such that it allows to achieve equal response for each pixel across the whole detector matrix. For the convenience of the Reader, the automatic procedure that is used to evaluate the GT and Trims is described briefly below. First, we set all pixels to use their $Trim_0$ value, and we scan the reaction of the pixels (thresholds) by moderating GT value and, each time opening the shutter for a constant amount of time T_C. Then we repeat this operation for $Trim_{15}$. In both trim cases, we record the total number of active pixels (pixels that registered noise) $N_{HITS}(GT)$ 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 noise events $H_{P_{x,y}}$ 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 as to calculate $TrimValue_{15_{x,y}}$ in $Trim_{15}$. That means that a scan in $Trim_0$ and $Trim_{15}$ is used to set the range of voltage for the trims. The $TrimValue_{0_{x,y}}$ is used as the lower bound for all of the 16 trims in each individual pixel and the $TrimValue_{15_{x,y}}$ is used as the maximal bound for the trims. Therefore one step of the trim is $TrimStep_{x,y} = TrimValue_{15_{x,y}} - TrimValue_{0_{x,y}}/15$.

That process allows us to calculate and calibrate trim values for each pixel. Now to determine which of the trim value should be set for the pixel, we take advantage of the $N_{HITS}(GT)$ measured in each of the

two scans ($Trim_0$ and $Trim_{15}$). Both of those yield noise events distributed normally. We calculate mean of the distribution in TRIM0 M_{TRIM0}, and TRIM15 M_{TRIM15}. The final value of the trims is calculated as $GT_{target} = M_{TRIM0} + M_{TRIM15}/2$. Then each of the pixels gain the trim value of $PixelTrim_{x,y_a}$ set as close to GT_{target}. Then 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 $Th_{x,y} = PixelTrim_{x,y_a} + GT_{target} + 1000electrons$ This procedure is visualised in Fig. 2.4.9. is

What emerged from the darkness of the caves, from the depths of millennia, we wish to pass on to machines, in one brief circuit closure, to spark reason into existence.

Stanisław Lem

3 Machine Learning

Although machine learning experienced rapid growth in the second decade of the new Millenium, it has come a long and difficult path to become almost omnipresent in everyone's life and in many scientific disciplines. The high energy physics is not an exception, and in fact, the majority of this thesis describes various approaches to application of computational intelligence in field of particle physics. This chapter contains an introduction to the machine learning methods used in further chapters.

One of the events that can be thought of as the birth of machine learning is the invention of the perceptron. The perceptron was the protoplast of the artificial neural networks we know today. It was created by Frank Rosenblatt**?** in 1958. In essence, the perceptron is what we today can call a single layer neural network. Famously, the over-promising of the capabilities of the perceptron was one of the leading causes for the so-called "first AI winter"**?** . It was quickly discovered that the perceptron cannot be efficiently scaled to solve more generic problems and is, in particular, unable to solve the XOR task. Although the concept of multi-layer perceptron was already proposed, it lacked a viable way of training the parameters. This was later mitigated by the use of back-propagation as a method for training the parameters of the artificial neural networks. Back-propagation was popularised in the 1980s. This also coincides with a time of significant growth in interest in artificial intelligence. The primary trend was the usage of so-called "expert systems", which in reality were a set of predefined if-then statements built around expert knowledge. Again, the hype for the AI and under-delivered solutions led to a cut in funding in research in this area and the "second AI winter", which lasted to about the second half of the first decade of the new Millenium. The significant development in computing power and the increased ability to create vast training data-sets allowed for the dynamic growth of this field of computer science that we observe to this day.

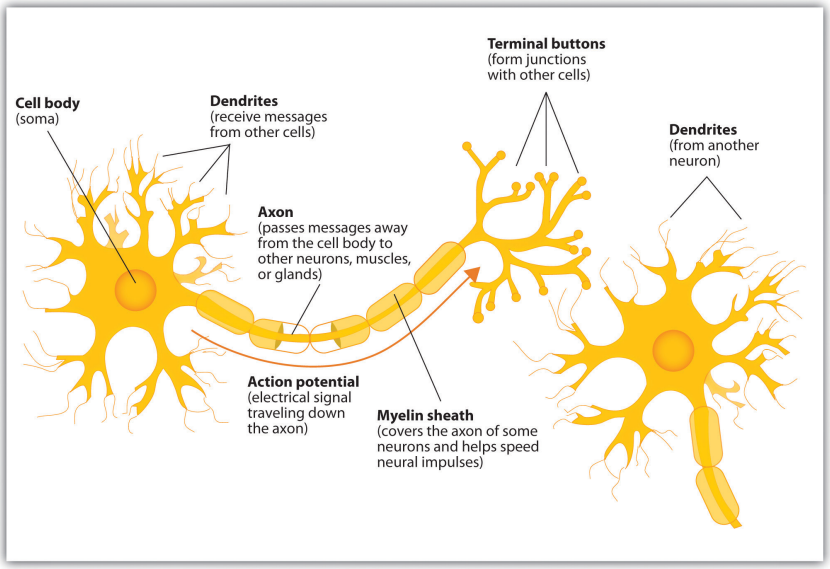


Figure 3.2.1: Components of the neuron*.

3.1 Areas of machine learning

Machine learning can be divided into three general categories:

- **Supervised learning** expects pairs of desired input and output (X, Y). Supervised learning models assume that there exists a function f where $y = f(x)$, and through its workings, tries to approximate the f function. This is by far most used type of machine learning algorithms, as given correct set of data, it is relatively easy to control the algorithms. Problems that can provide input-output pairs of data also often allow assessing a model's performance with a single, easy-to-understand metric.
- **Unsupervised learning** doesn't expect an exact output paired with given output. It does not expect output at all. Generally, the tasks that are solved by unsupervised learning focus on discovering structures inside the data, like clusters and groups. There is a rising area of usage for the generation of the data points using the unknown inner distribution of the data.
- **Reinforcement learning** usually focuses on the action taken in a given environment. Instead of the exact output matching the input, there is a defined reward that the reinforcement learning system is trying to maximise.

3.2 Neural Networks

Although the (artificial) neural networks are hardly the only machine learning technique available, they are the current go-to solutions for many machine learning problems. This section will introduce the reader to the basics of modern neural network techniques, as they play a crucial role in the research discussed in the further chapters.

*Source: https://en.wikipedia.org/wiki/File:Components_of_neuron.jpg

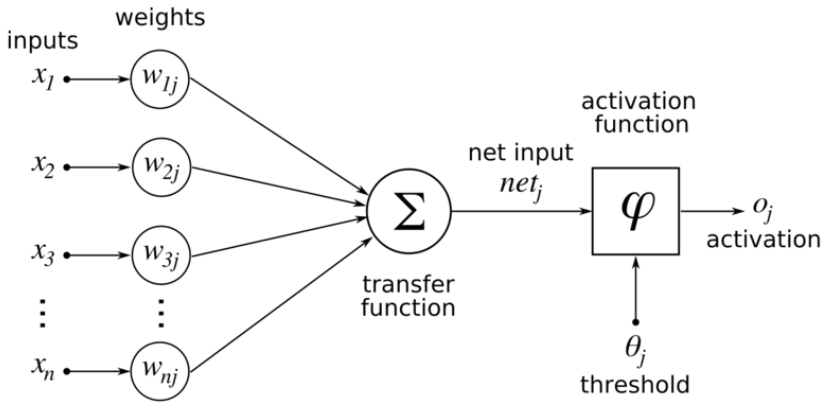


Figure 3.2.2: A visual, graph-based representation of a single artificial neural network neuron*.

3.2.1 Multi-Layer Perceptron

The artificial neural networks are based on a simplified model of the actual biological neural (nervous) tissue. A single neuron contains many dendrites and an axon, as visible in Fig. 3.2.1. The electrochemical potential supplied to the dendrites is accumulated, and then if a certain threshold is exceeded, the axon fires an electrochemical signal. While it is true that this model of a neuron is a large oversimplification, the idea can be boiled down to the following mathematical formula:

$$f(x) = \begin{cases} 1, & \text{if } \sum_i x_i \geq T \\ 0, & \text{otherwise} \end{cases}$$

The function above responds with 1 when the sum of the input signals x_i is greater than threshold T. This was extended by Rosenblat by addition of weights to the input and activation function. The threshold can be represented as additional non-weighed input b.

$$f(x) = \begin{cases} 1, & \text{if } activation(\sum_i w_i x_i + b) \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

For binary classification, it may be useful to get the answer represented in the range between zero and one, which actually may be applied by using the activation function only in 0..1 domain, but other uses of neural networks may call for linear output (regression tasks).

$$f(x) = activation(\sum_i w_i x_i + b)$$

This can be represented graphically as a simple graph with inputs and outputs (Fig. 3.2.2). Individual artificial neurons may be grouped in layers of neurons, sourcing the same inputs, as depicted on 3.2.3.

*Source https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png

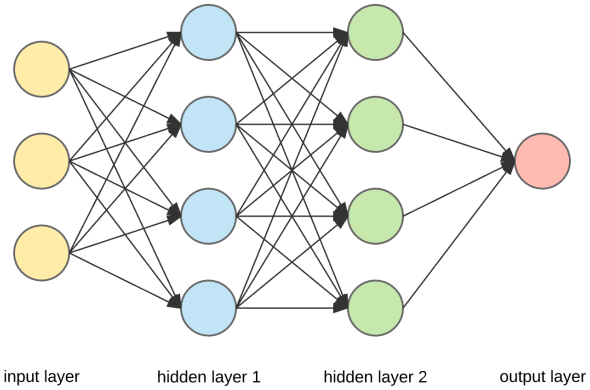


Figure 3.2.3: A graphical representation of multi-layer neural network*.

And this is where Multi-Layered Perceptron (MLP) got its name. The forward pass in the neural network is the input undergoing multiplication with each layer of the network sequentially, and in effect, it produces the output. The typical agreement in the ML community is that MLP is any simple neural network with arbitrary activation and with more than two layers. To much of the disappointment of many computer science students, the difference between "deep learning" and MLP is commonly accepted to be with just a number of layers, with deep learning starting after two layers.

3.2.2 Stochastic gradient descent and Backpropagation

The "learning" part of machine learning comes from the fact that the "machine" is given examples, pairs of input and desired output (x, y). The goal of the machine is to find such function $f(x)$ that $y = f(x)$. Realistically speaking, not all of the tasks might be representable by the machine, so we expect approximation $y \approx f(x)$. The error of this approximation can be expressed as $\varepsilon = y - f(x)$. Because a simple difference of two vectors may not be most suitable, we define a network's loss metric which is a function that given network output $f(x)$ and the target value y will yield an error value $\varepsilon = loss(f(x), y)$. Several possible loss metrics exist, which can be used in a number of cases. For classification, the popular loss metric is log-loss also known as cross-entropy, visible in Eq. 3.1 where p is a predicted probability. For regression tasks, an example of a loss metric is the mean squared error (MSE).

$$L_{\log}(y, p) = -(y \log(p) + (1 - y) \log(1 - p)) \tag{3.1}$$

The metrics can assess the quality of the output of the neural network. This gives the opportunity to measure the error of the network, and in turn, we can optimise its response with respect to its weights. The popular way of doing this is using a stochastic gradient descent which is a variation of a gradient descent algorithm. The gradient descent algorithm is a minimisation algorithm that finds the optimal solution by following the gradient of a function. In our particular case of machine learning we study the properties of the loss function. In each step the algorithm adjust the weights by taking into account

*Source
https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a

the gradient of the loss function calculated with respect to the model's weights. Let us denote $\theta = w_i$ and change the domain to time so that θ_t is the vector representing weights of the neural network at the time-step t. Then we can express gradient descent as follows:

$$J = loss(f(x), y) \tag{3.2}$$
$$\theta_{t+1} = \theta_t - a\nabla_{\theta_t} J \tag{3.3}$$

In the term $\sum_i w_i x_i + b$ we can "hide" the b variable by assuming additional element in input vector $x_{n+1} = 1$ and then $b = w_{n+1} x_{n+1}$. Also assuming MSE as the loss function, and using chain rule we can write that the derivative of loss in respect to network parameters $\nabla_{\theta_t} J$ in the following way:

$$\frac{\delta J}{\delta \theta_t} = \frac{\delta J}{\delta f}\frac{\delta f}{\delta \theta_t} \tag{3.4}$$
$$\frac{\delta J}{\delta f} = \frac{\delta \frac{1}{2} * (f(x) - y)^2}{\delta f} = (f - y) \tag{3.5}$$
$$\frac{\delta f}{\delta \theta} = \frac{\delta \theta_t x}{\delta theta_t} = x \tag{3.6}$$
$$\frac{\delta J}{\delta \theta_t} = (f(x) - y) * x \tag{3.7}$$

The same can be done for the network with an arbitrary number of layers. Given a two layered network, with arbitrary activation function σ:

$$a^{[1]} = w^{[1]} x + b^{[1]} \tag{3.8}$$
$$z = \sigma(a^{[1]}) \tag{3.9}$$
$$a^{[2]} = w^{[2]} z + b^{[2]} \tag{3.10}$$
$$J = \frac{1}{2}(a^{[2]} - y)^2 \tag{3.11}$$

We can calculate the gradient descent for two layers of the network:

$$\theta^{[1]}_{t+1} = \theta^{[1]}_t - a\nabla_{\theta^{[1]}_t} J \tag{3.12}$$
$$\theta^{[2]}_{t+1} = \theta^{[2]}_t - a\nabla_{\theta^{[2]}_t} J \tag{3.13}$$

Then we can write the gradients as: $\nabla_{\theta^2_t} J = \frac{\partial J}{\partial w^{[2]}}$ and $\nabla_{\theta^1_t} J = \frac{\partial J}{\partial w^{[1]}}$. In fact the chain rule is at it's core the backpropagation method. We follow the chain rule to calculate the gradient for the first layer θ^1_t.

$$\nabla_{\theta_t^1} J = \frac{\partial J}{\partial w^{[1]}} \tag{3.14}$$

$$\frac{\partial J}{\partial w^{[1]}} = \frac{\partial J}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial w^{[1]}} \tag{3.15}$$

$$\frac{\partial J}{\partial a^{[1]}} = \frac{\partial J}{\partial z} \frac{\partial z}{\partial a^{[1]}} \tag{3.16}$$

$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z} \tag{3.17}$$

Then using the equations above, we can rewrite the gradient of θ_t^1.

$$\nabla_{\theta_t^1} J = \frac{\partial J}{\partial w^{[1]}} = \frac{\partial J}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial w^{[1]}} = \frac{\partial J}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z} \frac{\partial z}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial w^{[1]}} \tag{3.18}$$

$$\tag{3.19}$$

Each solution to the partial derivatives can be easily found:

$$\frac{\partial a^{[1]}}{\partial w^{[1]}} = x, \quad \frac{\partial z}{\partial a^{[1]}} = \sigma\prime, \quad \frac{\partial a^{[2]}}{\partial z} = w^{[2]}, \quad \frac{\partial J}{\partial a^{[2]}} = y - x,$$

Which gives as the desired term $\nabla_{\theta_t^1} J$. This is the mathematical explanation of the back-propagation algorithm, as given more layers, the same rule can be applied. It is worth noting here that the partials used for calculating the first layer update $\nabla_{\theta_t^2} J$ ($\frac{\partial J}{\partial a^{[2]}}$ and $\frac{\partial a^{[2]}}{\partial z}$) are also used in the update for the next layer.

3.2.3 Convolutional neural networks

Although much of the math related to the neural networks is expressed as operations on vectors, the extension to higher dimensions is possible. A higher dimension of the input data can be image data, like a colour image represented as a tensor with HxWx3 * dimensions. Although the direct application of simple neural network architectures like MPL is possible, the memory requirements for fully connected layers of neural networks quickly grow. Before the inception of the convolutional neural networks, one way to mitigate this was to extract features from the image and then feed them to a neural network. Those features could be high-level (like the width of the eyes or the face to nose ratio in face recognition problems) or lower level (using visual filters and edge detection).

3.2.3.1 Classical convolutional methods

Many lower-level feature extraction algorithms could be boiled down to applying a convolutional filter (or kernel). The application of the convolutional filter can be expressed as operation on source image $f(x, y)$ that creates image $g(x, y)$ as:

*__H__, __W__ stand for __H__eight and __W__idth of an image, and 3 is the usual number of dimencions of color (RGB)

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x-dx, y-dy),$$

The convolution kernel of size *KxK*, is a real matrix *KxK*. Convolution operation must be applied on a window (a subset of image) of size *KxK*. The convolution operation is a sum of element-wise multiplication, between kernel and a window. This operation produces a scalar value. At each step of the application of the convolution filter, the window is moved by the arbitrary distance expressed in pixels. It is worth mentioning that depending on the size of the window, the output image will be smaller than the input. If necessary, this can be mitigated by adding empty (zero) pixels to the image's borders before applying the convolution (this technique is also called the image padding).

Some feature extraction methods can be expressed as convolutional filtering of the entire image. In this case, the configuration of the convolution kernel matters. In the Tab. 3.2.1 you can find examples of the application of a few kernels on an image.

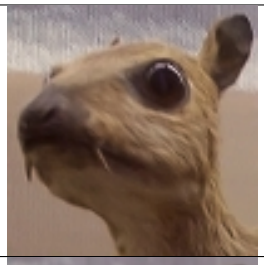
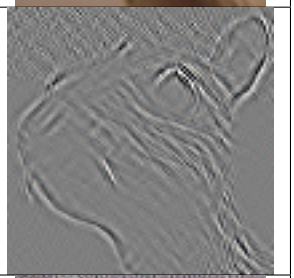
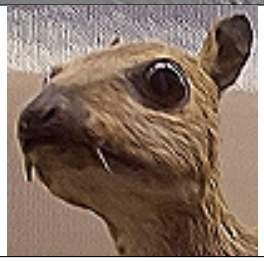
Filter name	Kernel	Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Gaussian Blur	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Ridge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Table 3.2.1: Examplary convolutional kernels applied on an image*.

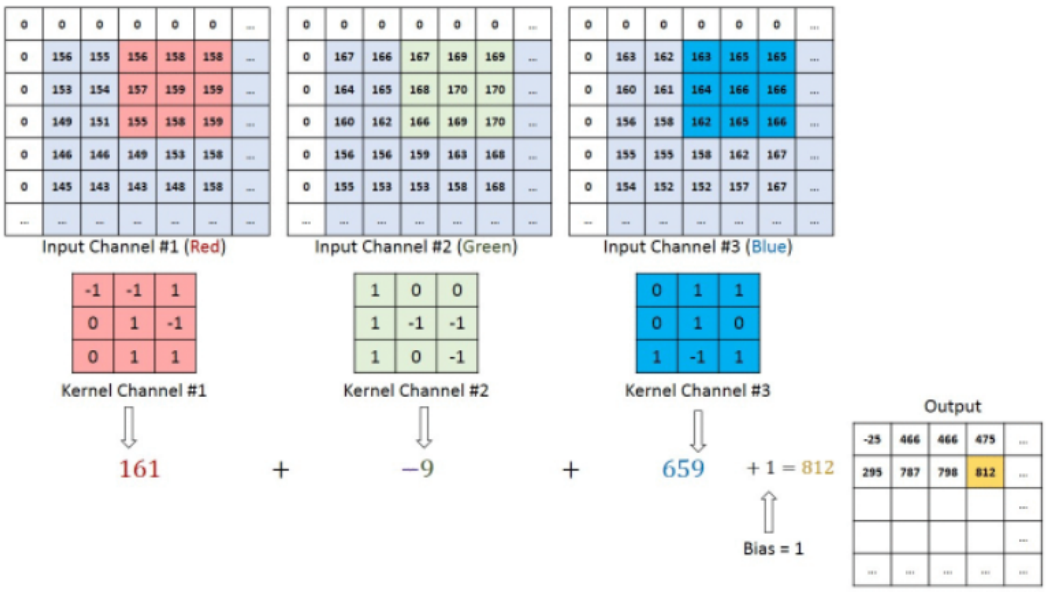


Figure 3.2.4: Step of application of a convolutional filter in a neural network *

3.2.3.2 Convolutional layer

A convolutional layer in a neural network is simply a layer that applies a convolutional filter. The properties of the filter, i.e. values of its elements, are learnable parameters. Usually, in the case of colour images, there are three layers of such filters, which outputs are summed as shown in Fig. 3.2.4. The values of the filters are trained by the backpropagation algorithm.

The application of the convolutional filters inside the neural network creates a map of more complex filters, which can extract the spacial features and build on other convolutional layers to learn the most valuable and significant features in a given task.

3.2.3.3 Pooling layers

The pooling layers in convolutional neural networks are used to decrease the number of features and filter the most significant spatial features (equivalent of denoising). The pooling layers use windowing on an incoming input to extract values given specific criteria. One of the popular pooling methods is max pooling. This method outputs the biggest value in a given window (see Fig. 3.2.5). The pooling layers do not have trainable parameters.

3.2.3.4 Convolutional networks: example architecture

The convolutional neural networks are nowadays a standard go-to solution for solving computer vision problems. There are many different architectures for convolutional neural networks. This section presents an exemplary implementation of the convolutional neural network. This architecture is known as VGG-16**?** .

*Source https://en.wikipedia.org/wiki/Kernel_(image_processing)

*Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

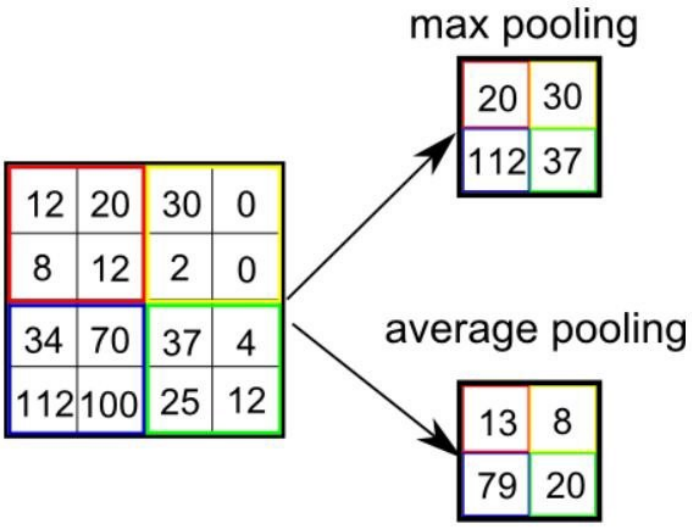


Figure 3.2.5: Example of max pooling and average pooling*.

A detailed description of the network is presented in Fig 3.2.6.

The network is considered to be a classical convolutional design. It assumes an input of the constant size and is divided into repeated blocks of 2 or 3 convolution layers, each followed by a ReLU activation function, with the last layer in a block being a Max Pooling operation. ReLU (Rectifier Linear Unit) function is defined as $f(x) = max(0, x)$. This means that its value is always zero for a negative number, and in other cases, it's x. Such blocks were repeated five times, with decreasing sizes in the first two dimensions and increasing in the last dimension. The last dimension represents a convolution feature output. After those convolutional blocks, there are three fully connected layers. The output is a vector of size 1000, as the network was initially created for the task of categorisation on the ILSVRC data-set containing 1000 categories.

The intuition behind this architecture is that the blocks of convolutions are responsible for extracting features. These convolutions build on top of each other and allow for more and more complex features to be learned. The fully connected layers allow for a linear combination of these features. Finally, the softmax function is used because it limits the output to range from 0 to 1 - which can be interpreted as a probability of given input belonging to a particular class.

3.3 Dimensionality reduction - Autoencoders

High dimensionality of data sets pose a major problem for many analysis tasks. It is often difficult to gain meaningful insights using only exploratory analysis of the data set or from simple metrics. Additionally, sparse, high dimensional data might pose a computational problem. Dimensionality reduction is a technique that reduces the number of dimensions of the data while retaining meaningful properties. There are many techniques, but here I will present the most basic one that stems from neural networks. This technique is known as autoencoder. Autoencoder is a deep neural network whose layers can be separated into two parts, Encoder and Decoder (Fig 3.3.1). The dimensionality reduction with autoen-

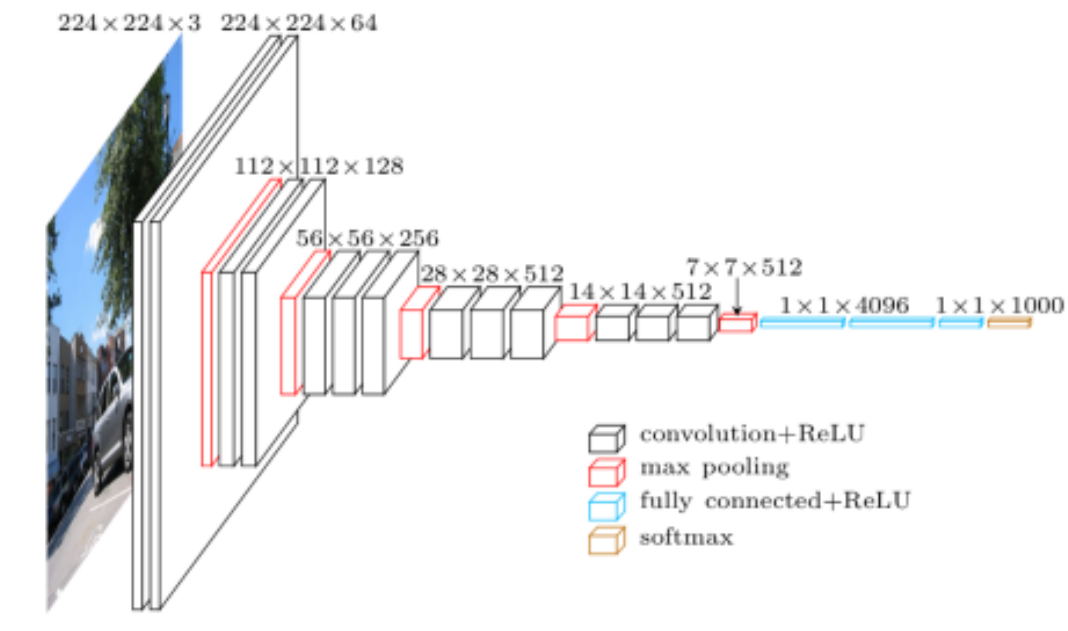


Figure 3.2.6: Graphical representation of the VGG16 architecture.

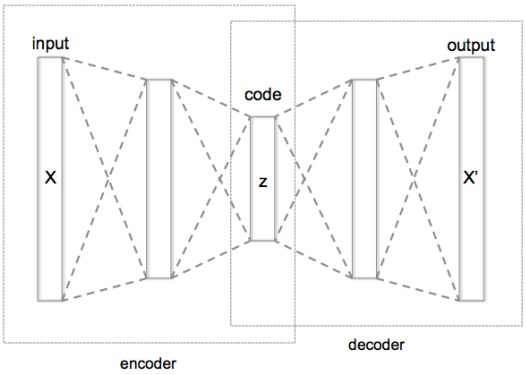


Figure 3.3.1: Graphical representation of the autoencoder structure*.

coders can be understood in the following way:

$$\tilde{X} = g(f(X)) \text{ where } \quad X\prime = f(X),\ g = f^{-1}$$

The f is an encoder, that reduces the input vector $X \in R^n$ to $X' \in R^m$. The g is a decoder that maps the vector X' to $\tilde{X} \in R^n$, thus recreating the dimensionality of the initial input. The goal of successfully trained autoencoder (encoder + decoder) is then to minimise a reconstruction error $f, g = \arg\min_{f,g} E(X - g(f(X)))$. The above can be implemented using neural network layers and arbitrarily small R^m, which will be the intermediate output from the encoder.

The encoder part of the autoencoder network is composed of fully connected layers with diminishing size, up to the desired size of the dimension required by the reduction. The last layer of the encoder is connected to the decoder part of the network, comprising fully connected layers growing in size up to

*Source https://en.wikipedia.org/wiki/File:Autoencoder_structure.png

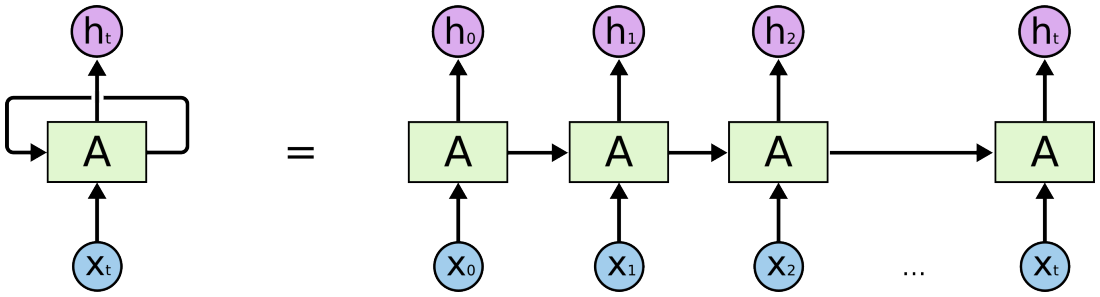


Figure 3.4.1: A representation of a simple neural network. The left representation is the rolled representation, and the right is a representation unrolled in the time dimension.

the original input size.

The encoder tries to compress the information to a smaller dimension (also called the latent space), while the encoder tries to uncompress this information and recreate objects from the original data space of dimensionality n.

3.4 Recurrent networks

Modelling of the ordered data sets is a very popular task and employs a special type of deep artificial neural networks. For instance we would like to preserve a temporal information for the modelled series of data points, where the next element may be influenced by the previous ones. Such a goal can be achieved by using recurrent connections in neural networks. The recurrent connection can be understood as a loop in a neural network graph.

3.4.1 Standard recurrent network

Generally we can describe the output from a simple recurrent neural network as follows:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \tag{3.20}$$

$$y_t = \sigma_y(W_y h_t + b_y) \tag{3.21}$$

Where elements W_h, U_h, W_y, b_h, b_y are weights shared in the temporal dimension, and assuming that σ_h, σ_y are activation functions.

The h_t term is a recurrent output, which will be used in the next forward pass of the network. It is composed of a current input x_t and the previous looped connection output h_{t-1}. That recurrent output is also used to create the output of the neural network y_t. Neural networks can be represented as unrolled in time dimension (Fig 3.4.1). Recurrent neural networks can be divided into categories based on the input, output and recurrent connection configuration (Fig 3.4.2).

The pitfall of the regular RNNs is their poor performance in tasks that require long term memory (tasks that require the model to hold certain information for an extended amount of time). Also, in the backward pass in RNN's there is a problematic partial differentiation $\frac{h_t}{h_{t-1}}$. If this term is consistently less or greater than 1, it may, in the end, cause this gradient to diminish to values close to 0 or reach very large values. This is also called the vanishing/exploding gradient problem.

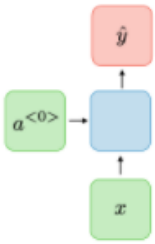
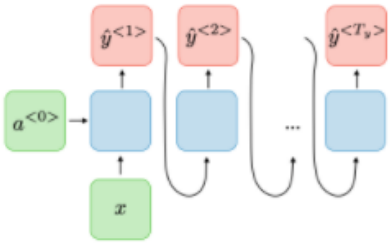
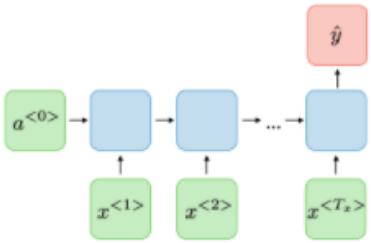
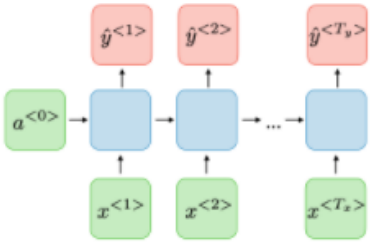
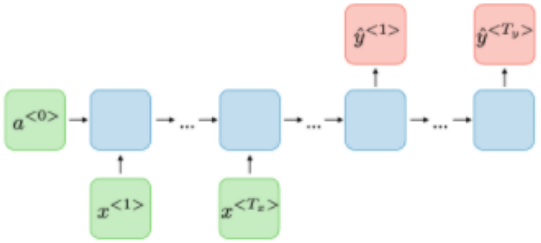
Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

Figure 3.4.2: Different recurrent neural networks, unrolled in time. The first example at the top is just a regular neural network. The second row represents a neural network which uses a single output and continuously produces output as many times as desired. The third row represents an inverted example when the network accumulates several different time step inputs and has a single output. This method can be helpful in sentiment analysis when a singular output marking sentiment is desired after a sequence of words. The fourth example depicts a RNN, which continuously produces output for each timestep. The last row shows that the number of inputs and outputs can be changed according to particular requirements*.

3.4.2 LSTM

The problems of the classical RNNs are partially solved by LSTM (Long short-term memory) architecture. It can retain information for more extended periods and partially solves the problem of vanishing/exploding gradient.

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

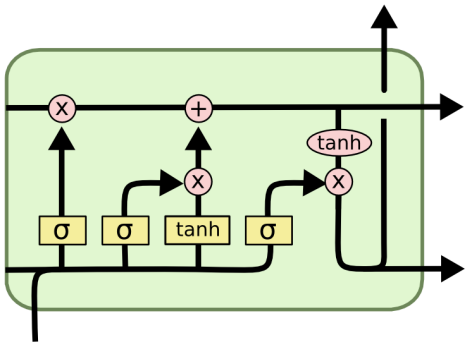


Figure 3.4.3: Architecture of LSTM. LSTM in equation form on the left and in graphical form on the right*.

Fig. 3.4.3 presents LSTM, there σ_g is sigmoid function, and the σ_c and σ_h are hyperbolic tangents. The LSTM's inner workings can be explained intuitively. The c_t vector can be assumed to represent the memory of the layer, as it is passed in a time dimension. f_t values range from 0 to 1, and thus can either let the values of c_t continue to propagate (multiplied by 1) or forget them (multiply by 0). The i_t values range from -1 to 1, and can add the new values to c_t.

3.4.3 WTTE-RNN

Another enhancement to the RNN models can be achieved by coupling them with the Weibull distribution. In that way we can create a model that can be able to predict a specific event (for instance time of the next aircraft engine checkup). The Weibull distribution (Fig. 3.4.4) is defined by the following equation:

$$
f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} & x \geq 0, \\ 0 & x < 0, \end{cases} \tag{3.22}
$$

The Weibull distribution is often used for modelling the "time-to-failure" in the survival analysis. For assessing the probability of an event (churn/survival), a version of Weibull log-likelihood is used (proposed in WTTE-RNN**?**):

$$
\lambda(t) = (t/a)^{\beta-1}\beta/a\Lambda(t) = (t/a)^{\beta} \tag{3.23}
$$

Where $\lambda(t)$ and $\Lambda(t)$ are hazard function and cumulative hazard function derived from Weibull's distribution. The term a and β are scale and shape parameters. In this case the value of beta is set to be constant $\beta = 1$. For incorporation of this distribution with the recurrent network, a log-like likelihood

*Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

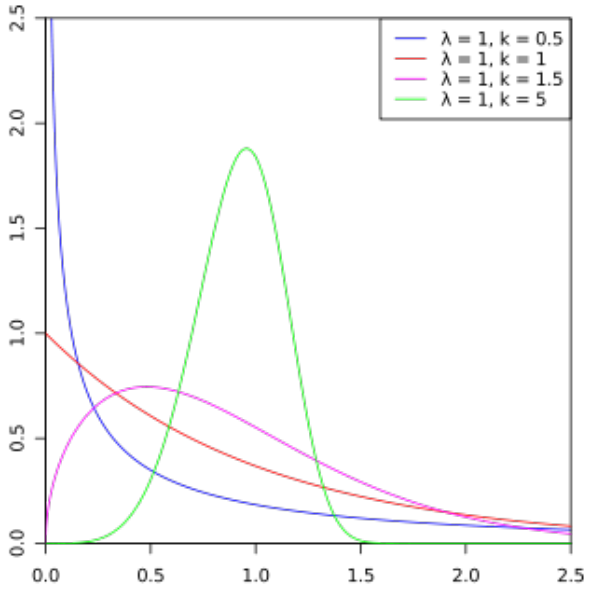


Figure 3.4.4: Weibull distribution with examplary paremeters*.

is used in the following form:

$$Loss = u * log(e^{\Lambda(t+1)-\Lambda(t)} - 1 - \Lambda(y+1))$$

And the custom activation function:

$$activation(x_i) = e^{x_0} + Softplus(x_1) \tag{3.24}$$

3.5 Deep Reinforcement learning

Many of the tasks in the real world do not have easy do define metrics and cannot follow input - output model of learning. In some areas, a reward response is more straightforward to define than the exact desired output. An example of this may be a game of chess - it is hard to define what set of moves will win with a given opponent, and in the end, there is only a win-lose response. Such problems are solved with Reinforcement Learning**?** .

3.5.0.1 RL formalism's

As mentioned previously, reinforcement learning focuses on the action being taken in a specific environment. This is represented on the flowchart in Fig. 3.5.1. More formally, in RL there are two important entities: **Agent** and **Environment**. The agent is an entity operating in the environment via **action**. The environment provides agent with **observation** and **reward**. While the observation expresses the state of the environment, the reward expresses how well the agent performed.

The environment in RL is usually expressed as an MDP (Markov decision process). For that, the system (environment) is expressed as observable and distinguishable **states** *s*. All of the observable

*Source: https://commons.wikimedia.org/wiki/File:Reinforcement_learning_diagram.svg

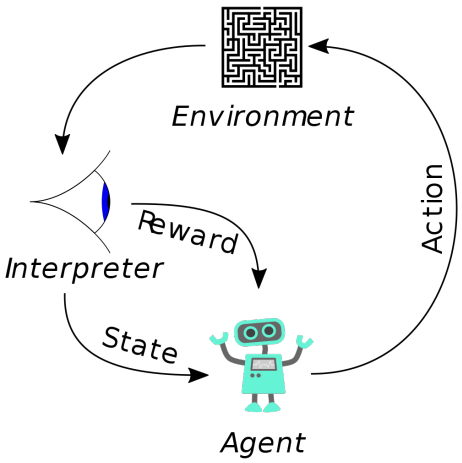


Figure 3.5.1: A diagram of the flow of information in a reinforcement learning setting*.

states create state space S. In MDP, the system states have Markov property, meaning that the following state is only dependent on the current state, not the previous states. This assumption is not valid for many of the environments and problems that actually have been solved with reinforcement learning. Also all of the possible **actions** a create action space A. The relation of action and states is expressed as function $P_a = Pr(s_t|s_{t-1}, a_t)$, which assigns a probability of a state given a previous state and action. And finally, the **reward** is expressed as $R_a = R(s_t, s_{t-1}, a_t)$, where an immediate reward is given for transition from s_{t-1} to s_t given action a_t. Those factors are usually expressed as MDP tuple (S, A, P_a, R_a). The last important element in the MDP formalism is policy π. $\pi(a|s)$ is a probabilistic mapping from S to A. The goal of MDP is to find a mapping π which maximises the reward.

3.5.1 Q-Learning

As stated before, reinforcement learning is seeking to maximise the cumulative reward. We can define this cumulative reward as return G_t, as a sum of the rewards in the ongoing time-steps**?** :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \ldots = \sum_{t=0}^{\infty} r_t \gamma^t \tag{3.25}$$

The γ $(0 \leq \gamma \leq 1)$ term in the equation above is called discount. It is used to parameterise and asses the greediness of the algorithm by discounting future rewards.

Q-learning is an algorithm for solving reinforcement learning problems. It is also known to belong to a subset of algorithms known as Value-based algorithms. This means the algorithm uses a state-value function $V(s)$, as defined in 3.26

$$V(s) = \mathbb{E}\left[G_t \middle| s_t\right] \tag{3.26}$$

In plain words: value is the sum of all possible future discounted (γ) rewards (r). The value can also be applied to an action, as after applying an action, the new state and its reward are given. Assuming a

policy π, we can also define a value of state s under the policy:

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t \middle| s_t \right] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} r_t \gamma^t | s_t \right] \tag{3.27}$$

We can also define a q-function Q_π, which given a state and action identifies the value.

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \middle| s_t, a_t \right] \tag{3.28}$$

We can extend Eq. 3.27 to the recurrent form, as the value function is dependant on future value functions.

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right] \tag{3.29}$$

Eq. 3.29 is called Bellman equation and represents the connection between the current value and future value.

For both state-value and action-value functions we can define optimal functions:

$$v_*(s) = \max_\pi v_\pi(s) q_*(s, a) = \max_\pi a_\pi(s, a) \tag{3.30}$$

Those are the optimal functions in sense of using the policy that maximises the value.

The belman optimiality equations for $q_*(s, a)$ is:

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma max_{a'} q_s(s', a') \right] \tag{3.31}$$

The optimal q action-value function may be aproximated by using a action-value table $Q(S_t, A_t)$. This table can be iteratively filled by choosing the actions using the policy derived by Q.

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a') \tag{3.32}$$

$$Q(s, a) \leftarrow (1 - a)(Q(s, a)) + a(r + \gamma \max_{a'} Q(s', a')) \tag{3.33}$$

The last equation above is the key equation begin the q-learning algorithm. This table can approximate the optimal solution to the given reinforcement learning problem. You can notice that the Q should contain all of the different state-action pairs, which for many problems is not practical. This is where the Q-learning turns to the neural network's algorithms, as instead of using the Q table, we can approximate Q using a neural network.

3.5.2 Deep Q-Learning

Given that there exists an optimal bellman action-value function $Q^*(s', a')$ **?** and it's value in the next time-step given the state s' and action a' is known, then the optimal strategy is to select an action which maximises the expected value of future reward:

$$Q^*(s, a) = E_{s' \sim \varepsilon} \left[r + \gamma \max_{a'} Q^*(s', a') \middle| s, a \right] \tag{3.34}$$

In terms of a target for the neural network, this can be specified as:

$$Y_t^Q = R_{r+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t) \tag{3.35}$$

The basic idea of Q-learning is to update the Q function/table iteratively with the above equation. As stated previously, this may not be practical and lacks generalisation. Instead, a parameterised function an approximate one may be used $Q(s, a; \theta) \approx Q^*(s, a)$. This Q function approximator can be a neural network parameterised by weights θ. Next, we can build a loss function $L_i(\theta_i)$

$$L_i(\theta_i) = \mathbb{E}_{s,a\ p()} \left[(y_i - Q(s, a; \theta)) \right] \tag{3.36}$$

where $y_i = \mathbb{E}_{s'\ \varepsilon} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \right]$

Assuming MSE as our loss metric, we can write down loss function as follows:

$$\mathcal{L} = (Q(s, a, \theta) - (r + \gamma \max_{a'} Q(s', a'; \theta_{i-1d})))^2 \tag{3.37}$$

This, in practice, creates a particular problem, as we need to process the states and the actions in the neural network Q twice, once to get the output that will be compared with the prediction and once to calculate future prediction. This creates a loop that can lead to the network's instability, as it will be chasing a target that is constantly changing. This is why usually, in practice, there are two separate networks (target network), one for calculating the prediction and the other for calculating the output.

3.5.3 Experience replay buffer

In the simple approach to reinforcement learning using DQN, the model would be trained on the rewards, states and actions that have been just exchanged with the environment. This means that there would have to be a step in the environment for each training sample. Additionally, this would make the model training less stable, as the recently visited states would directly impact the model the most. Because of this, for the model training, an experience replay buffer is used. The replay buffer (or memory buffer), is used to store the MDP tuples received from the environment instead of using them immediately and discarding them. The training and the exploration are logically separated. After a period of exploration (interaction with the environment), the training process begins on the gathered samples.

There are multiple ways to implement a replay buffer, but one of the most common one is to create a limited length list (first in - last out), where the old samples are discarded. Then such a list is randomly sampled for the training cases.

3.5.4 Deep Double Q-Learning

One of the problems of the DQN algorithms is over-optimistic value estimates. This is because when selecting the maximal value in the $\max_a Q(S_{t+1}, a; \theta_t)$ term we use one network pass to both choose (selecting the action of the max value), and calculating it's value (by returning only the maximum). The deep double q-learning algorithm**?** aspires to solve this problem by using an additional step of the network (eq. 3.38).

$$Y_t^{DoubleQ} = R_{r+1} + \gamma Q(S_{t+1}, \max_a Q(S_{t+1}, a; \theta_t); \theta'_t) \tag{3.38}$$

This calculation of the target Y is done using two sets of networks. Network θ_t is the network that is currently online learning and is used to pick the action, but the network θ'_t is used to estimate the value of this action. The values of θ'_t can be updated periodically with the weights of θ_t. This can also be done using blending of the parameters $\theta'_{t+1} = \beta\theta'_t + (1-\beta)\theta_t$ where $0 < \beta < 1$.

3.6 Probabilistic programming

Probabilistic programming**?** was born out of the Monte Carlo simulations and Bayesian reasoning. In brief: it uses a probabilistic approach to modelling and machine learning. It is considered to be a white-box approach. Probabilistic programming uses Markov Chain Monte Carlo (MCMC) to sample probability distribution function.

3.6.1 Metropolis-Hastings algorithm example

Metropolis-Hastings algorithm**?** is an MCMC based algorithm used for sampling a probability distribution. This process can also be used to approximate the distribution. Let's assume function $f(d_i)$ that is a probability density function:

$$f(d_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{d_i-\mu}{\sigma}\right)^2} \tag{3.39}$$

This function is parametrised by two parameters. Let's assume that the $\mu = 0$ is already known, and let's describe the unknown parameter $\theta = \sigma$. The task at hand is to approximate the $f(d_i)$. Now, at it's core, all Bayesian methods use simple, yet powerful Bayes theorem:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} \tag{3.40}$$

$P(\theta|D)$ is called posterior, $P(D|\theta)$ is the likelihood, $P(\theta)$ is the prior, and $P(D)$ is the evidence. In the case of machine learning tasks, we have data to test our beliefs, and so in this case, in order to find the approximation of $f(d_i)$, we would like to find the probability distribution of θ under the condition of existing data D.

For this purpose we can say that the following approximation is true:

$$f(x_i|\theta) \approx P(D|\theta) \tag{3.41}$$

The Metropolis-Hastings method uses a transition model $Q(\theta'|\theta)$ to generate candidates for the searched parameter θ. In this algorithm, Q is used for creating random steps θ in the distribution space. For example Q can be defined as $Q(\theta'|\theta) = N(\theta, \mu)$ and that $\mu = 1$ - a random walk. With each step, we assess if the new position θ' is more likely to fit the distribution. If it is not, we discard the newly generated position and sample again. Each step is directly influenced only by the previous step, which means it satisfies the Markov chain rule. The test of the likelihood of the step more likely belonging to distribution can be described as a ratio:

$$R = \frac{P(\theta'|D)}{P(\theta|D)} \tag{3.42}$$

Which can be expanded with the Bayes theorem, and then expressed in terms of a PDF

$$R = \frac{P(D|\theta')P(\theta')}{P(D|\theta)P(\theta)} = \frac{\prod_i^n f(d_i|\theta')P(\theta')}{\prod_i^n f(d_i|\theta)P(\theta)} \tag{3.43}$$

Then the acceptance probability can be explained as:

$$A(R) = \begin{cases} R, & \text{if } R < 1 \\ 1, & \text{otherwise} \end{cases} \tag{3.44}$$

The eq. 3.44 means that we will accept new θ' if the ratio R is greater than 1, but otherwise, we will accept it with probability R.

The steps of the Metropolis-Hastings algorithm can be described as the pseudo-code in Listing 1. The first step is to generate random θ. Then, we generate $\theta\prime$, and calculate the probability of belonging to the distribution for θ and $\theta\prime$. Then we use the acceptance rule to either accept or reject $\theta\prime$ as new θ. Those steps are repeated k times.

The process described in the listing 1 creates a "trace" of candidate θ parameters, as visualised in Figure 3.6.1.

Data: $\theta \leftarrow random$
for $k \in \{1, \ldots, K\}$ **do**
  $\theta\prime \leftarrow Q(\theta)$
  $prob\_theta \leftarrow f(\theta)$
  $prob\_theta\prime \leftarrow f(\theta\prime)$
  $R \leftarrow \frac{prob\_theta\prime}{prob\_theta}$
  if $uniform\_random(0,1) > R$ **then**
    $\theta \leftarrow \theta\prime$
end

Algorithm 1: Steps of Metropolis-Hastings algorithm

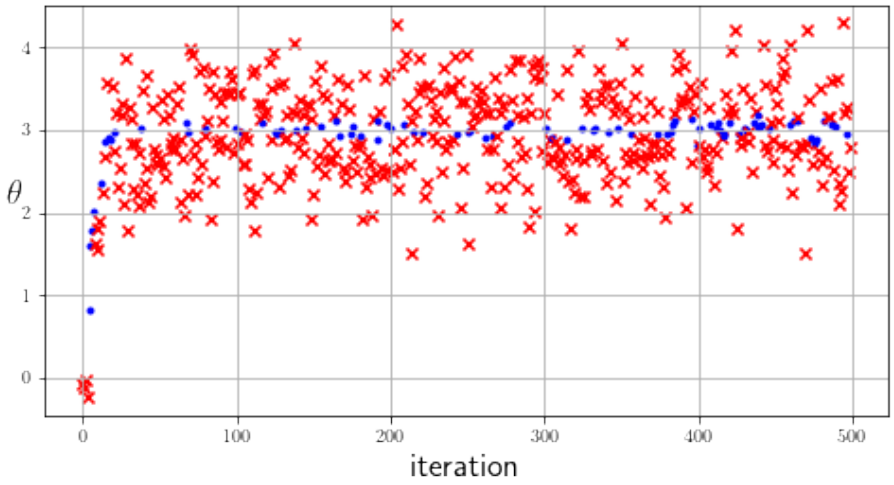


Figure 3.6.1: Exemplary steps of Metropolis-Hastings steps. In this example, the target distribution is $\mathcal{N}(\mu = 0, \sigma = 3)$. The accepted values are in blue, and the red crosses are rejects.

3.6.1.1 Tuning in Metropolis

The simple version of the Metropolis-Hastings algorithm assumes that the proposed $\theta\prime$ comes from a random walk $Q(\theta'|\theta) = N(\theta, \mu)$ with $\mu = 1$ but this creates a problem. Depending on the type of the task, the variance μ of the sampled distribution can result in either too small or too big steps. An additional effort can be made to find a step size μ better suited for our particular case. It has been proven that one of the good indicators of a well-picked step μ is the acceptance rate of 23.4%**?** . We set a tuning parameter as several additional steps of the algorithm that will be used to find a proper μ for the rest of the run**?** .

3.6.2 Probabilistic programming and Bayesian modelling

Probabilistic programming is a combination of programming paradigm and probabilistic modelling. It is usually conducted using programming frameworks, such as PYMC3**?** , and allows for automatic inference of the models.

Depending on the use case, it bears the marks of machine learning, as it is possible to obtain models via training (approximation of the parameters). The critical difference with ML methods such as neural networks is that in the probabilistic modelling, the models have to be defined at least approxi-

mately. This means that some knowledge of the statistical nature of the underlying modelled processes is necessary.

It also means that the trained model is not a black box, and its analysis can provide helpful insight and understanding of the model. Another advantage of Bayesian modelling is its generative capabilities. The MCMC process can be used to obtain the needed parameters (usually by taking the expectation value of the generated distribution). Then, these parameters can be used in the assumed underlying properties and distributions to generate synthetic samples of the data.

3.7 Clustering

In the general case, clustering is a process of grouping unlabeled data points into clusters that segregate cases with similar traits. There are many different clustering algorithms, each with its own preferred use case. In this section, I will present the most basic algorithm known as the K-means and two algorithms used in the later chapters. All of the presented algorithms are implemented as a part of the scikit-learn **?** package.

3.7.1 K-means

K-means (Fig. 3.7.1) clustering is a very basic algorithm. It divides the dataset absolutely (all of the points belong to a cluster) and expects the number of clusters k as the input.

The algorithm can be initialised by picking k data points randomly, and treating them as initial clusters' centroids m_k. Then, for the remaining data points the distance to each of the centroids is calculated $|x_i - m_k|$. Data points are assigned to the nearest centroids, and the set of the points assigned to each centroid creates a cluster. The last step of the algorithm is to recalculate the centroids $m_k^{(t+1)} = \frac{1}{\left|S_k^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j$

3.7.2 DBSCAN

In contrast to the K-means clustering, the DBSCAN **?** (Fig. 3.7.2) algorithm does not divide the dataset absolutely, as it doesn't assign all data points to clusters (there are points that do not belong to any cluster - outliers). Additionally, it does not require a number of clusters as the input. It does require a minimum number of points *MinPts* and a minimum distance ε.

The most important for understanding how the DBSCAN algorithm works is the idea of "core points". A point p is considered to be the core point if in the range lower than ε there is at least *MinPts*.

The steps of the algorithm can be described as follows:

1. The algorithm starts by choosing an arbitrary point p in the dataset.
2. First, the neighbouring points q that satisfy $distance(p, q) < \varepsilon$ are pushed onto the stack for further processing.
3. If the number of q is equal or greater than *MinPts*, then the point p is considered to be a "core point", and is assigned to the current cluster.

*Source: https://commons.wikimedia.org/wiki/File:DBSCAN-Illustration.svg

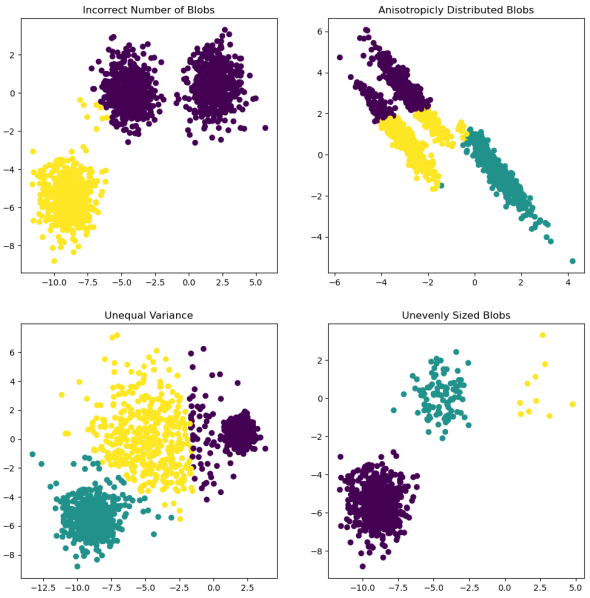


Figure 3.7.1: Performance of the k-means algorithm on synthetic datasets. The upper left plot shows the effect of the algorithm using k=2, but the data clearly has 3 clusters. The right upper plot shows how the k-means fails to work with anisotropically distributed data, and the lower-left plot shows how it performs with the data of unequal variance. The last lower right plot shows an ideal case of a k-means algorithm with a proper number of clusters, with equal variance and isotropically distributed data.

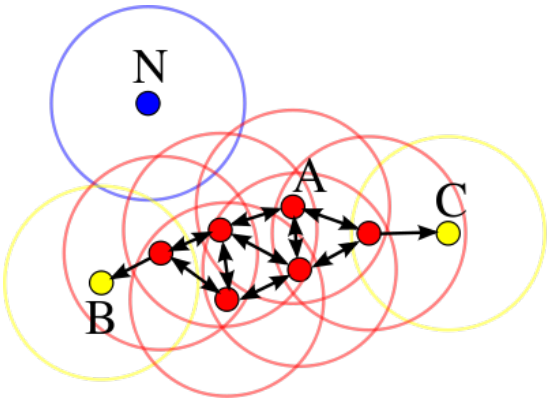


Figure 3.7.2: A representation of the DBSCAN clustering. The blue N dot is a noise point, as it does not contain any point within the given ε. The red points have at least two other neighbours and thus are core points. The yellow B and C points are border points as they are only connected by one other point.*

4. If that number is lower than *MinPts*, but one of those points is the core point, then p is the border point and also is assigned to the current cluster.

5. If none of the above applies, then p is labelled as "noise".

6. The process repeats by popping the stack and getting new p.

The DBSCAN algorithm can be understood as density clustering. If for two points p and q the following is true: p is a core point, and q is a core point, and $distance(p, q) < \varepsilon$, then we say that those points are joined by density edge. If for any two points p and q in the graph created by density edges there exists a path, then p and q are density connected. In this sense, we can say that any two points in a cluster are density connected.

3.7.3 OPTICS

The OPTICS**??** algorithm can be thought of as an extension of the DBSCAN, although by itself, it only provides a way of ordering points without assigning them to clusters. OPTICS requires only the minimum number of neighbouring points with respect to a core point *MinPts*, and the maximum allowed distance for a core point ε_{max} is optional but can speed up calculations. It introduces two important notions:

Core-distance - the minimum value of ε that makes the point a core point, given *MinPts*. The *core_distance* can be as small as needed but no bigger than ε_{max}.

Reachability-distance - a measure between two points p and q. It can be expressed as $max(core\_distance(q), distance(p, q))$ if q is a core point. It is undefined if q is not a core point.

Reachability-score - calculated per point. If p is a core point then this value is equal to *core_distance* otherwise it is equal to $reachability\_distance(q)$ where q is the nearest neighbour of p.

The algorithm itself relies on updating the output (reachability list). It traverses all of the points one by one by picking the point with the highest priority (lowest reachability). The algorithm starts with picking any starting point, and the proceeds as follows.

1. Calculate the reachability score for all neighbours of point p in the given ε range.

2. Update the sorted list of all unprocessed reachability scores.

3. Append the p to the end of the reachability-score list output.

4. Pick the top point (a one with the lowest reachability score) from the unprocessed reachability list and repeat until all points are processed.

This can also be expressed as the following pseudo-code:

reachability_scores ← ordered queue
ordered_points ← ordered queue
for *each unprocessed p in DB* **do**
    $s_i \leftarrow reachability\_scores\_of\_neighbours(p, \varepsilon, MinPts)$
    $update\_reachability(reachability\_scores, s_i)$
    $p.processed = True$
    $ordered\_points.append(p)$
    $p \leftarrow min(reachability\_scores)$
end

Algorithm 2: OPTICS algorithm steps

This process creates an ordered list of points where each neighbour of a point in the list is at a close distance from each other. However, it does not solve the problem of clustering. One of the more straightforward methods to do that is to use an epsilon cutoff - where we attribute those points that are below a certain threshold ε_{cut} to the noise and consider others to belong to clusters. Another method called ξ-steepness is based on the steepness of the reachability plot. Results of both methods are presented in the Fig. 3.7.3.

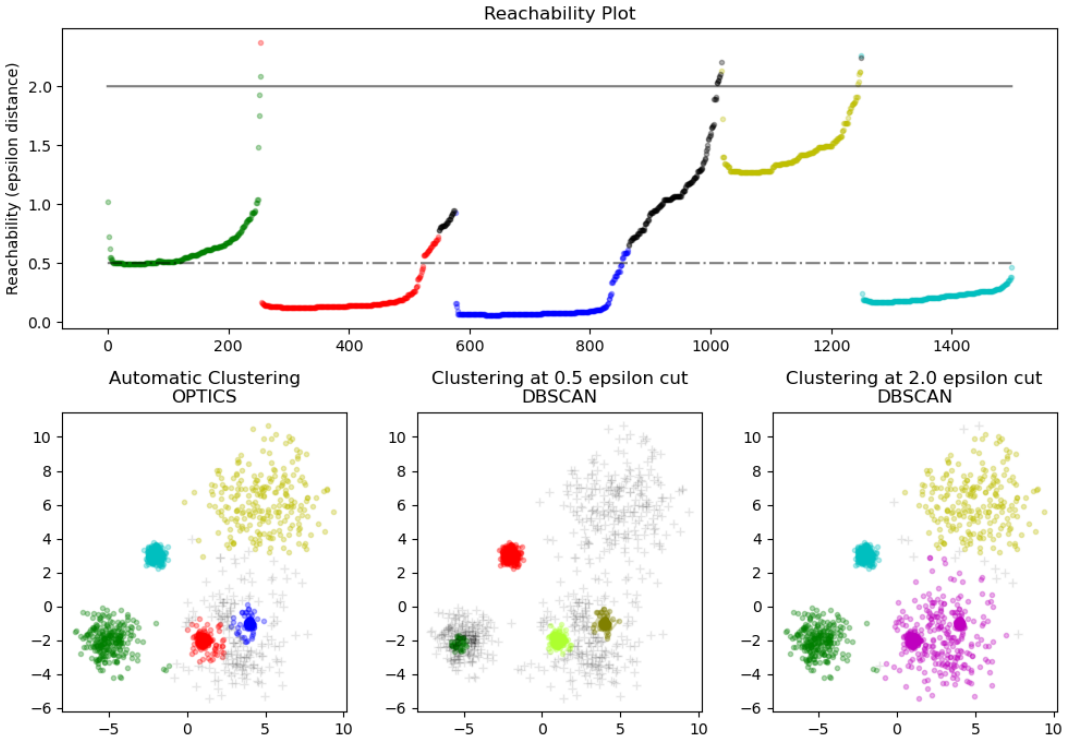


Figure 3.7.3: Reachability plot (above) of the exemplary dataset (row below). The reachability plot colouring was created for a ξ scheme of clustering. The first scatter plot in the second row represents clustering for the Xi scheme as well. The next two plots are for epsilon cuts as specified in the plot's titles. (source: scikit-learn)

Science isn't about WHY, it's about WHY NOT!

J.K. Simmons (Portal 2)

4

Machine Learning methods and analysis for strip Velo

The particle beam present at the LHC, while being a source of particle colissions that can unveil mysteries of the universe, is also constantly damaging the detector. The more particles is there flowing through the detector, the more the detector is damaged, the less the particles it will detect. The role of the maintainers of the detector is to balance on the edge of destruction of the detector, and expansion of knowledge. This carefull ballet motivates the work presented in this Chapter. The studies presented here, are hoped to foster the balance, by providing quick insights into detector behaviour.

The first Section (4.1) of this Chapter provides analysis of the calibration parameters of the Velo detector in years 2010-2017. This analysis shows how the radiation damage influences the calibration, and puts the bias voltage correction to the ultimate test. Additionally, using probabilistic programming and the insight from the analysis, we present an intelligent method for finding anomalious calibrations (Sec. 4.2). Which was succesfully implemented to the Velo monitoring.

The changes in the calibration may be elusive to a human operator, given a number of silicon strips (170 000). The usual plots and visualisation tools might not suffice to reflect the overall picture, and so we present the study of the dimensionality reduction techniques (PCA and autoencoders) applied on the calibration data (Sec. 4.3).

Next (Sec. 4.4), we answer he question: how does the calibration influence the signal? The answer led us to developing a tool that can help to a detector maintainer to decide about the timing of the calibration. The calibration process requires a specific condition of the detector, and must be carefully scheduled.

In summary, these studies are meant to equip the detector maintainers with intelligent tools that will

enhance the data taking, mainly in Run 3 at the LHCb, but also the detectors to which these methods might be transferable.

4.1 Run 1 and 2 calibration analysis

Runs 1 and 2 of the LHC machine were the initial runs of the Large Hadron Collider. Run 1 started in 2011 and lasted to the end of 2012, and Run 2 lasted from 2015 to 2018**?** . Because of the global pandemic, the plans for the years 2019-2022 changed, and the LHC actually will start taking data in 2022 (Fig. 4.1.1). Although the detector operated nominally throughout the time of its working, the study of its behaviour was needed to ensure that the Velo team at LHCb had a complete picture of its condition. The Velo being one of the core detectors, capable of performing the stand alone tracking and vertexing, provided the vital input for the physics programme.

4.1.1 The Data

The data used in this Section comes from a timespan 2010-08-18 to 2017-06-21 and contains 30 Velo calibrations.

Out of many different parameters the following ones were considered the most vital for the daily operation of the Velo detector: **Pedestals** P, **Low Threshold** L_t and **High Threshold** H_t. As discussed in the Sn 2.4.1.1 Low Threshold and High Threshold only differ in scaling factor, so in further analysis, only high threshold and pedestal parameters will be used in this Section.

The dimensionality and complicated nature of the Velo calibration dataset pose a serious problem for their efficient analysis. A single Velo sensor has 2048 readout channels, which constitutes the channel dimension. The sensors are of two types R and φ-type for each of the 42 modules, creating sensor-type and module number dimensions. Finally, each calibration is distinguished by the calibration date. This is summarised in Table 4.1.1.

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.

As mentioned above, the dimensionality in this dataset is challenging to navigate and most certainly needs a visualisation tool that aggregates a part of it. The most helpful graphical object for handling this kind of data is 2D histogram, where the channel number is defined on X-axis (horizontal), and a given parameter (H_t or P_t) value on the Y-axis (vertical), Example: Fig. 4.1.2. The colour of the histogram is the number of occurrences of a given parameter value in a given bin. When there is no occurrence of pairs of values, the colour is set to white for easier differentiation. In this Chapter, the colour scale is different for each plot unless mentioned otherwise. I do not provide the exact colour scale in most

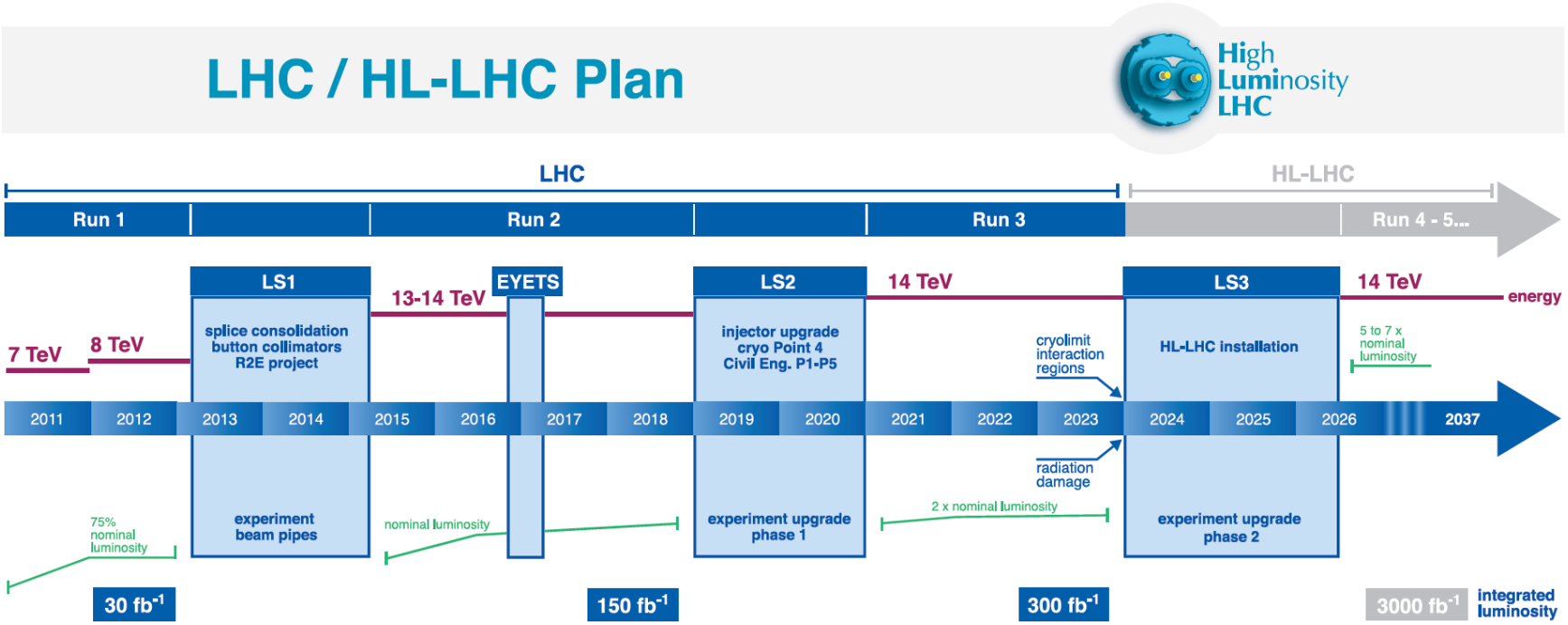
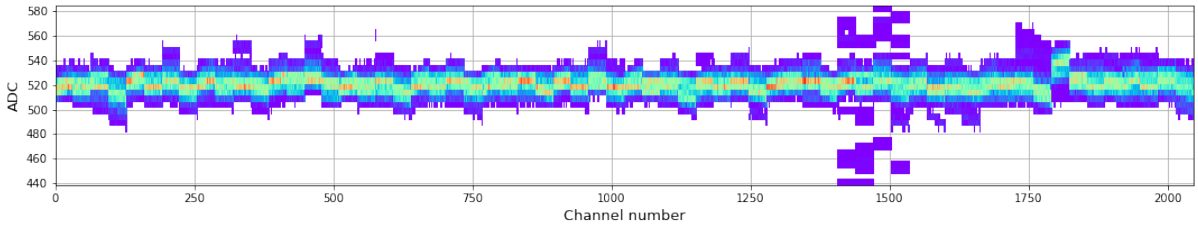
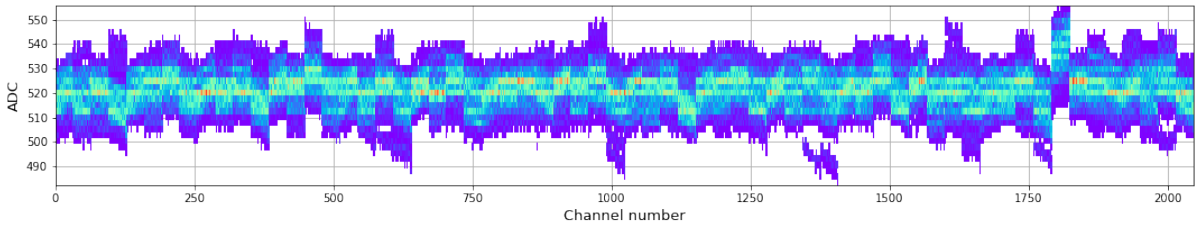


Figure 4.1.1: The schedule of the runs from 2019. This schedule has already been changed for 2020 due to the global pandemic.



(a) All phi pedestals.



(b) All R-type pedestals.

Figure 4.1.2: Histogram of the all of the pedestal values across all of the calibrations The histogram above is $P_{Ch,R,\#,T}$ and the below $P(Ch,\varphi,\#,T)$).

plots, but it follows the intuitive rule of warmer colours meaning more occurrences. This kind of figure has proved to be very useful since it can aggregate multiple dimensions of the data.

For ease of understanding, I introduce the following notation of the dataset slice:

$ParameterType(Ch,S,\#,D)$,

where *ParameterType* stands for the type of the parameter like P or H_t and the other symbols are explained in the Table 4.1.1. A single symbol without values marks the use of a full range of data on that dimension. Exemplary notation: $P(Ch100-Ch500,R,\#11,D)$ is a slice of R-type sensor channels from 100 to 500 in module $\#11$ in all of the calibrations. As one can see, even the slicing procedure is quite challenging and needs to take into account many aspects of the whole calibration dataset collected for Velo during its operation.

4.1.2 Pedestals

Pedestals represent the absolute electronics offsets measured in each detector readout channel. Their variations have proven to influence the data taking significantly (see Sec. 4.4). A special procedure, called noise data taking, was designed and put in place for Run 1 and Run 2 in order to evaluate the pedestals for each individual channel of the Velo detector. Basic exploratory analysis of this parameter starts with an overall look at all pedestal values in the calibration dataset (see Fig. 4.1.2). The pedestals oscillates around 520 ADC, with visible artefacts at channels 1400-1520 in R-type type sensor and near channel 1750 in both sensor types. The first artefact (ranges 1400-1520) comes solely from the sensor #85 , visible in the Fig. 4.1.3. This is believed to be a sensor malfunction (a visual inspection identified surface cracks). The other artefact near channel 1750 is believed to be coming from a design feature, identified as coupling to the digital clock line. This is not a significant problem for the overall data quality, since the effects can be precisely evaluated and seem to be fixed in time.

The intense mixed radiation field, consisting of fast hadrons, gradually damaged the sensors during

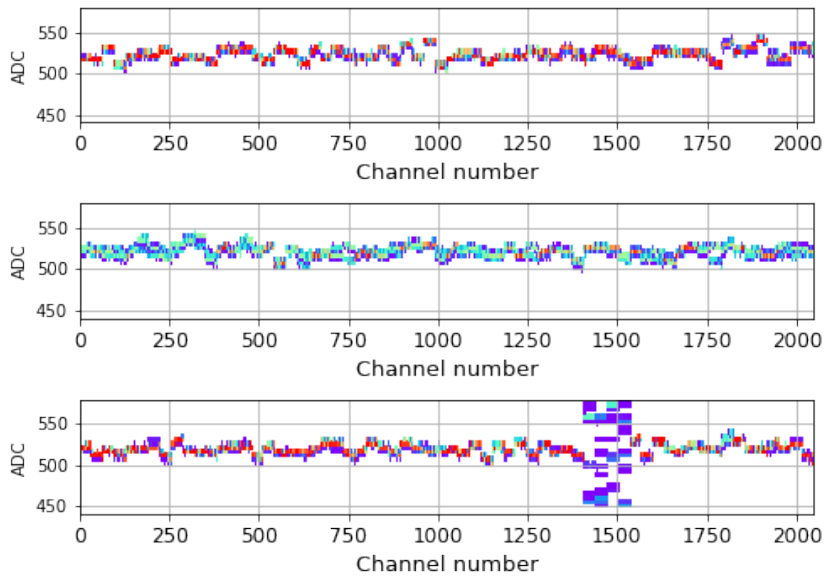


Figure 4.1.3: Pedestal plot for sensors (from the top) $P(Ch, R, \#64, T)$ $P(Ch, R, \#35, T)$ $P(Ch, R, \#85, T)$. $\#64$ and $\#35$ represent a typical pedestal values, while in $\#85$ there is a malfunction visible close to channel 1500.

their operation, resulting in the need to adjust the bias voltage. Proper estimation and correction for the pedestal value turned to be the critical part of the calibration procedure for Velo. The nature of this quantity is complicated and reflects the overall properties of the whole electronics readout chain. In principle it should not change significantly over time under stable conditions of the hardware. Any strong variations could be interpreted as serious problems with the detector. Thus, a study of the pedestal trend is an essential aspect of this analysis. The trend was calculated by fitting a line $y = ax + b$ to the pedestal values using the time of the calibration as an independent variable. The value of the linear coefficient a, which is responsible for the trend, is calculated individually for each of the detector channels. The distribution of this coefficient is presented in the Fig. 4.1.4. The mean value of the coefficient is $9.63e^{-5}$ which is so close to zero that it can be assumed that there is no overall trend, which proves that the detector did not suffer from any systematic effects, related to its hardware component, during the Run 1 and Run 2 data taking periods.

4.1.3 Clusterisation Thresholds

The high threshold parameter, H_t, is the one actually responsible for the sensitivity the hit detection. It is one of the many steps of filtering the signal coming from Velo. Further, the hit information is crucial for the tracking algorithms that aggregate the collection of hits into a track and fit its trajectory (see Sec. 2.4.1.1). Again, we start first at the simple exploratory analysis of the whole dataset. At the first glance, the 2D histograms presented in Fig. 4.1.5 are cluttered by outlying values. The most visible ones are near value 127 across all channels. This is an expected sight, as value 127 is the maximal possible value. Setting the value of the threshold in the channel to 127 means that only the signal that exceeds this threshold is accepted. Which, in reality, means that no signal will come through the channel as it would have to reach values greater than the dynamic range of the ADC value after the pedestal subtraction

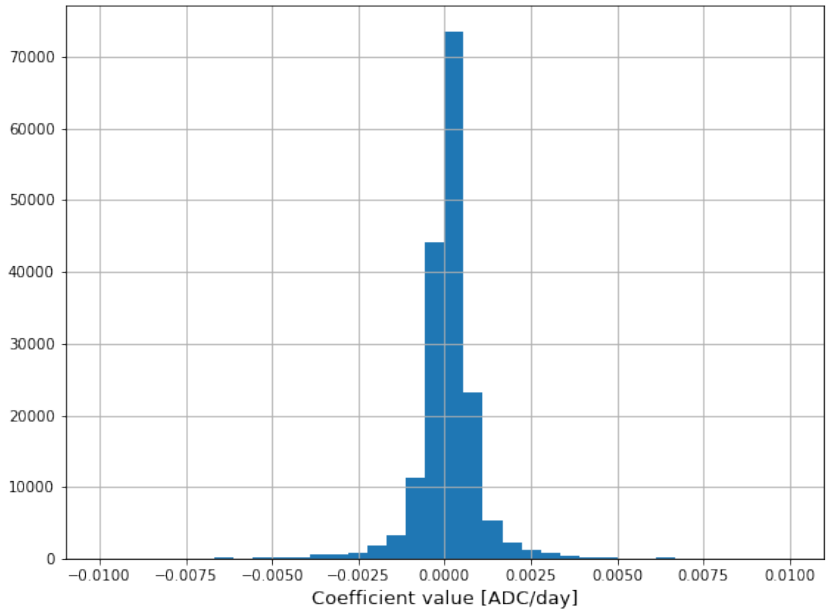


Figure 4.1.4: Histogram of linear coefficients of the pedestal trend per channel. Some of the outlying coefficient values have been trimmed and are outside the scope of this plot.

*. This, forces the data coming from such channels to be discarded. This is a masking value, evolution of the distribution of those masks are discussed in detail in the next Section (4.1.4). The other visible artefact are spikes within channel range fron 1000 to 1250. This comes from faulty sensor #67 and dates '2016-11-07' and '2016-11-11', where the threshold values reached non-physical value of 400 ADC. This is likely an error that could accidentally get into the dataset, and yet there is no explanation for those values (most likely an electrical disturbance influenced severely the global offset level). Moving on, we will treat those values as outliers and remove from the analysis.

The Fig. 4.1.6 depicts $H_t(Ch, R, \#, D)$ and $H_t(Ch, \varphi, \#, D)$ in range of values from 0 to 30. One of the immediately visible features are the sparse spikes regularly distributed across the sensors. Those spikes represent, so called, header cross-talk effect (Sec. 2.4.1.2), that affects specific channels during the analogue transmission of the data from the detector to the trigger processing farm. In some cases, it might be helpful to exclude the channels that contain the header cross-talk effect. The $Ch*$ denotes all of the channels without the header cross-talk effect. The reader can examine the difference of the histogram with and without header cross-talk in the Fig. 4.1.6 and 4.1.7.

Another insight coming from the 2D histogram presented in Fig. 4.1.7 is that low-intensity regularities repeat roughly every 512 channels. The periodicity of those regularities comes from the sensors' physical design and the specific topology of the strip implants. More precisely, they are related to imperfect calibrations coming from two dates: '2012-07-30', '2012-08-01'. Those two dates are imperfect due to the improper response of the high voltage system that failed to provide bias voltage to the sensors on those dates. In consequence a very large noise was observed. The particular shape of the noise

*The dynamic range of the Velo readout chain after signal digitisation and before pedestal subtraction is 10 bits without sign. Pedestal subtraction aims at equalisation of the properties of the readout channels across the detector. After the pedestal correction the dynamic range corresponds to 8 bits with sign, so effectively, we can observe signals in the range of -128 to 127 ADC counts.

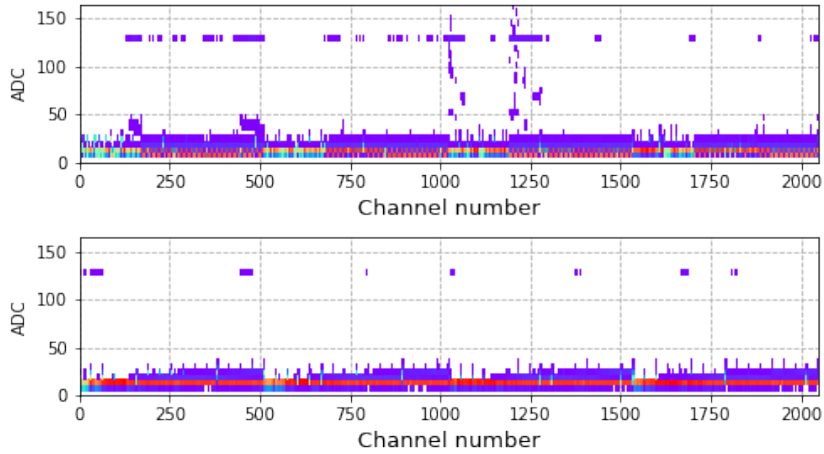


Figure 4.1.5: High threshold distribution of all of calibrations, across all sensors. $H_t(Ch, \varphi, \#, D)$ is above, and $H_t(Ch, R, \#, D)$ is the plot below.

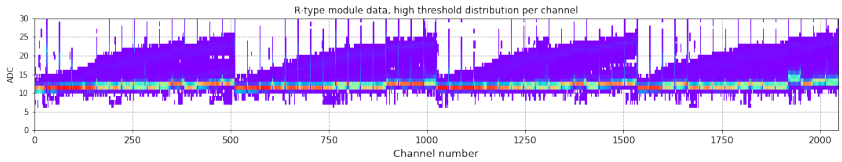


Figure 4.1.6: High threshold distribution with header cross-talk $H_t(Ch, R, \#, D)$.

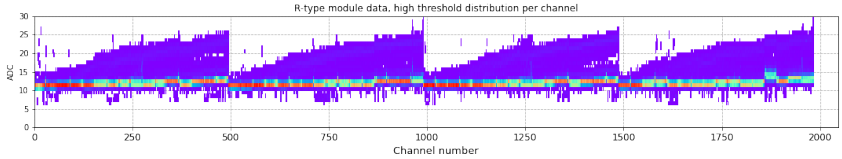
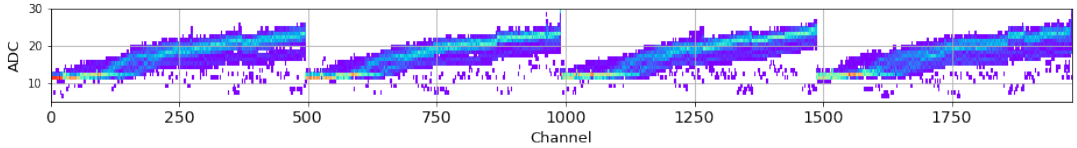
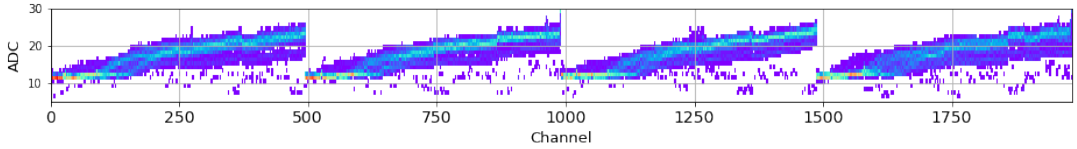


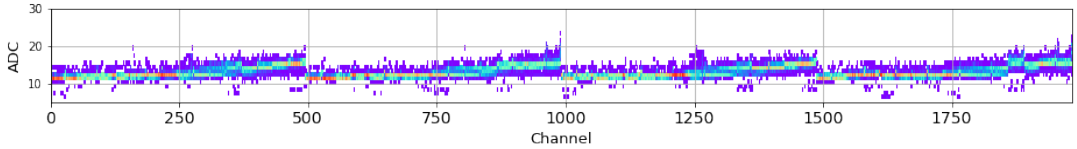
Figure 4.1.7: High threshold distribution without header cross-talk $H_t(Ch*, R, \#, D)$



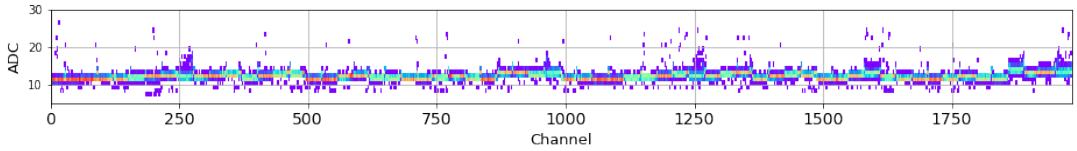
(a) $H_t(Ch*, R, \#, 2012-08-01), X = 10$



(b) $H_t(Ch*, R, \#, 2012-07-30), X = 10$



(c) $H_t(Ch*, R, \#, 2012-08-02), X = 3$



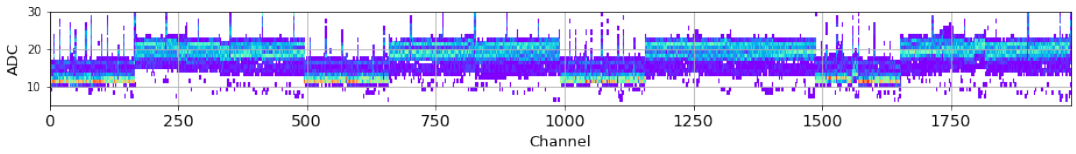
(d) $H_t(Ch*, R, \#, 2011-03-07), X = 1$

Figure 4.1.8: Histograms of imperfect calibrations in R-type sensors with their outlierness value X.

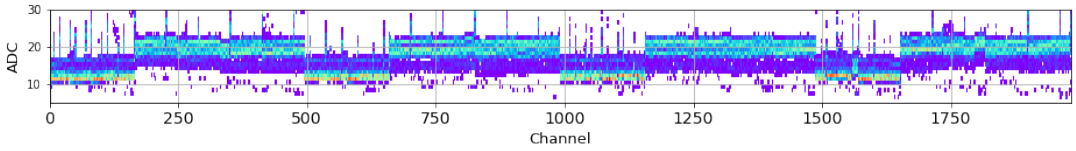
distributions for $R-$ and φ-type sensors reflect the physical dimensions of the strips. For the former strip length increases gradually, in each of the sensor's four sectors, as a function of radius. For the latter strip length is constant for inner and outer sectors (with shorter strips in the inner region). These imperfect calibrations influenced the sensitivity of the detector and impacted severly the Velo data quality. Additionally, for the purpose of outliers analysis, two calibration dates also are being recognised as imperfect 2012-08-02 and 2011-03-07. The overall four imperfect calibrations can be seen in Fig. 4.1.9 and Fig. 4.1.9. Those occurrences of mis-calibrations were a direct motivation for detecting anomalies in the calibration in Section4.2. Lack of such automatic procedures led to data loss due to very low hit reconstruction efficiency of the Velo.

4.1.4 Analysis of the outliers

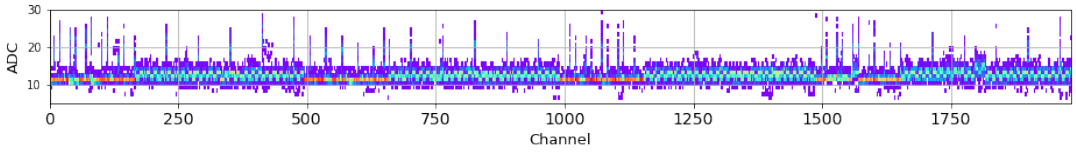
As mentioned previously, the $H_t = 127$ ADC means that the channel is masked. Setting the threshold so high means that no hit will be registered in a given channel. The masks were given to the channels that exhibited exceptional noise levels, were associated with broken or cold bonds, were connected to problematic readout chips or were close to sensor surface damage. Their occurrences are depicted in Figs 4.1.11 and 4.1.12. In 4.1.11 the channels from all calibrations are plotted as a number of blocked channels per sensor in the time domain. The important insight is that some sensors are more prone to masking than others and that the amount of masked channels changes. In some cases, the masked channels can go away and return to the sensor. The figure 4.1.12depicts a number of masked channels in



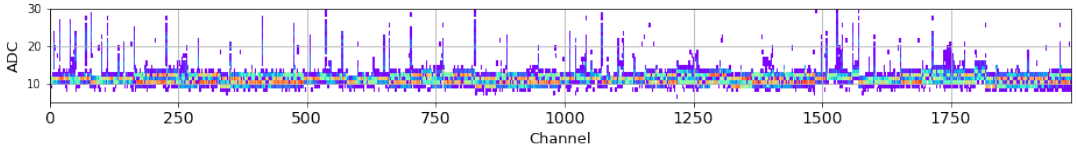
(a) $H_t(Ch*, \varphi, \#, 2012-08-01), X = 10$



(b) $H_t(Ch*, \varphi, \#, 2012-07-30), X = 10$

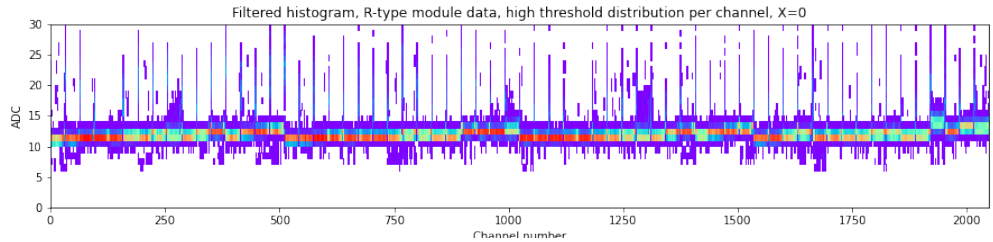


(c) $H_t(Ch*, \varphi, \#, 2012-08-02), X = 3$

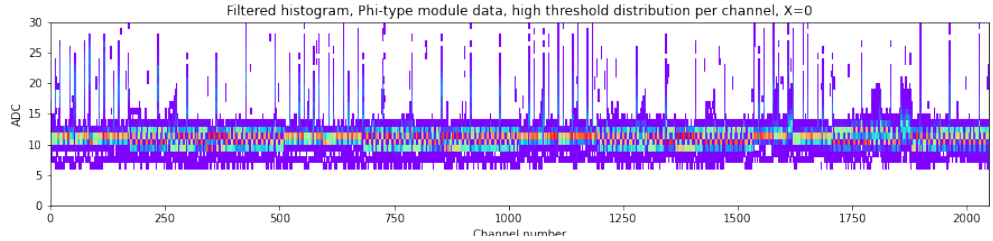


(d) $H_t(Ch*, \varphi, \#, 2011-03-07), X = 1$

Figure 4.1.9: Histograms of imperfect calibrations in '' sensors with their outlierness value X.



(a) .



(b) .

Figure 4.1.10: All of the calibrations, with reduced dimensionality using autoencoder and PCA.

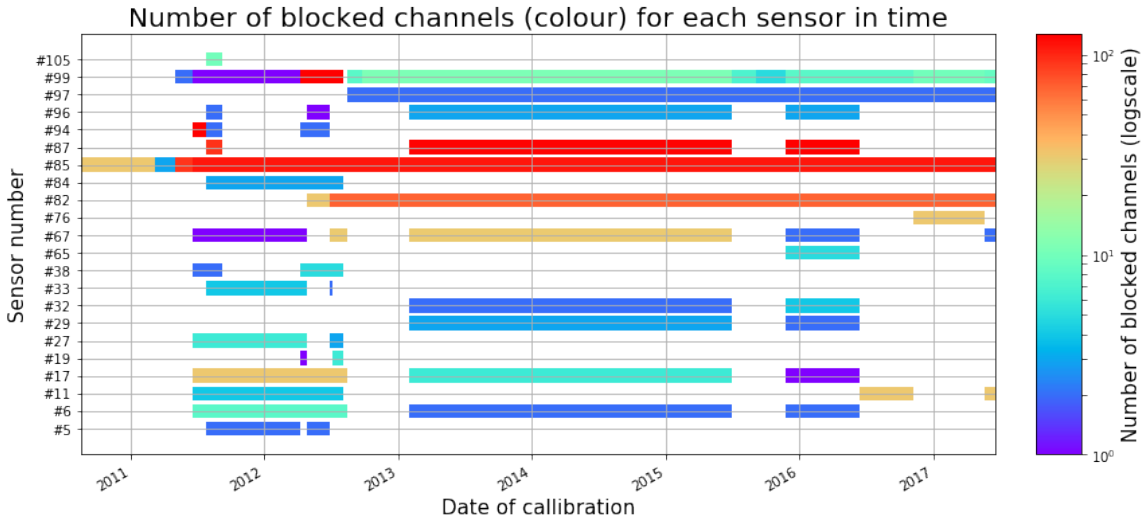


Figure 4.1.11: Distribution of blocked (masked) channels per sensor in time.

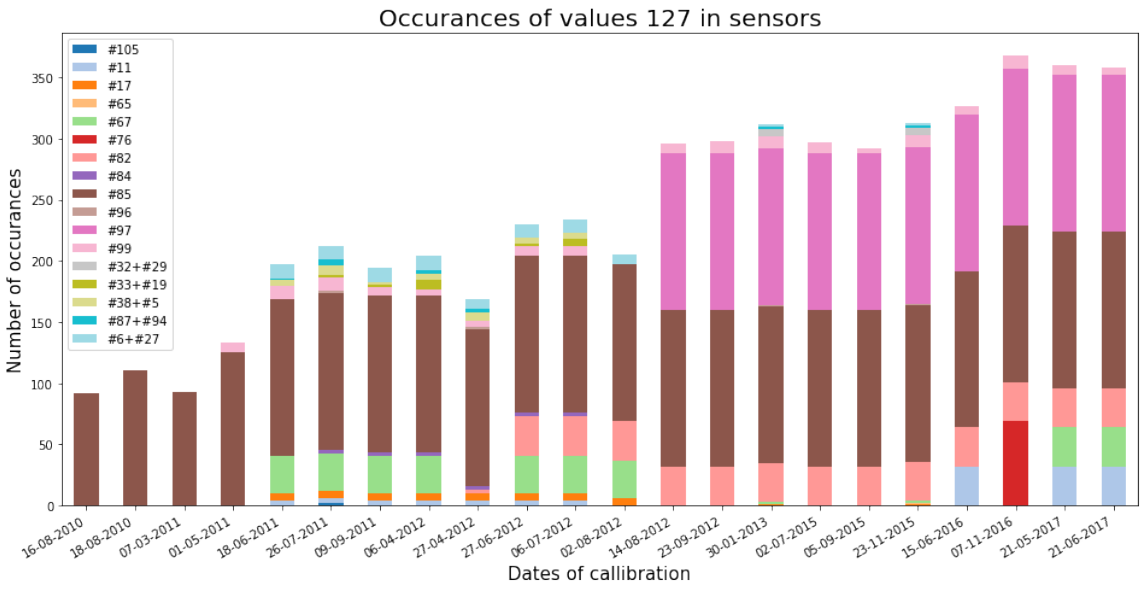


Figure 4.1.12: The total number of occurrences of masked channels versus time.

the time domain with a sensor split. It is clearly visible that the total number of masked channels grows with time. This is expected as with time the negative radiation effects accumulate (see Sec. 2.4.1.3).

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 $H_t > 50ADC \wedge H_t \neq 127ADC$. Fig 4.1.13 depicts those other outliers. The number of occurrences of these outliers changes from calibration to calibration, but overall is limited to three sensors: #85 #67 and #94 (all φ-type sensors). These anomalies have never been clearly characterised. The visual inspection did not indicate any surface damage on sensors, so that would point to instabilities in the electronics readout chain. Nevertheless, these bad channels were accounted for and did not affect the data quality.

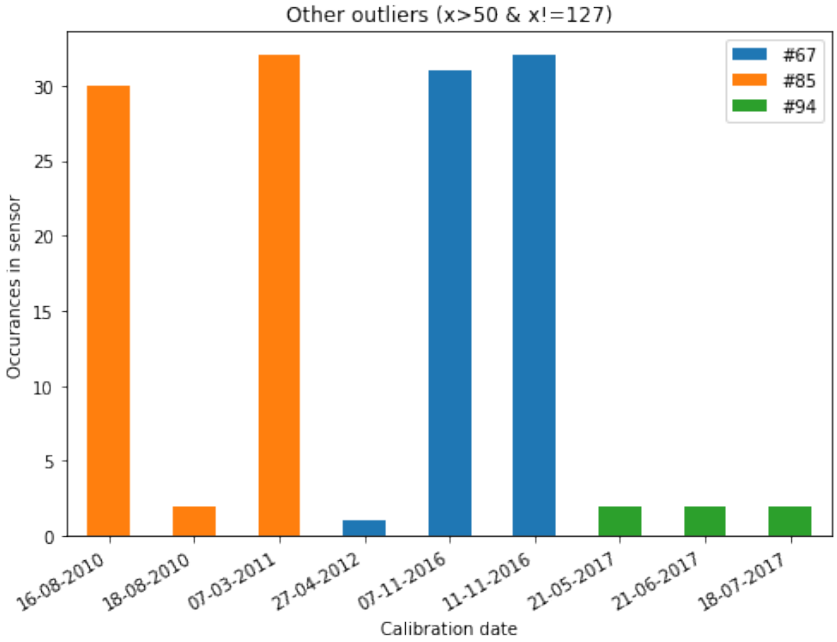


Figure 4.1.13: Anomalous values other than masked channels.

4.2 Intelligent analysis of the threshold data with probabilistic programming

The threshold evaluation procedure in the Velo directly influences the overall data taking quality in LHCb. If the thresholds are too high, then there is a missing signal that decreases the precision of the track reconstruction and, in turn, decreases the trigger efficiency. If the thresholds are too low, then the channel contributes to the noise and also may harm the quality of the data and stress the trigger system with a large number of fake hits. Calibration process also requires time without beam, which is beyond the control of the LHCb and is the LHC collaboration responsibility. To facilitate the correct calibration process, an additional check is advised. The next Section presents the method for detecting anomalies in the calibration process.

4.2.1 Dataset creation

The exploratory analysis of the calibration data presented in the previous Section yielded a helpful insights into the properties of the high threshold, H_t, parameters. Since high thresholds are evaluated using noise measured for each individual channel for R- and Φ-type sensors we expect it to be within a well defined interval that should not depend on time *. It is assumed further that the distribution of the high thresholds should approximately follow a Gaussian distribution for properly configured detector. One should note, that the electronics noise has a number of components one of which is proportional to the strip capacitance, that in turn, is proportional to the strip length. For properly biased silicon sensors this component does not contribute significantly to the total noise, thus, we can assume the noise for R-type sensors stay constant within one ADC count. The same behaviour is measured for Φ-type ones. Complete opposite behaviour is observed for the improperly configured sensors, i.e., the noise

*Properly configured detector should yield a noise that is approximately independent on time, however, there may be non trivial dependence on the radiation damage induced by fast hadrons.

component proportional to the strip capacitance completely dominates the total noise value measured on each strip. This effect led to inventing an auxiliary variable, called "outlierness", that can quantify the departure of the noise distribution from the regular operation conditions.

The "bad" calibrations were assigned an "outlierness" value X. This "outlierness" measure is purely artificial and subjective and also expresses the strength of belief that a given calibration is an outlier. The bigger the number, the stronger the belief that calibration is an anomaly. 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, and the related calibration parameters can be seen in Fig. 4.1.9a-4.1.9d and Fig. 4.1.8a-4.1.8d.

Table 4.2.1: Calibration dataset and their corresponding outlierness 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

Although the method was developed for both R- and Φ-type sensors, we will be using only the R-type sensor data for simplicity in this subsection. For the Φ the procedure is exactly the same. I will use the following notation to describe the high threshold:

$$T\prime_n = H_t(n, R, \#, D\prime) \tag{4.1}$$

$$T_n = H_t(n, R, \#, D) \tag{4.2}$$

Where n stands for a channel number across all sensors in time, and $D\prime$ means calibration date with the assigned outlierness values $X = 0$. As mentioned previously, it is assumed that the threshold distribution in a particular channel follows a Gaussian distribution.

$$T\prime_n \sim Gaussian(\mu = \mu\prime_n, \sigma = \sigma\prime_n) \tag{4.3}$$

As presented in 4.1.10, the value of the normal calibration for the R-type sensors oscillates around 12 ADC counts. Therefore the parameters $\mu\prime_n$ and $\sigma\prime_n$ can be calculated by fitting the channels histogram to a Gaussian distribution. In this case, the probabilistic programming paradigm is used to achieve that. We use the dataset containing only "good" calibrations $(D\prime)$ to calculate that.

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

$$T_n \sim Gaussian(\mu = X * \mu_n + \mu\prime_n, \sigma = X * \sigma_n + \sigma\prime_n) \tag{4.4}$$

The X, as previously, stands for the outlierness of the calibration, and μ and σ are coefficients of the linear dependency on the X. These parameters are taken as an intrinsic property of the n-th channel. The X value is the same across the calibration date for all channels and sensors. Notice that when $X = 0$, the model in Eq. 4.4 is equivalent to 4.3. This allows to use the parameters $\mu\prime_n$ and $\sigma\prime_n$ calculated for the previous base-line model to be used in the extended one. Given X from the Table 4.2.1, we are able to calculate μ_n and σ_n.

4.2.3 Training

This entire model was implemented using PYMC3**?** framework, and was trained using the PLGRID computation network. The training process was split into two parts, embodying two parts of the models. The first part is visible in Listing 1, and the second in listing 2. The first part uses only $D\prime$ calibrations, and the second part uses $D \setminus D\prime$. Both of the learning processes were conducted with 80000 training steps and 20000 tuning steps.

```
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, chains=1)
```

Listing 1: A snippet from the training process for the first part of the model (Eq. 4.3)

```
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, chains=1)
```

Listing 2: A snippet from the training process for the second part of the model (Eq. 4.4)

4.2.4 Results

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

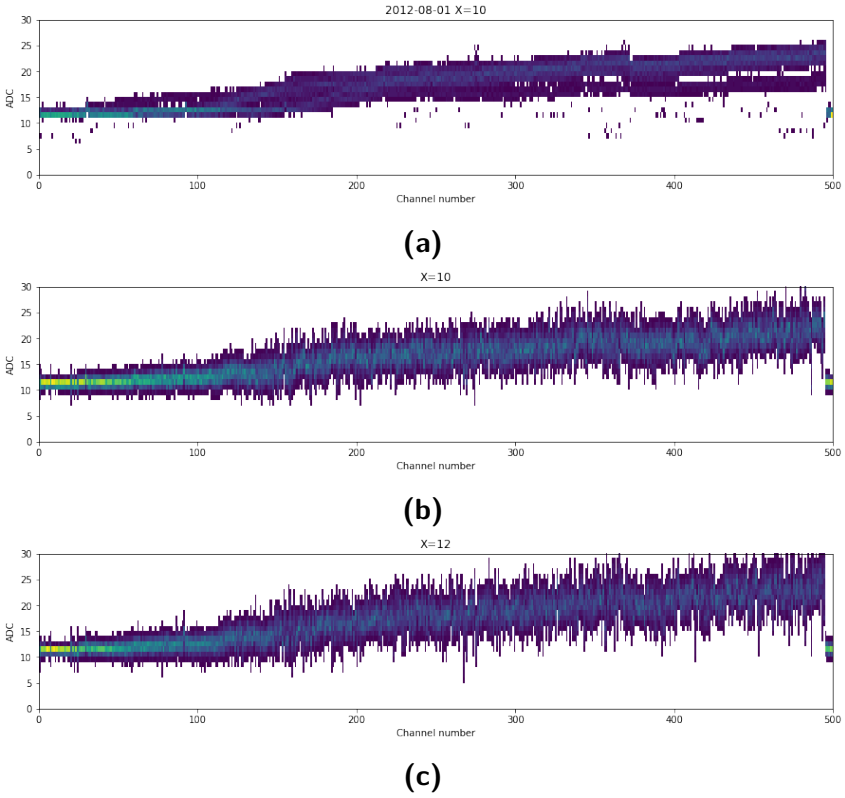


Figure 4.2.1: 500 channels from a (a) single calibration date marked with X=10, (b) an example of a single generated calibration parameters at X=12, (c) an example of a single generated calibration parameters at X=12.

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

- Partial dimensional independence - This model can be used to extract the value of the outlierness not only for the entire calibration but also for a specific sensor, or module, or a group of channels, or even just a specific channel. This allows for a more detailed insight as it can bring focus to one particular part of a detector that exhibits more anomalous behaviour.
- Generation of artificial data - Because the probabilistic programming is based on statistical inference, simply by calculating the μ, $\mu\prime$ and σ, $\sigma\prime$ parameters of a Gaussian distribution we can generate a high quality artificial sample.
- Interpolation and Extrapolation - This model was trained only on four different values of X, and yet, it is capable to assess the outlierness value not only in the range between those four values (e.g. $X = 5$), but also through the generation of artificial data, it can show what a calibration that is outside the scope of observed calibration dataset (e.g. $X = 12$) might look like. This feature is also essential for detecting any further anomalies.

In Fig. 4.2.1a there is a sample of the data (500 channels) that comes from one of the improper calibrations. It can be noted that the range of the H_t values is on par with those visible in the plot of the

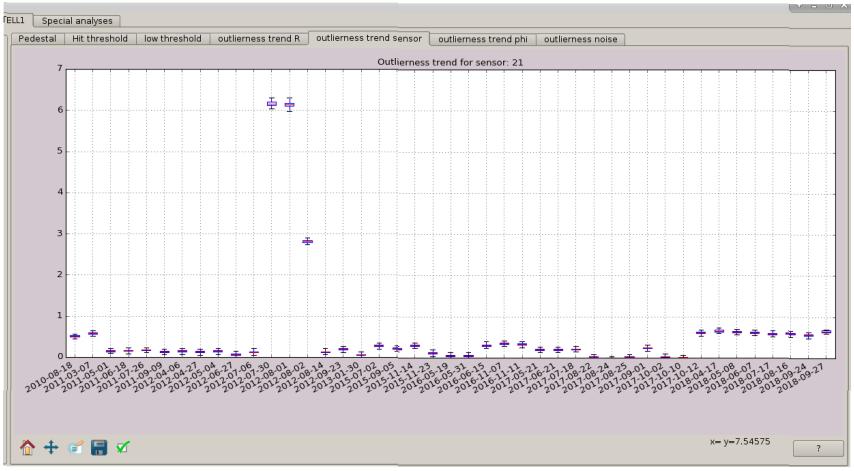


Figure 4.2.2: Screenshot from the outlierness monitoring in Lovell software.

generated dataset Fig. 4.2.1b. Additionally in Fig. 4.2.1c 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 introduced to the Lovell VELO monitoring system in the second half of the year 2018, and was used to monitor the calibration data. The Figure 4.2.2 depicts a screenshot from the monitoring system and shows outlierness levels for sensor # 21. It is noticeable that the value of the outlierness was slightly elevated in 2017, which is attributed to some changes in the cooling system. It is worth noticing that this algorithm was the first Machine Learning based model specifically trained for purpose of supporting the monitoring tasks in the LHCb experiment.

4.3 Dimensionality reduction

The Velo detector, in its strip version, had approximately 172 000 readout channels (that corresponded to physical strips). Many of the parameters needed for its calibration were calculated per channel. This is a huge amount of numbers, which is not easily understandable for human operators. In its pixel version, the number of such parameters will grow to approximately 41 million (exactly $256 * 256 * 12 * 52 = 40894464$)**?** . Thus, an early test of dimensionality reduction techniques will be very useful for the future. A very popular class of models targeting specifically dimensionality reduction is based on PCA (Principal Component Analysis) or on a particular deep network architecture called the autoencoder. There are multiple ways of applying PCA or autoencoder to a given dataset (using different sets of values - or dimensions). This Chapter presents selected applications of dimensionality reduction applied to the VELO calibration dataset.

4.3.1 Pedestals dimensionality reduction

One potential benefit of dimensionality reduction would be to see what differentiates the pedestal parameter in some sensors from the others. For this purpose, we used PCA reduction by reducing each channel's time progression (a single dimensionality reduction for each of the sensor's channels). This

dimensionality reduction can be understood as a mapping:

$$P(n, R, \#, D) \rightarrow X_{n,\#}, X_{n,\#} \in \mathbb{R}^2 \tag{4.5}$$

This means that $X_{n,\#}$ is a 2 dimensional vector, representing an n-th channel in $\#$ sensor. Because of the high number of individual sensors, the split into several plots gives more clarity (Fig. 4.3.1, **??** and 4.3.3, 4.3.4).

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. Examining the particular channels can explain what happens with a single datapoint with the transformation. In Fig. 4.3.5 there are three selected channels from a plot of the reduced data in sensor 11. Those points are also plotted at the Fig. 4.3.6. We can see that the channels on the opposing edges of this circle-like structure formed by the reduced data represent different trends in the pedestals' value. Channel's 1899 pedestal value grows over time, but 543's decreases. Values in channel 322 stay high and stable. Therefore we can say that essentially the PCA reduction separates the channels based on their trend. The 2-dimensional Gaussian distribution structure makes it consistent with the trend calculation in previous sections, as the distribution centre around 0. This means that there is no overall trend (no growth or decrease).

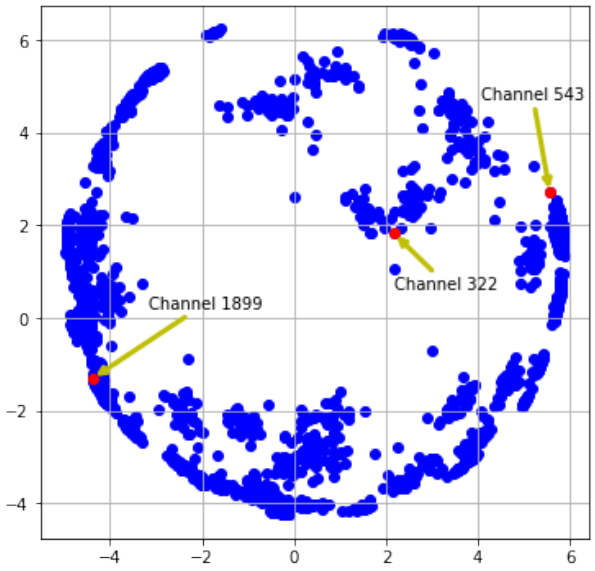
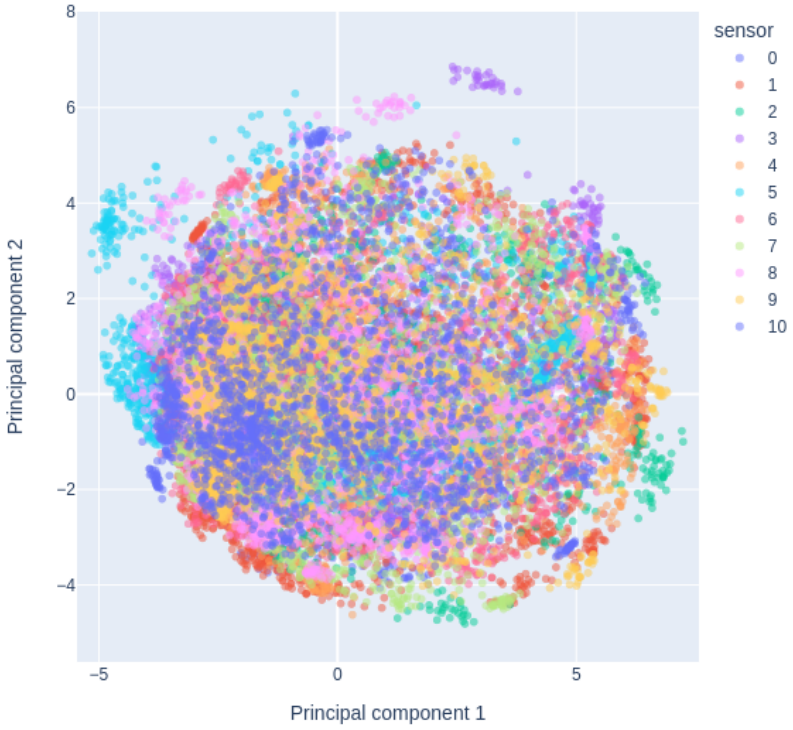
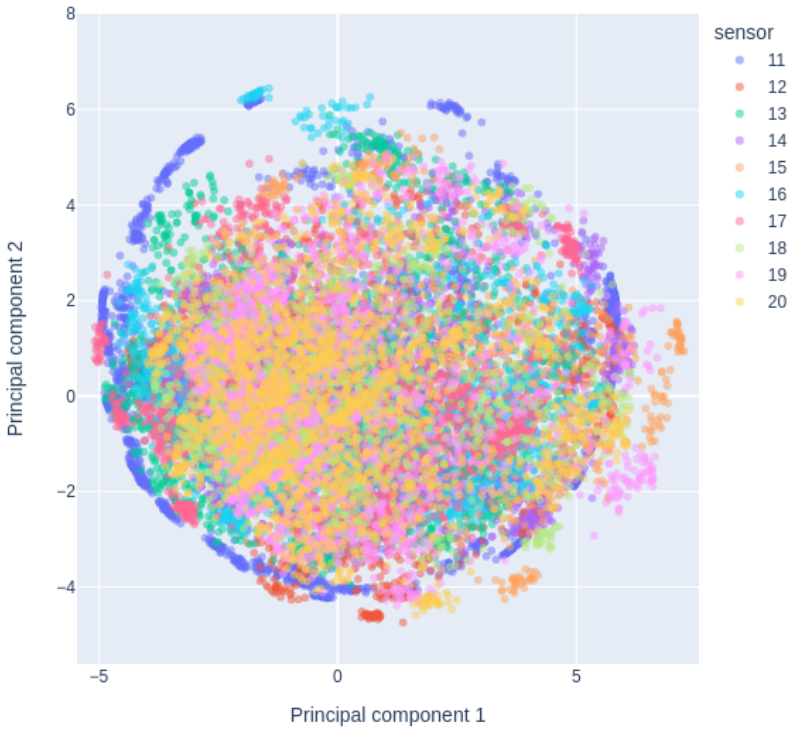


Figure 4.3.5: A 2-dimensional representation of the PCA-reduced dataset for sensor 11, with three distinct exemplary channels marked.

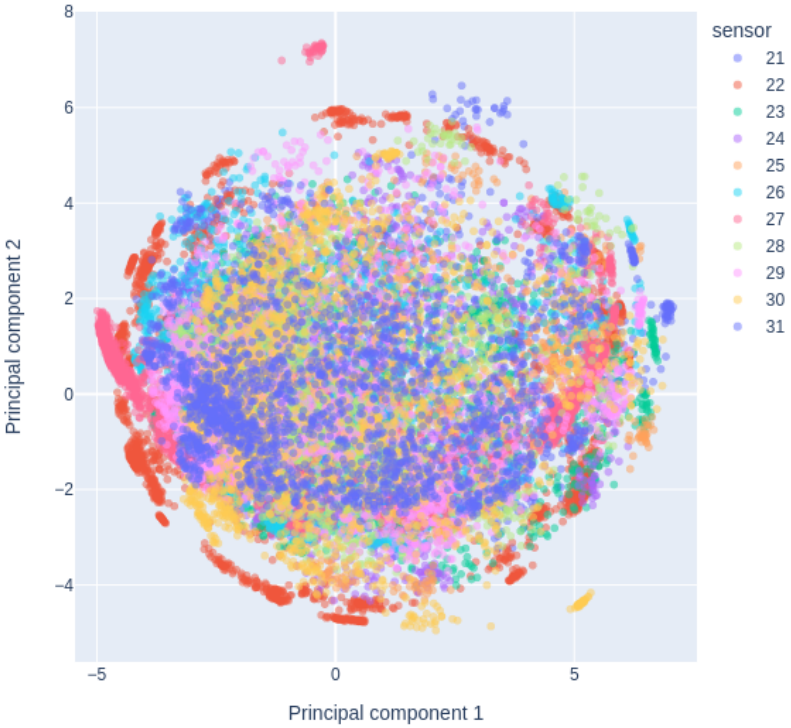


(a) PCA, R-type sensors 0-10

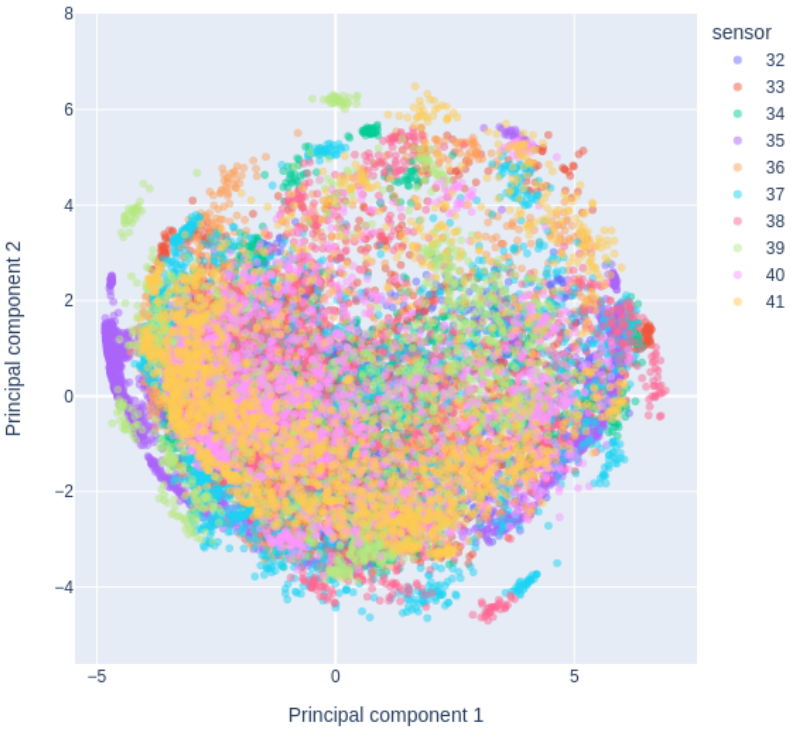


(b) PCA, R-type sensors 11-20

Figure 4.3.1: Pedestal calibrations in R-type sensors (0-20) reduced with PCA.

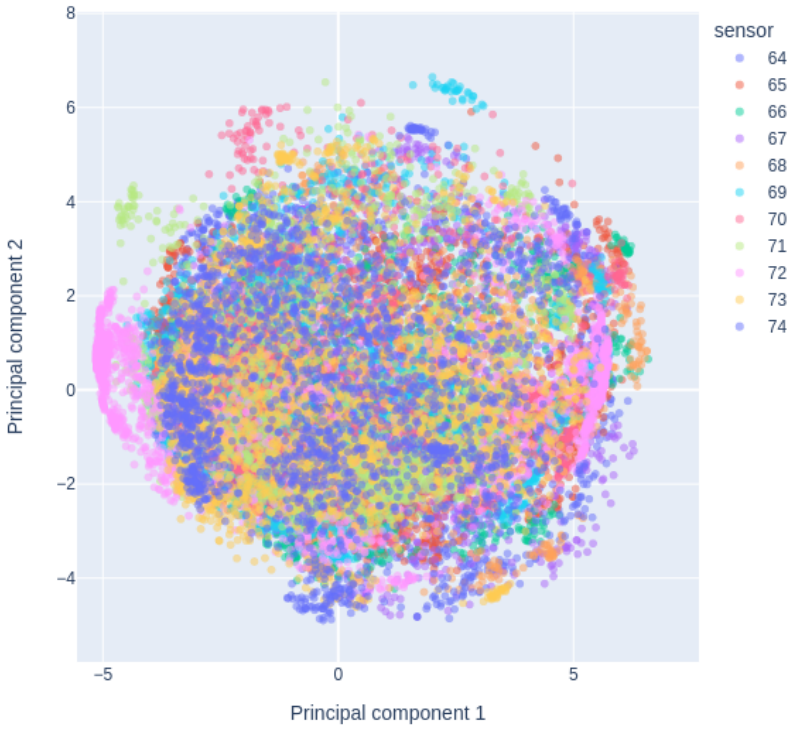


(a) PCA, R-type sensors 21-31

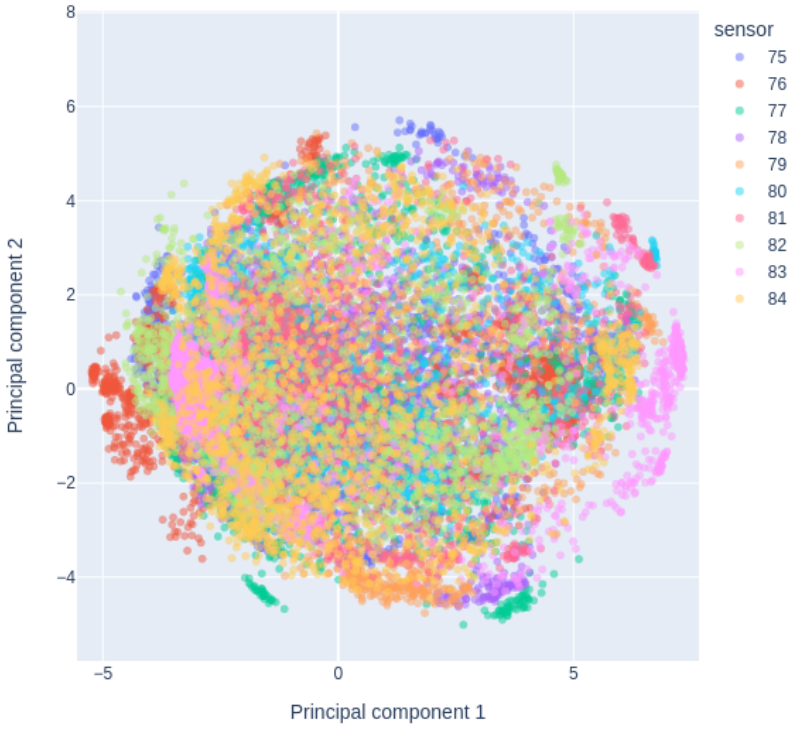


(b) PCA, R-type sensors 32-41

Figure 4.3.2: Pedestal calibrations in R-type sensors (21-41) reduced with PCA.

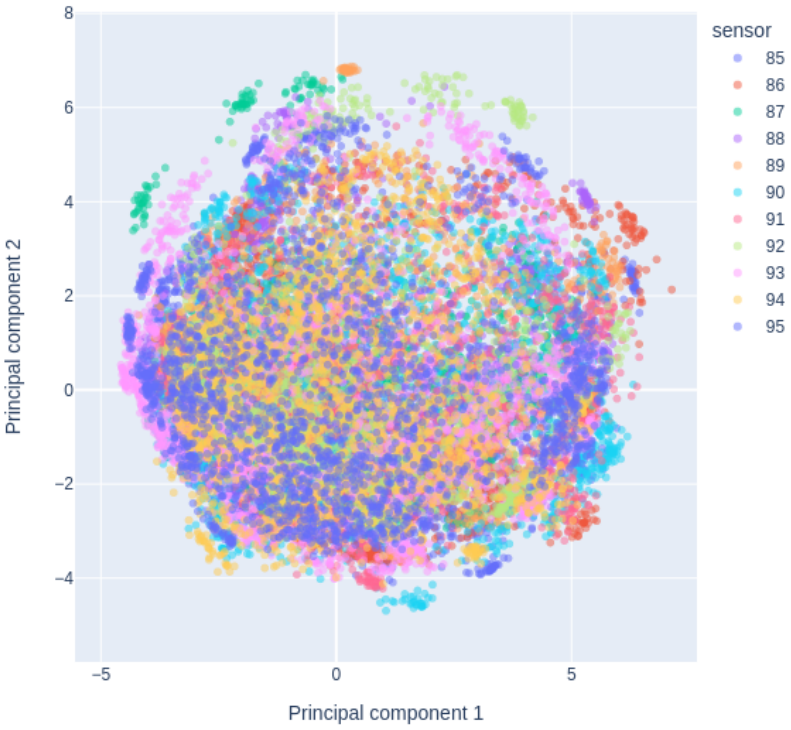


(a) PCA, Φ-type sensors 64-74

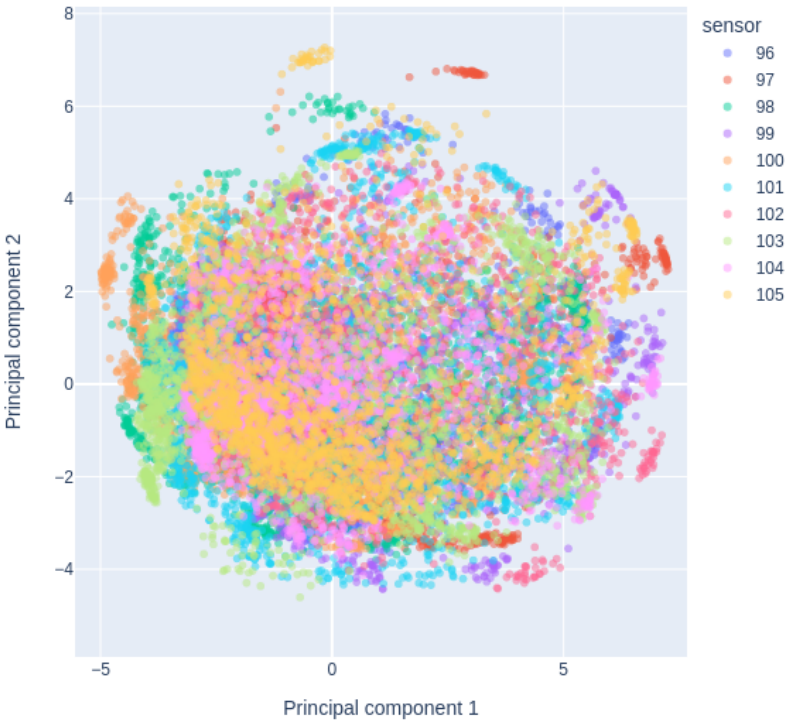


(b) PCA, Φ-type sensors 75-84

Figure 4.3.3: Pedestal calibrations for Φ-type sensors (64-84) reduced with PCA.



(a) PCA, Φ-type sensors 85-95



(b) PCA, Φ-type sensors 96-105

Figure 4.3.4: Pedestal calibrations for Φ-type sensors (85-105) reduced with PCA.

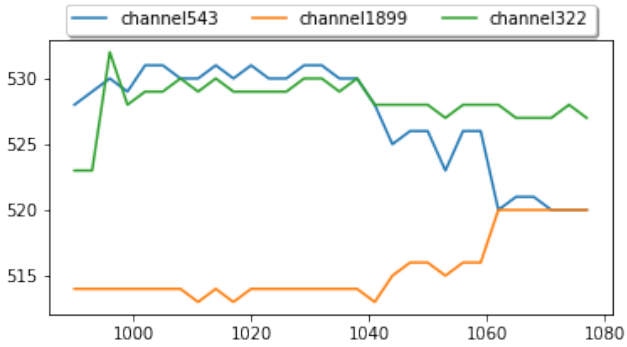


Figure 4.3.6: Time progression of the pedestal value for the selected values.

4.3.2 Threshold dimensionality reduction

Because threshold data for the R and Φ-type sensors differ significantly, it's necessary to split the dimensionality reduction into two cases respectively. That means training the chosen algorithm for the $H_t(Ch*, R, s, d)$ and $H_t(Ch*, \Phi, s, d)$ separately. This transformation can be written as Eq. 4.6 and 4.7.

$$H_t(Ch*, R, \#, d) \rightarrow X_{\#,d}, X_{\#,d} \in \mathbb{R}^2 \tag{4.6}$$

$$H_t(Ch*, \Phi, \#, d) \rightarrow X_{\#,d}, X_{\#,d} \in \mathbb{R}^2 \tag{4.7}$$

The two separate models reduce the dimension of the channels into two dimenstions. The $X_{\#,d}$ means that for each sensor $\#$ on given calibration date d there is a dimensional vector X. This form of reduction allows us to easily see any changes common to all of the sensors at once. For studying the high threshold parameters two models, PCA and autoencoder artificial neural network, were used. In Figures 4.3.7-4.3.10 high thresholds for ten consecutive calibrations are plotted after the dimensionality reduction. In this case the high thresholds measured at one sensor with dimensionality $n_{dim} = 2048$, corresponding to all physical readout channels, are reduced to a single point on a 2-dimensional $(n_{dim} = 2)$ plane. Respective sensors are colour coded for convenience. Regardless of the used model, PCA or autoencoder, one can clearly see two outlying calibrations 2012-07-30 and 2012-08-01. In contrast with an 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. This may, in some cases, make the PCA based reduction more robust.

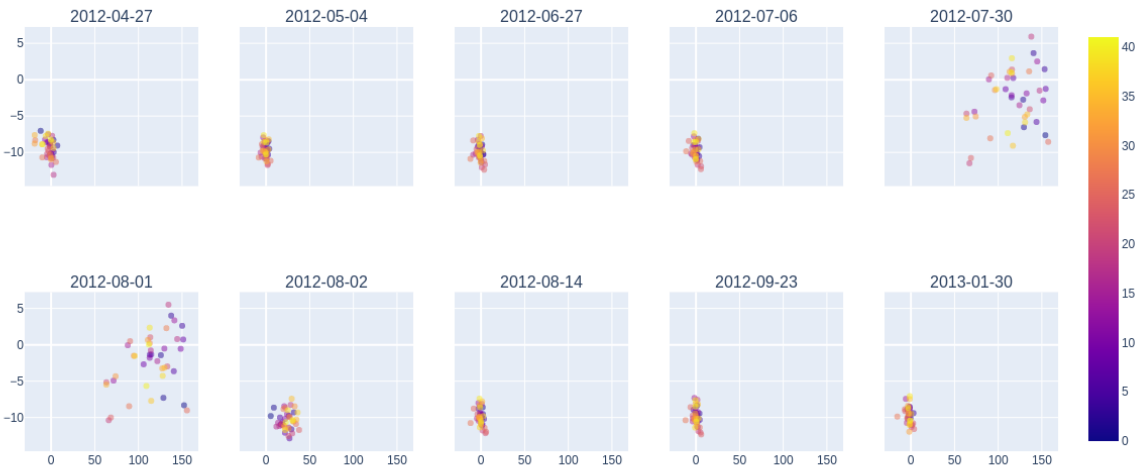


Figure 4.3.7: Time progression of the selected section of calibration dates, with reduced dimentionality using PCA, only for R-type sensors.

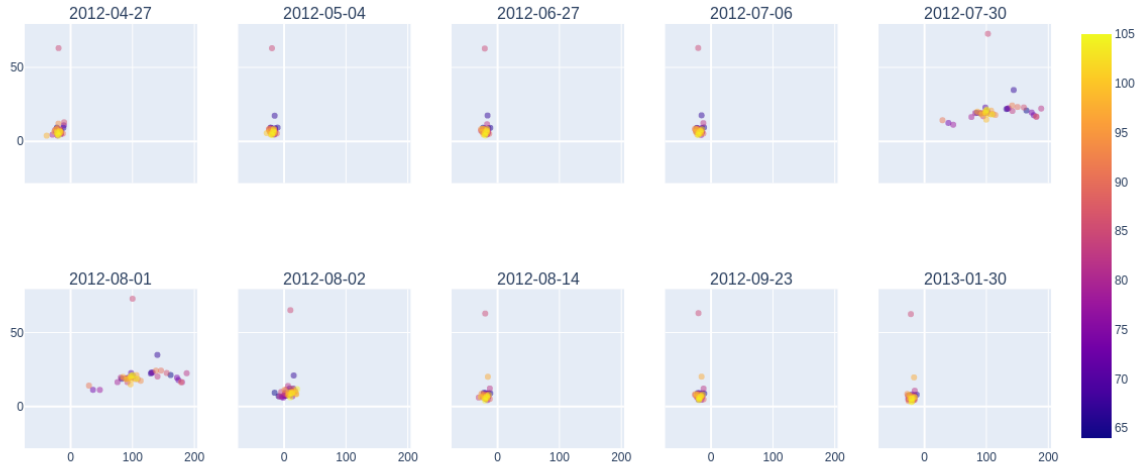


Figure 4.3.8: Time progression of the selected section of calibration dates, with reduced dimentionality using PCA, only for Φ-type sensors.

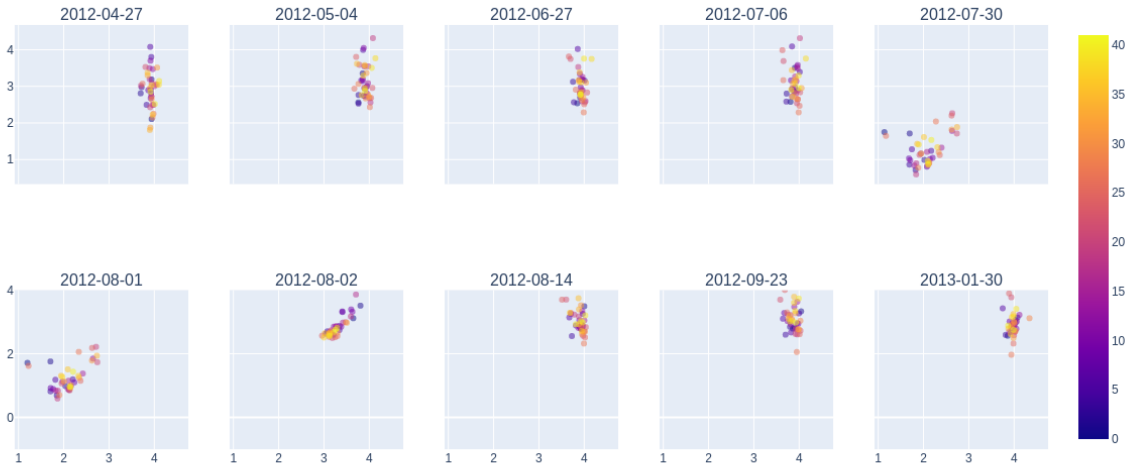


Figure 4.3.9: Time progression of the selected section of calibration dates, with reduced dimentionality using an autoencoder, only for *R*-type sensors.

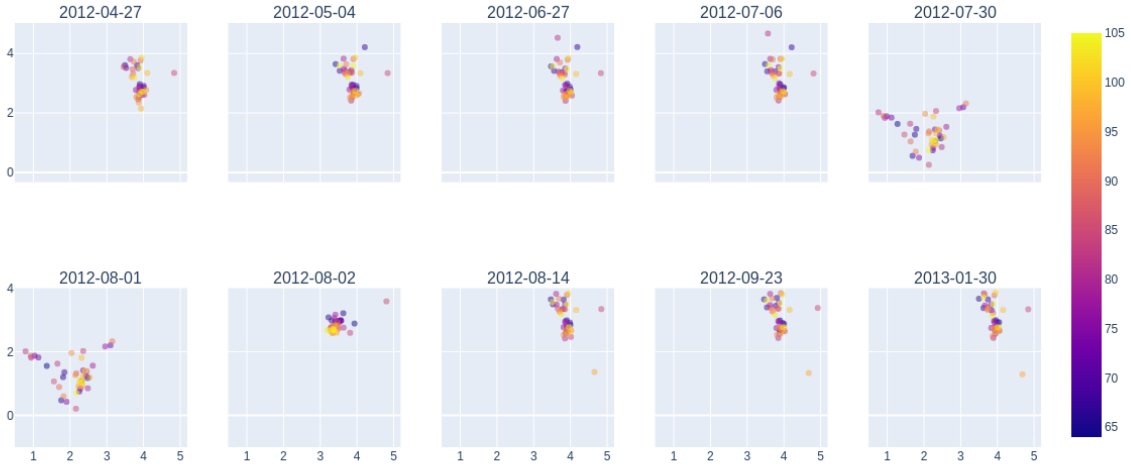


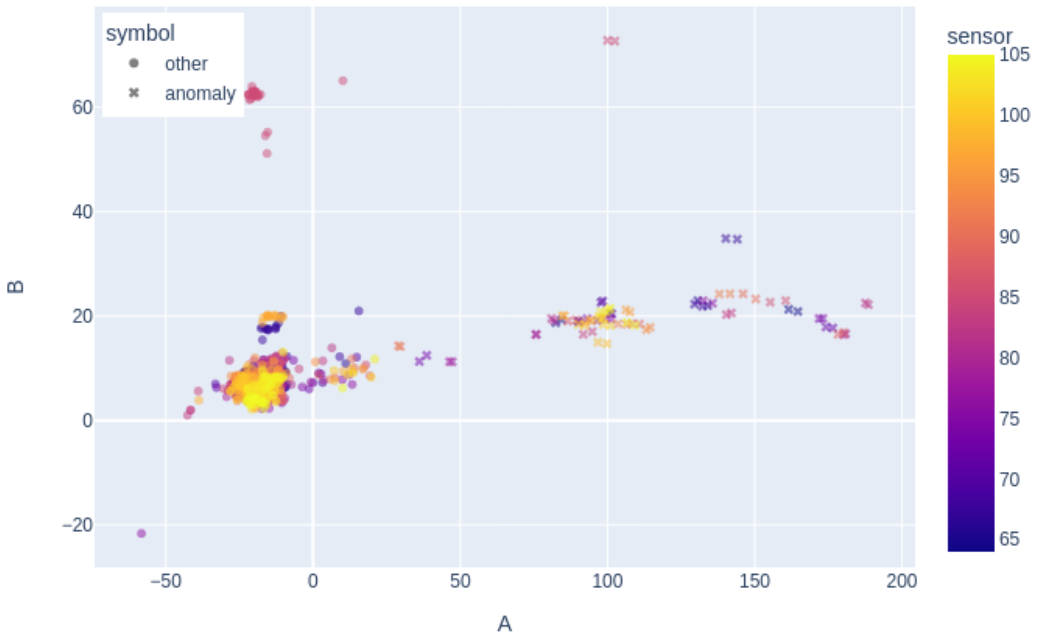
Figure 4.3.10: Time progression of the selected section of calibration dates, with reduced dimentionality using an autoencoder, only for phi sensors.

The Figures 4.3.11a-4.3.12b represent all calibration dates on the same plot, split into different sensors, and created with different techniques. Notice that all of the plots have different axes scales, as the methods used to reduce dimensionality do not retain the scale. The most helpful insight to those plots is that the relative change represents outlying calibrations.

4.4 Time to the next calibration forecasting

The calibration procedure requires a special LHCb trigger activity and had to be always planned with great deal of care. This special data taking task was performed during no-beam time slots and due to

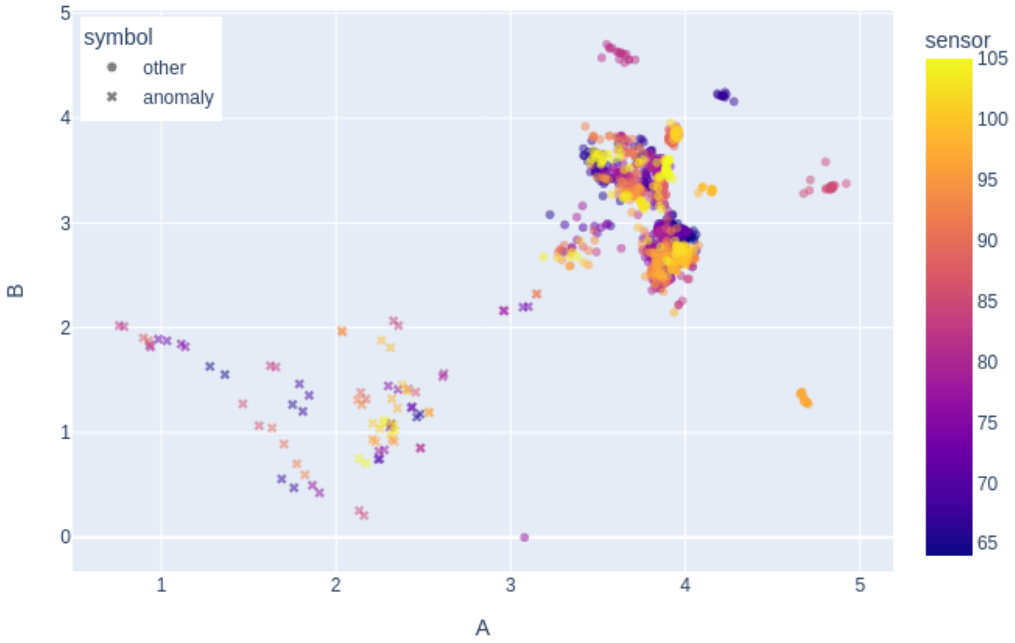
(a) PCA, R sensor type



(b) PCA, Φ sensor type

Figure 4.3.11: All of the calibrations, with reduced dimentionality using PCA.

(a) autoencoder, R sensor type



(b) autoencoder, Φ sensor type

Figure 4.3.12: All of the calibrations, with reduced dimentionality using autoencoder.

the complicated Velo system at least one expert must have been present. Also, the RAW data necessary for the calibration are of completely different nature than the regular physical data taken during proton-proton collisions. A typical event size from the entire detector amounted to a few kB during Run 1 and 2, whilst, a single RAW data event from Velo (no suppression, signal is written out from each detector channel) amounted approximately to 190 MB. That large amount of data sent through the trigger system put an enormous stress on it. Also, the significant volume of data challenged each time the disk storage system. Due to its complexity the calibration procedure was run approximately once per month. The data was then processed and the benchmark calibration parameters were evaluated and uploaded to the memory of Tell1 boards.

All these inconveniences related to the calibration data taking motivated a study on an intelligent method of predicting the necessity of the calibration based on the detector's current condition. The studies performed with the machine leaning approach are described in Section.

4.4.1 Dataset

The calibration data stream contains, in principle, two related types of data: RAW data and setup parameters (also called calibration parameters). The latter are extracted by a set of algorithms, identical for all calibration dates, from the RAW data. The RAW data contains, in turn, signals registered on each readout channel during no-beam time slot.

The statistical analysis of the calibration parameters' trends, described previously, has indicated a candidate for a key quantity that could be used to identify a need for the next calibration. Since, the pedestals turned out to be critical for the quality of the data produced by the vertex detector the analysis focused on them. It was observed, that the pedestal subtracted data slowly drifted, as a function of time, to the point where the data quality was no longer acceptable. During the regular operation a 1-dimensional histogram, also called the summary pedestal histogram, with pedestal subtracted values evaluated on each channel, was used as an observable related to the data quality. Formally, one can define the $\Delta\mu$ as:

$$\Delta\mu = \mu_{current} - \mu_{calibration} \tag{4.8}$$

Where $\mu_{current}$ is the mean of all pedestal subtracted data coming from the current run and $\mu_{calibration}$ all pedestal subtracted data evaluated after the calibration. Since, the calibration should produce the flat baseline (i.e., the mean value measured at each channel should be centred at 0 ADC), the 1-dimensional summary histogram after the calibration was expected to be approximately normal with the mean value equal to 0 ADC counts and variance close to 0.5 ADC counts. Now, one can analyse the time evolution of the summary histograms produced for RAW data taken at later times* with respect to a given calibration parameters. The main premise being, that the drifting pedestals should affect the summary histogram in a specific way. The study of the $\Delta\mu$ was not conclusive, which is compatible with previous analysis that showed no clear trend in pedestal values over the time. However, the analysis of the variance, $\sigma(\Delta\mu)$, showed a clear trend in time. An example of this is presented in Fig. 4.4.1a. These results

*RAW data samples were also taken during regular physics runs for the monitoring purposes. Usually, for a long run a few thousands of RAW events were taken.

show that the width of subsequent summary plots getting larger, thus, $\sigma(\Delta\mu)$ steadily grows in time.

The data used for the forecasting model comes from 2018. The respective calibration RAW data sets collected in 2018 show the best quality and in this year the calibrations were performed relatively frequently comparing to the previous years (e.g., in 2017 only two calibrations were performed). The more frequent calibrations are related with the radiation damage effects that started to be very pronounced in 2018. The exact time span of the data is from 2018-05-08 to 2018-09-24. This constitutes exactly four calibration periods. The used data comes in the form of $\sigma(\Delta\mu)_n$ where t stands for a given run time. The time information is translated to delta time $(dt_n = t_n - t_{n-1})$. Both of the vectors are used to craete an input X_n.

$$X_n = \begin{bmatrix} \sigma(\Delta\mu)_n \\ dt_n \end{bmatrix} \tag{4.9}$$

The continuous-time series of data is transformed to windowed time series of 100 data points, beginning with the first data point after calibration and padded with zeroes. The padding here means that the the $\sigma(\Delta\mu)$ vector calculated at the first input looks like this: $[0, 0, 0, \ldots, 0, 0, \sigma_0]$ and the next one $[0, 0, 0, \ldots, 0, \sigma_0, \sigma_1]$ etc. The analogical padding was used for the dt_t.

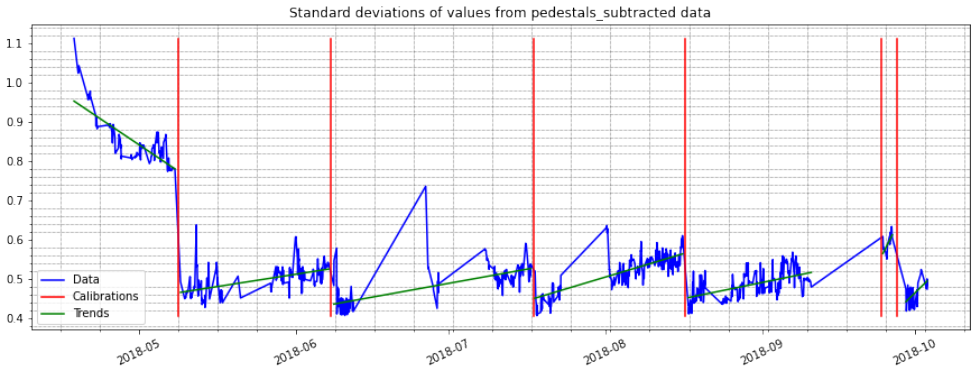
As it is a case of supervised learning, the additional component Y_n is set to be a number of days left until the next calibration. We have chosen the number of days as the most suitable time range for this purpose, as there can be many runs per day. In practice, it means that the input to the neural network is a vector with 42 values + 1 value for the time difference. The used model is a version of a recurrent neural network, which means that the temporal dimension of the data can be modulated. In this case, the model is trained using 100 vectors long sequence.

4.4.2 WTTE-RNN model for Velo

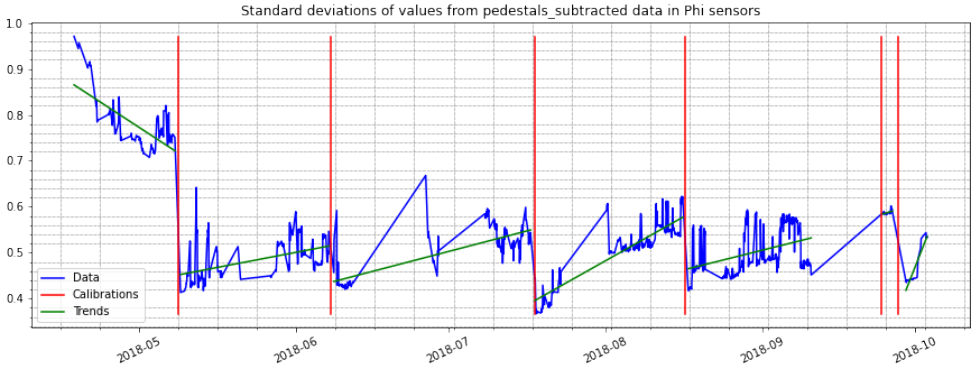
The WTTE-RNN, as discussed in Sec. 3.4.3 is capable of outputting the parameters of the Weibull distribution. The actual type of problem, related to analysis of the time ordered events, in machine learning is called survival analysis. The architecture of the neural network used for forecasting is presented in Table 4.4.1. It contains six layers in total. The LSTM layer is necessary for the time series, that is used as input data, in order to retain the knowledge on time relation between the analysed events.

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

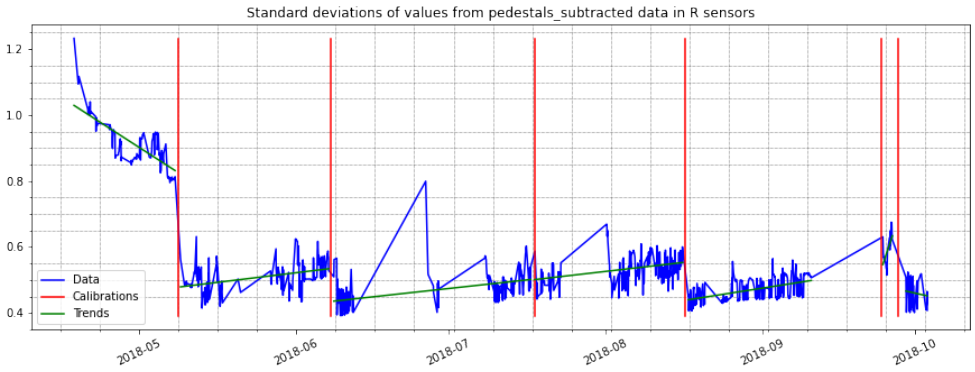
Table 4.4.1: The neural network layers used for the WTTE-RNN model



(a)

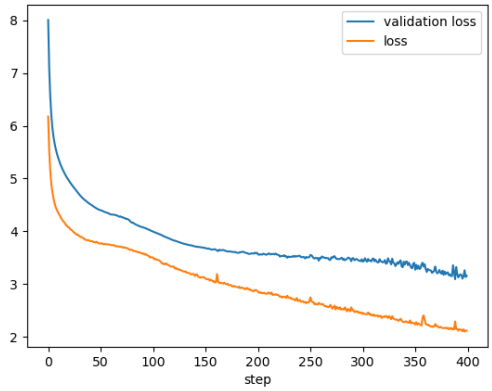


(b)

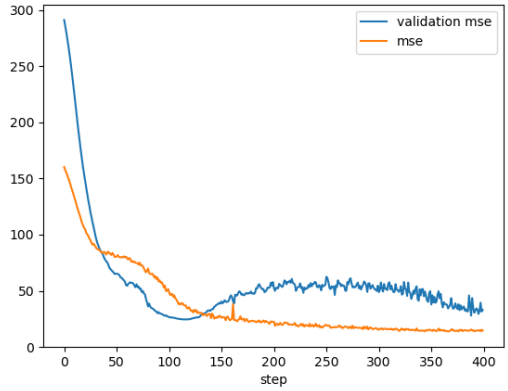


(c)

Figure 4.4.1: $\sigma(\Delta\mu)$ plotted for the whole detector (top), Φ-type sensors (middle), and in R-type sensors (bottom). Dates of the benchmark calibrations are marked as red vertical lines for convenience, the green lines correspond to linear trends calculated between calibrations.



(a) The loss metrics during the training of the WTTE-RNN.



(b) The mean square error metrics during the training of the WTTE-RNN.

Figure 4.4.2: The training process of the model.

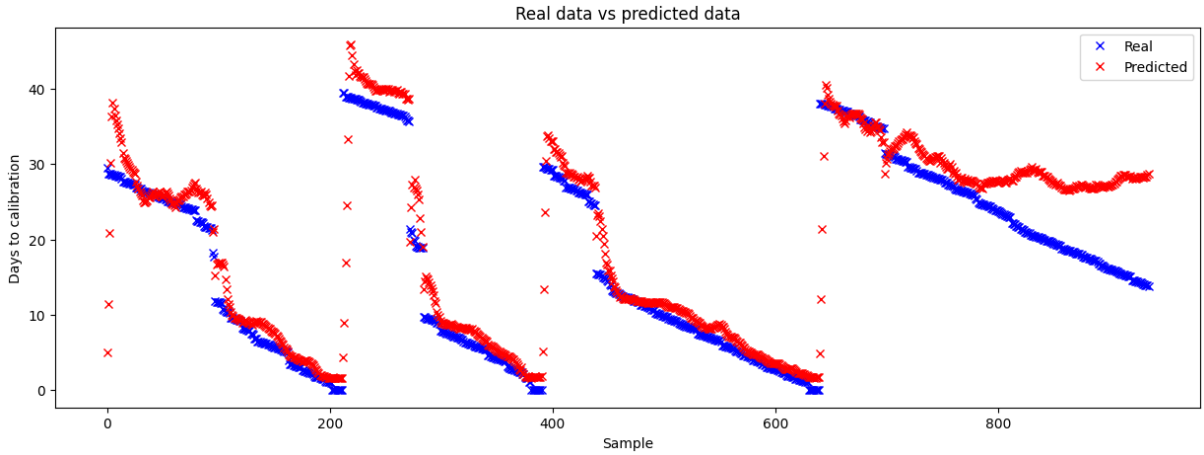


Figure 4.4.3: Real time to calibration (blue points) and the predicted by the trained WTTE-RNN model (red points)

4.4.3 Training and results

We used Adam optimiser with 400 epochs in the training process, with the Weibull log-like discrete loss. The batch size was equal to 50.

The process of the network training is depicted in Fig. 4.4.2. You can see that both loss metrics and the additional MSE is decreasing. For testing purposes, it is helpful to compare the output to the real data. In Fig. 4.4.3 you can see a comparison of the time left to the calibration with the time predicted from our model. The model is incorrect about its predictions at the beginning, right after calibration, due to a low number of available data points, because the model has less knowledge about current callibration, since it is a new calibration. It is also visible that the discrepancy between the ground truth and a prediction comes from the large gaps in the subsequent pedestal subtracted data coming from the in the last weeks of the data taking. The results shown here are auspicious, but ideally should be confirmed on a larger set of data in the upcoming runs of the LHCb detector. Because this method relies on the pedestal information, it could be easily adapted to other detectors.

This concludes the studies for the strip Velo.

The analysis in Section 4.1 proves that the Velo maintainers mitigated the radiation damage. The bias voltage introduced to the detector has kept the overall trend of the pedestals close to zero. The noisy channels have been systematically turned off when exhibiting improper behaviours. However, the total number of masked channels was increasing, which is consistent with the expectation of the effect of the ionising radiation.

This analysis also shows that the imperfect calibration process influences the thresholds. We have shown that it can leverage that influence to create a probabilistic programming algorithm for anomaly detection in the calibration.

We have shown that the PCA and autoencoders create a valid candidate for the dimensionality reduction tasks in the space of the calibration parameters for the strip Velo.

The studies of the influence of the calibration on the detector signal have shown that the difference between the calibration pedestal and the mean noise of the signal coming from the channel is growing with time after the calibration and can be used to predict the need for recalibration.

Overall, this research shows promise in the application for the upcoming data taking in Run 3.

Any sufficiently advanced technology is indistinguishable from magic

Arthur C. Clarke

5

Machine Learning methods and analysis for VeloPix

While these studies in Chapter 4 were necessary, they were also the testbed for developing methods that may find use in Run 3 with the upgrade Velo detector. They also rely heavily on the calibration data recorded during the working of the detector. Without the newly upgraded detector working entirely and without the actual data, it is impossible to introduce the same methods before the data taking starts. Nevertheless, using the test-beam data which was used for finding the optimal configuration of VeloPix ASIC and the experience gained in the process, we can continue research before Run 3.

In the field of particle detectors, one should always be wary of the unexpected. The radiation damage or just pure unluck can lead to various scenarios. One of them is a sensor that either introduces too high an amount of variance or behaves unexpectedly. Similarly to the strip Velo, the VeloPix has mechanisms to silence such a sensor, which renders the detector "blind" in a given sensor. However, even when the noisy sensors are silenced, another malicious effect still exists, which can harm the reconstruction of the track of particles. Given an unfortunate grouping (clusters) of the masked pixels in multiple sensors, there can exist a track that the entire detector can be blind to. This exceptional threat to particle track reconstruction is nearly impossible to be found on a detector calibration with about 65 million individual pixels. In Section 5.1 we present a novel tool based on an unsupervised learning algorithm for detecting clusters of masked pixels. We believe this algorithm can be used not only for combatting the particle track reconstruction problem but also broadly for mask cluster tracking in any MediPix family sensor.

The radiation damage in a silicon sensor is a foe most effectively fought off with preparation and awareness. The Velo detector was designed to be retractable from the collision point. This allows for regulation of the amount of particles flowing through the detector, which in another case, could overflow it with data. However, this also makes a compelling case for measuring the fluence of the detector, as

quite literally, it is a moving target. One would think that since the detector is designed for particle track reconstruction, it would allow for easy calculation of the fluence by counting the hits. Sadly it is not the case. The data-taking mode of the VeloPix is focused on selecting the tracks that can be used for physics study purposes and discards all other information. The rate of the particles incoming to the detector promotes the processing speed and forces not to process more data than is necessary. The awareness of the radiation damage always comes only after the fact. We cannot measure it before it happens. Thus, in the VeloPix sensors, there exists a mechanism that is influenced by the damage introduced to the silicon. By measuring the change in the behaviour of that mechanism, we can determine its damage and, in turn, its fluency. The surrogate function relates the ToT signal from the sensor to the charge gathered. The process of calculating the parameters of the surrogate function is de-facto measuring the response of the silicon to the charge deposit. This is the mechanism and the link between the fluence and the detector parameters that we can leverage to our advantage. In the Sec. 5.2 I present the studies of the method for linking the surrogate parameters with the fluence introduced to the sensor.

5.1 Intelligent pixel mask cluster tracking

As discussed in Sec. 2.4.2.1, the VeloPix masks are essential for filtering signals that otherwise would significantly increase the number of produced noise hits. In short, the pixels in the VeloPix sensor that exhibit too large noise are "masked" - meaning that their readout is deactivated. The masked pixels are less problematic when they are evenly distributed across the sensor, but when they cluster together, they can cause a problem for the particle track reconstruction algorithm (Fig. 5.1.2). Simple counting of the masked pixels may not be enough to detect clusters or other structures* that can be formed by the masked pixels. During a regular operation of a properly configured pixel sensor we expect that the distribution of masked channels should be flat and random and should not feature any accumulation points or structures. Due to its critical impact on the data quality the masking procedure is going to be, arguably, the most important for the daily operation of VeloPix detector. Precise, robust and fast analysis of the masked channels will play a vital role and cannot just include a simple counting algorithm but should also offer more sophisticated metrics. For instance, per sensor analysis of the masked channel density distribution as a function of the channel position.

5.1.1 Simulation of the VeloPix masks

Due to the VeloPix being still under development and commissioning during these studies the real calibration data were not available at the time of this analysis. Instead, a dedicated readout system built in the microelectronics lab at AGH-UST was used to produce a fake mask files. Each such file contains 255 x 255 masks that can be uploaded to the memory of VeloPix chip to configure it. The masks can take on two values zeor or one, where zero stands for good channel and one for channel to be masked. The position of the mask is directly mapped to a position of a channel on the sensor. To make a realistic simulation, we have considered the following scenarios:

- Random uniform changes *P*, in which a random portion of pixels changes their state to masked

*The worst case scenario is a problem with an entire column or row of pixels.

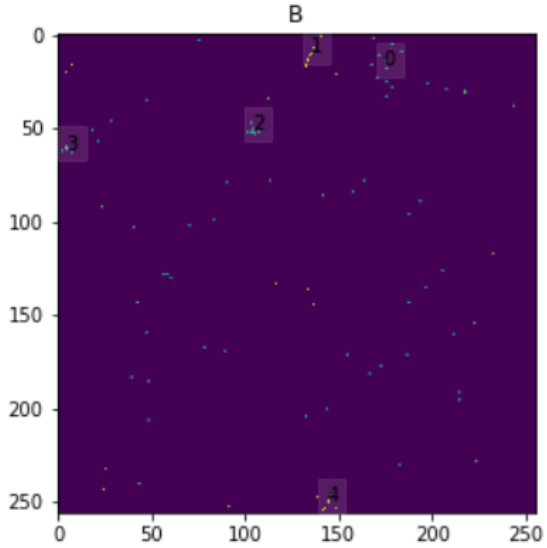


Figure 5.1.1: An example of clusterisation performed using DBSCAN ($\varepsilon = 10$, MinPts $= 4$) on the binary pixel-map. Clusters are numbered 1-5.

- Linear cluster L where a linear portion of pixels are masked.
- Blob cluster G with masks created by a 2D isotropic Gaussian distribution.

These scenarios were created to reproduce some effects observed during the preliminary lab tests of the Velo pixel sensors. These intrusions are added to an initially empty mask file with a different probabilities. Each of the intrusions is created and added to an empty mask matrix to form a single simulated matrix A. One crucial step of this simulation is a random cleaning of the mask matrix, where a random fraction of the bad pixels are set back to a no-masked state. The pseudocode, describing the fake mask files generation, can be found in Listing 3. This process creates a continuous simulation of VeloPix mask matrices, as the new masks and clusters are generated and the old ones slowly disappear. In this way we could simulate multiple sensors and multiple calibration runs.

Data: $n \geq 0$
Result: $y = x^n$
$N \leftarrow n$;
while $N \neq 0$ **do**
  if $random() < blob\_prob$ **then**
    $A \leftarrow A + G$
  else if $random() < line\_prob$ **then**
    $A \leftarrow A + L$
  else
    $A \leftarrow A + P$
  end
  $A \leftarrow A + Pu$
  $A \leftarrow N - 1$
end

Algorithm 3: Steps of the mask cluster simulation.

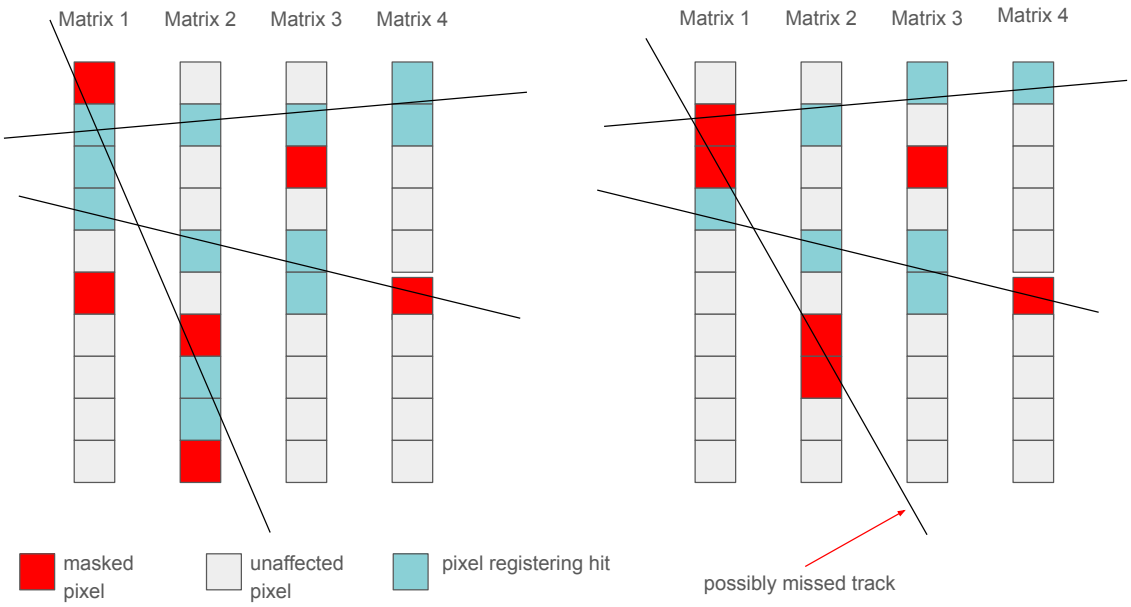


Figure 5.1.2: A visualisation of the problem of the clustered masks. Both images contain a represenataion of multiple aligned sensor matrices pixels, with particle tracks marked with lines. The left image contains uniformly distributed masks, while the right one containse clustered masks. Both cases have the same number of masked pixels.

5.1.2 Mask clustering

It was decided to use machine learning approach for searching for potential ordered structures in mask files. Due to the nature of the data an unsupervised technique was chosen based on clusterisation. There are many clustering algorithms present in the field of machine learning. Although they all come under the term "clustering", there are multiple different goals that various algorithms can achieve. The most common density search clustering algorithms are DBSCAN and OPTICS. The details of these algorithms are described in Section 3.7. In the case of application of the clustering algorithm towards analysis of mask files produced during the calibration, the desired algorithm should be able to differntiate points that belong to a cluster from the outliers. The two algorithms selected for this study fulfill this requirement.

5.1.3 Mask cluster features

After finding mask clusters for a given calibration (a one set of mask files), they 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 characteristic vector as $v_k = [s_k; h_k]$. Then we define a cluster as a set of unique features $cluster_k = p_k, v_k$.

5.1.4 Mask cluster tracking

This set of unique features, defined in the previous Section, characterises every cluster found for a given calibration. The $cluster_k$ features are then used to identify cluster k from timestep t_n as the 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:

$$M_{i,j} = \Phi_{i,j} * V_{i,j} \tag{5.1}$$

Where Φ is defined as:

$$\Phi_{i,j} = \frac{1}{d_{min}} * \max(d_{min} - D_{i,j}, 0) \tag{5.2}$$

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

$$D_{i,j} = ||p_j - p_i|| \tag{5.3}$$

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. The exemplary pairing of the clusters in consecutive simulation steps is visible in the Fig. 5.1.4 with it's similarity matrix calculation in Fig. 5.1.3.

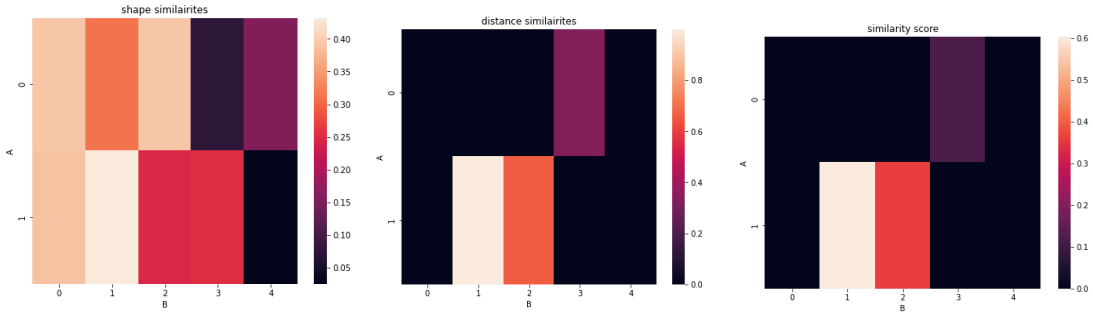


Figure 5.1.3: Rows represent clusters on image A in Figure 5.1.4, columns represent clusters on image B in Figure 5.1.4. Values indicate the Spacial Characteristics Similarity Measure V (left plot), and Positional Similarity Measure Φ (middle plot). The M matrix is the rightmost plot.

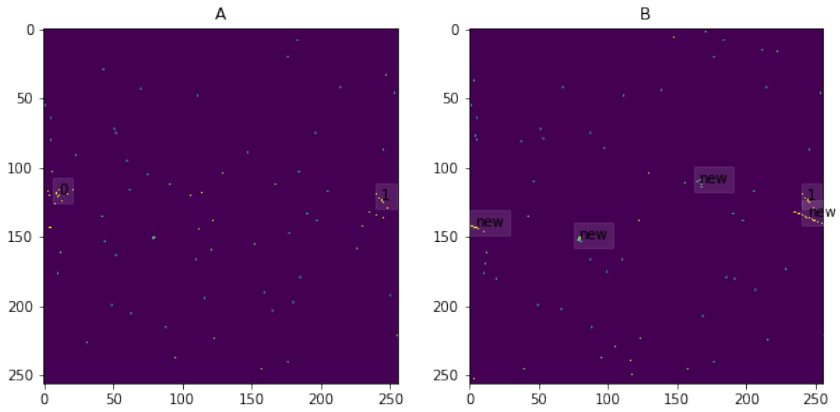


Figure 5.1.4: The algorithm chooses clusters labelled with the same integer as the consecutive generations of the same cluster. Clusters labeled as 'new' are clusters on the time step t_n that were absent on time step t_{n-1}.

5.1.5 Results

The DBSCAN and OPTICS algorithms were tested on 3000 consecutive simulation steps of a single sensor evolution in time. The ground truth in the context of clustering of the masks is not something that can be determined in real-world data. But the simulation, along with the simulation steps, can attribute any generated mask pixel to its source (random uniform, linear, Gaussian blob), thus providing information that can be used as ground truth. It is assumed, in this analysis, that the generated pixel should be identified as belonging to a cluster after up to 8 timesteps. The Table 5.1.1 presents a confusion matrix calculated using that ground truth information. The OPTICS algorithm has three times as many false positives as the DBSCAN. On the other hand, it has almost twice more true positives.

Table 5.1.1: 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 for the ability to track the consecutive mask cluster's through the steps of the simulation. Fig. 5.1.5 depicts the number of the masks associated with any cluster starting at the timestep of the introduction of said masked pixel. It is visible that The OPTICS algorithm recognises more pixels as belonging to clusters and for an extended amount of time, whereas the DBSCAN very quickly drops its attention to the pixels.

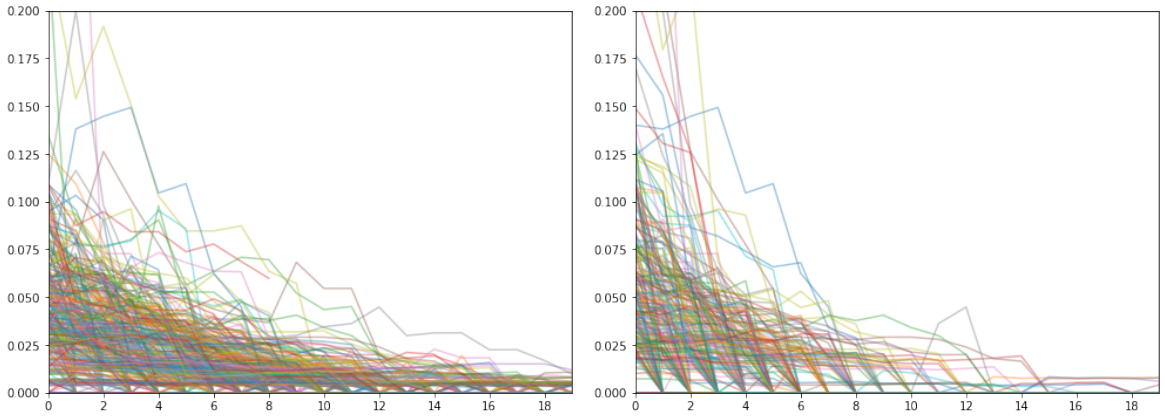


Figure 5.1.5: The fraction of pixels categorised as belonging to any cluster (Y-axis) in the next consecutive calibrations (X-axis), since the cluster introduction to calibration (number of timesteps $n = 300$). The number of detected masked pixels slowly decreases with time. The OPTICS algorithm (left plot) recognises the masked pixels of the clusters as belonging to a cluster (not necessarily the same one) for a more extended amount of time. The DBSCAN is more strict in distinguishing the masked pixels that belong to clusters.

The important test is the influence of the clustering algorithm on the cluster tracking ability. The Fig. 5.1.6 shows a total number of mask clusters in a given simulation timestep, recognised as new or coming from the previous timestep. It is clear that the cluster tracking with OPTICS detects much more clusters as new, and loses more clusters from calibration to calibration, whereas the DBSCAN is finding fewer new clusters, and retains old ones.

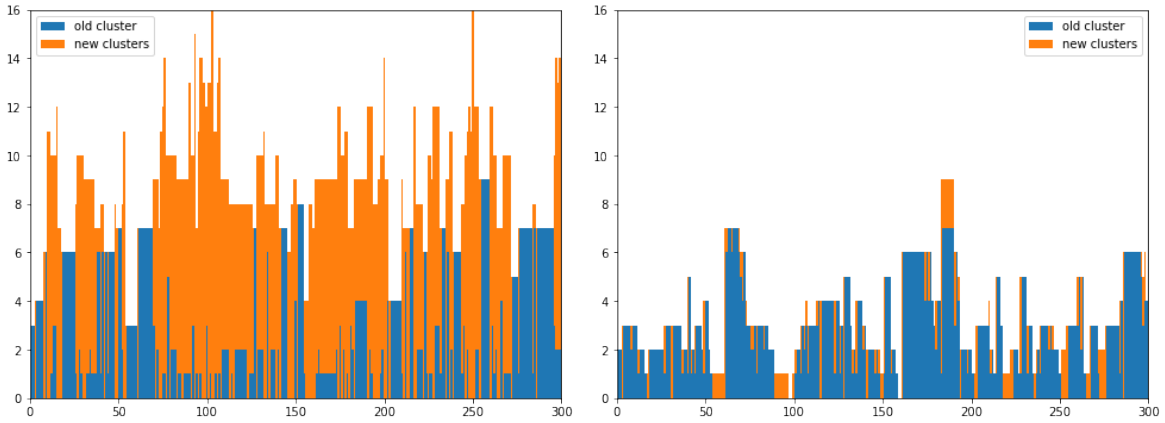


Figure 5.1.6: The number of clusters classified as being "old" - same as in previous calibration (blue colour), and a number of clusters marked as "new" - not being the same clusters as in previous calibration (orange colour). The left plot belongs to OPTICS, and the one on the right to DBSCAN.

5.2 Studies of the surrogate activation functions measured with the irradiated VeloPix sensors

The LHCb vertex locator upgrade led to a major change in sensor technology to provide a much larger channel density and radiation hardness. Both of these features targeted the need to cope with five times higher particle fluence after the upgrade and could only be realised by using silicon planar pixel sensors.

The initial design assumed that the new pixel sensors would be read out in analogue mode. However, simulation studies showed that it would lead to an unmanageable high data stream. Thus, a few years into the project, the decision was made to support only the binary readout for the upgrade Velo detector. Still, there is a dedicated data channel, called ECS (Experiment Control System) link, that allows to send out extended digital data at a low rate. This is essential for taking calibration data and understanding the radiation damage effects and measurement of noise. The information about the deposited energy is measured using the TOT**?** (Time-Over-Threshold) technique. So, instead of measuring voltage pulse amplitudes, the readout electronics measure the time window when the voltage pulse is higher than a set analogue threshold (see Fig. 5.2.1). One should stress that the TOT is, by definition, a digital signal. In order to properly get the proper relation between the TOT counts and the deposited energy, each pixel should be calibrated via measurement of the so-called surrogate or activation function (see Section 5.2.1). With the ECS data link it will still be possible to take the extended data and measure the charge collection efficiency for the irradiated VELO pixel sensors. The technique that will be used for VELO is the measurement of the MPV (Most Probable Value) value of the Landau deposition curves. Such studies have been performed during dedicated test beam campaigns to evaluate if the new pixel sensors will efficiently operate at the end of the Run 4. One idea, that can possibly bring a significant breakthrough for irradiation studies, pertains to fluence estimation based on the properties of the surrogate function and it's relation to the fluence. This would bring a novel method for measuring the fluence for VeloPix, and possibly for other sensors of MediPix family.

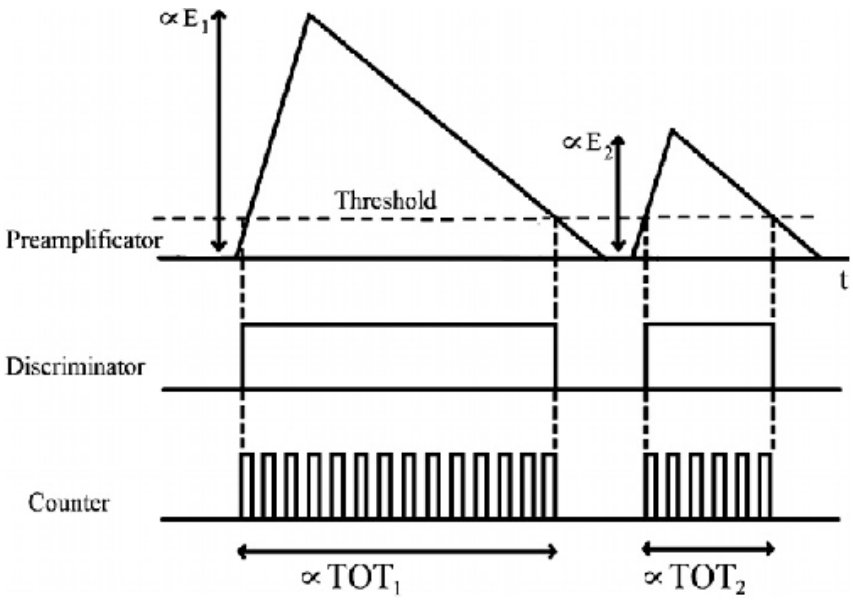


Figure 5.2.1: A schematic of the TOT working mode.

5.2.1 Surrogate function model

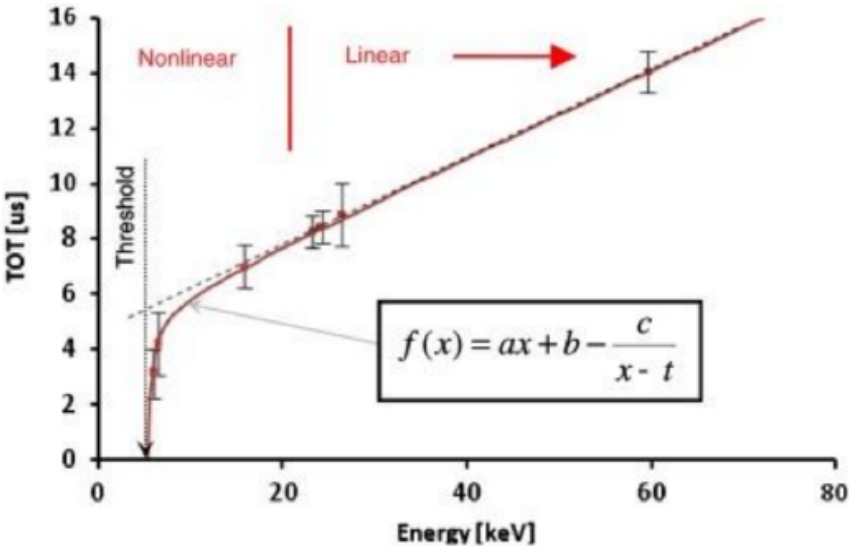
The surrogate function is the analytical model that relates the TOT counts to the generated charge, that in turn, is proportional to the deposited energy**?** . In essence, the surrogate function is required for the proper energy calibration of each individual pixel. It is predicted that the surrogate curves may change significantly during the operation time of the VELO detector due to severe radiation damage. Under-

standing and quantifying such changes is a top priority for predicting the detector behaviour after irradiation. We also show that using the properties of the surrogate functions one can attempt to evaluate the fluence.

The surrogate function can be approximately modelled using the following analitical formula:

$$ToT(q) = p_0 + p_1\dot{q} - \frac{c}{q-t} \tag{5.4}$$

The formula depends on four parameters: p_0, p_1, c, t, and is essentially a combination of linear and hyperbolic functions. There is an underlying assumption that $q > 0$. An exemplary surrogate function is presented in Fig. 5.2.2. Starting from the lower values, the surrogate function's hyperbolic part is most dominant. Later, the linear component of the function is most dominant as the deposited energy increases.



Dependence-of-the-time-over-threshold-signal-measured-by-a-single-Timepix-pixel-on.png

Figure 5.2.2: An exemplary plot of the surrogate function. Source: [?] .

5.2.2 Experimental data

For the source of the data in our studies, we turn to the dataset collected during one of the test beam [?] experiments conducted while developing the LHCb Velo upgrade detector. Particulary, we are interested in the case where the irradiation of the sensor was performed non-uniformly. That is, that within the sensor different areas have different ammount of total fluence. This allows to link the parameters of the surrogate (which are calculated on per-pixe basis) with different fluence levels. The test beam setup used a Timepix3 telescope device [?] that allows for a very precise charged particles tracking with a track pointing resolution, at the centre of the telescope, close to 2 μm. The device was also able to provide a timestamp for each track with a resolution of 1 ns.

5.2.3 Cross sensor study

The test beam data contained multiple different types of sensors with different irradiation schemes. Unfortunately, in the entire dataset, only one sensor (designated as S8) is suitable for studying the surrogate activation curves in function of particle fluence. This is due to the irradiation profile of the sensors and available raw data. Sensor S8 was the only one with a clear, non-uniform profile of the irradiation. For easier analysis and better statistics, we group the pixels into bins of simmilar level of fluence. The S8 sensor was the only one that allowed for a binning of the sensor based on the amount of irradiation that the sensors were exposed to. The S8 is a 200 μm thick, Hamamatsu n-on-p planar silicon pixel sensor that is virtually identical to the sensors used in the upgrade Velo detector construction. During the test beam experiment it was connected to the Timepix3 readout chip. Although other tested sensors featured uniform irradiation profile, nonetheless, we also investigated their activation curves. The main strategy of the analysis was to study the properties of the surrogate function model's parameters distributions before and after the irradiation. The respective distributions are presented in Figs. 5.2.3 and 5.2.4. The important feature, from the perspective of our analysis, is that for most of the sensors, the distributions of $p0$ and $p1$ parameters shifted to the left after the irradiation. This shift shows that those parameters are sensitive to the fluence. For the parameters c and t the trend is not as evident.

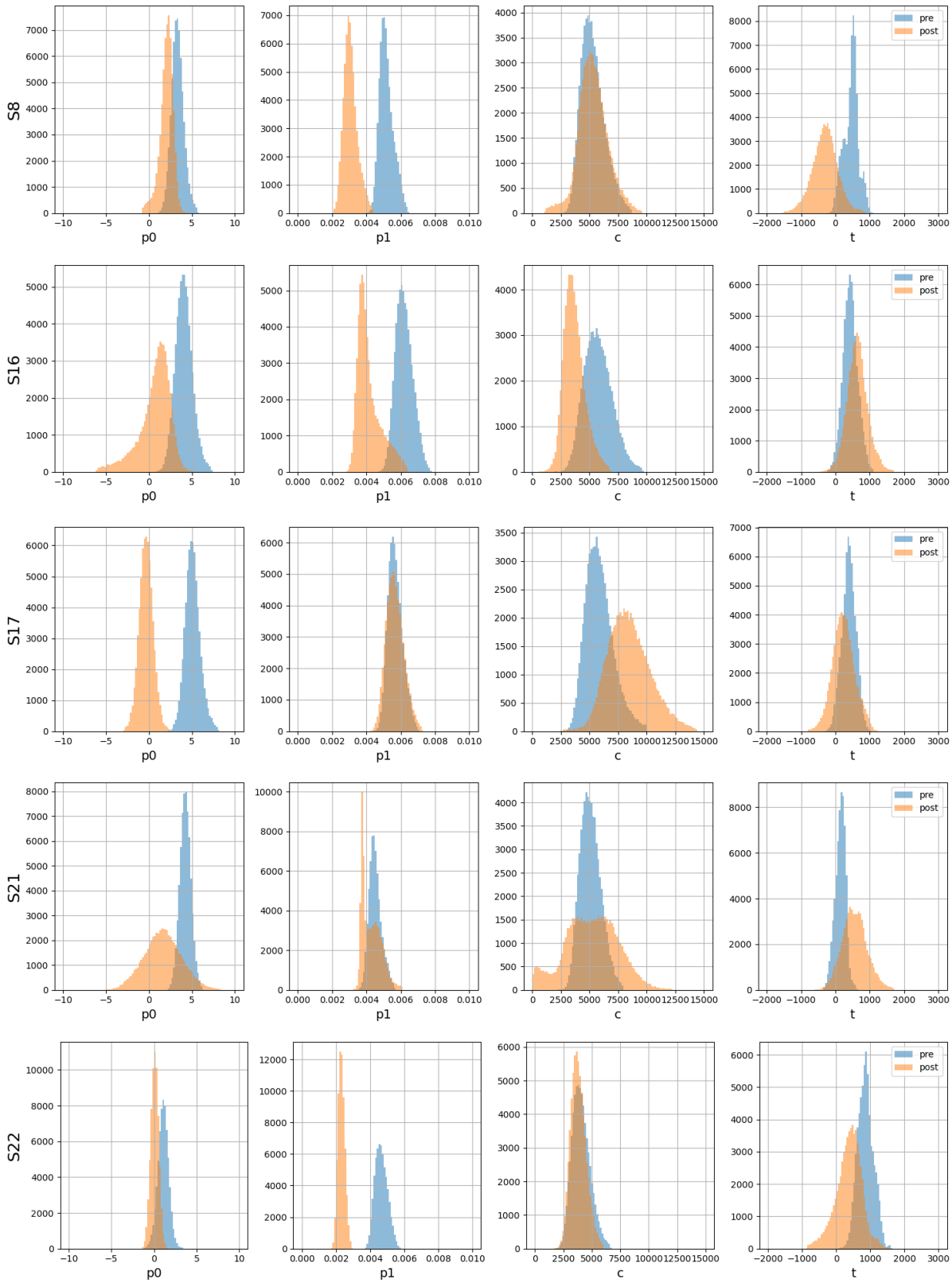


Figure 5.2.3: Distributions of the surrogate model's parameters measured using sensors S8, S16, S17, S21 and S22. The rows of plots represent respective sensors, each of the columns of the plots represents a parameter of the surrogate model (p0, p1, c, t). The colour blue (pre) denotes the distribution before irradiation of a given parameter; the orange colour is the distribution after the irradiation of the sensor.

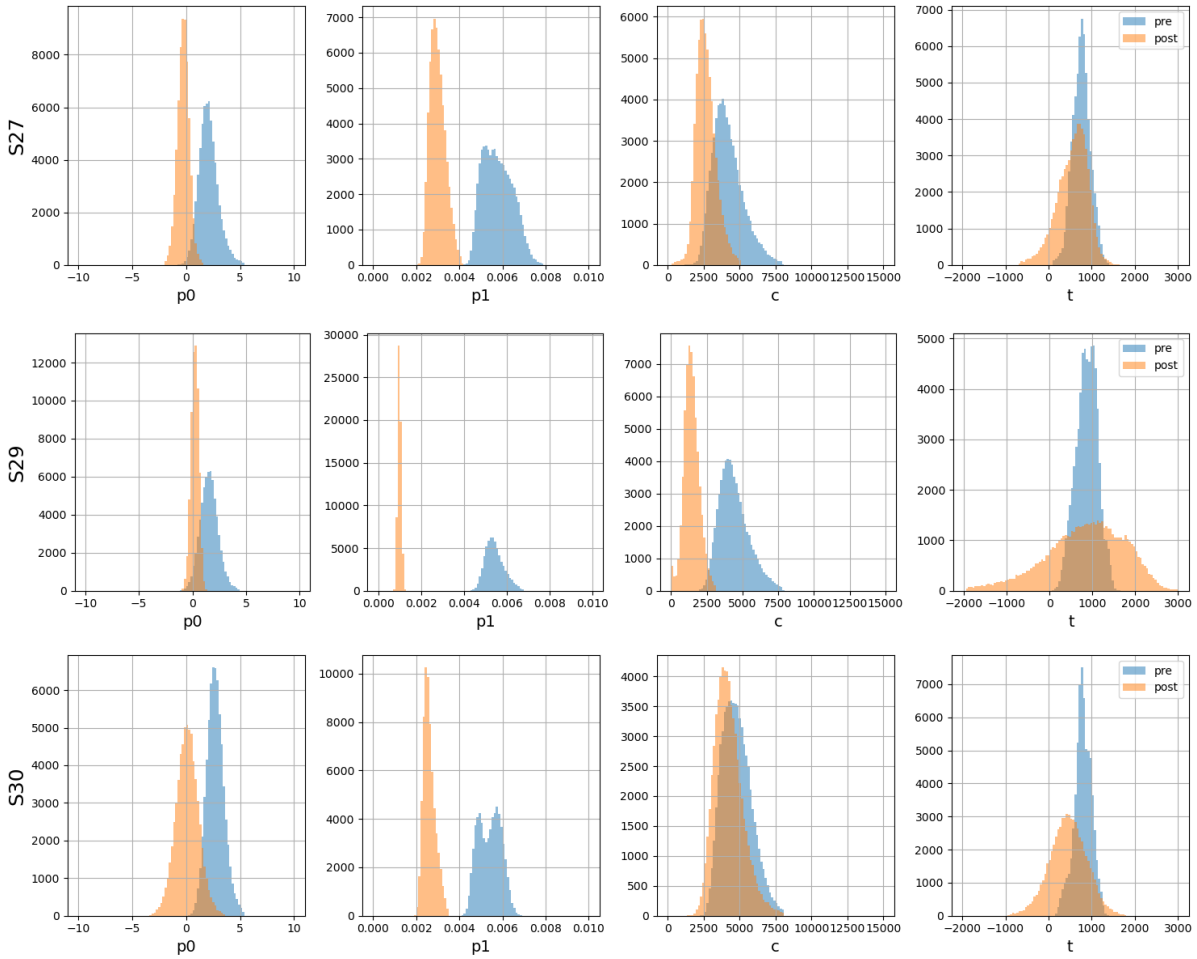


Figure 5.2.4: Distributions of the surrogate model's parameters measured using sensors S27, S29 and S30. The rows of plots represent respective sensors, each of the columns of the plots represents a parameter of the surrogate model (p0, p1, c, t). The colour blue (pre) denotes the distribution before irradiation of a given parameter; the orange colour is the distribution after the irradiation of the sensor..

5.2.4 The fluence

As mentioned previously, the sensor S8 was the only one with well-documented non-uniform irradiation profile. For the purpose of this studies, we propose an innovative analysis using an adaptive elliptical binning presented in Fig. 5.2.5. This binning allows for assigning fluence values per bin and group the pixels with the same fluence. Such approach not only accommodates in a natural way the actual fluence profile but also allows to evaluate various properties of the sensor, expressed as a function of the fluence, with much greater precision and sensitivity. The measured fluence per bin is as seen in the table 5.2.1, the innermost bin has the highest fluence level. This means that we expect that the effects of the radiation will be most visible in the centre of the studied pixel matrix.

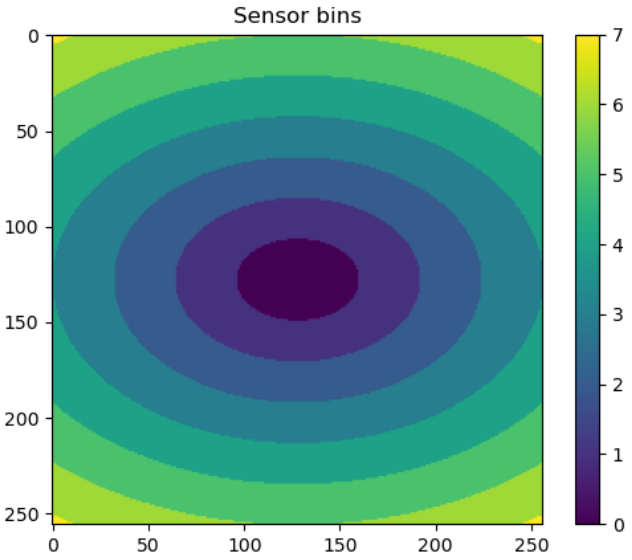


Figure 5.2.5: A heat-map of the binning used in the analysis of the fluence. The binning starts from 0 at the centre, and the outermost bin is labelled as 7. Each colour represents a different elliptical binning.

Bin label	Fluence $[n_{eq}/cm^2 1e^{15}]$	No. pixels
0	7.504	2153
1	7.187	6406
2	6.595	10718
3	5.760	15040
4	4.766	13715
5	3.655	12228
6	2.681	5141
7	1.779	135

Table 5.2.1: Table of fluence per bin

The visible shifts in the surrogate model's parameters $p0$ and $p1$ distributions (with respect to the irradiation) visible across the sensors (Figs. 5.2.3, 5.2.4) are complementary to the shift visible in the sensor S8, when comparing the distributions of these parameters per bin, in the Fig. 5.2.7. The parameters distributions are depicted as boxplots, per fluence bin, pre- and post-irradiation. A detailed explanation of the used boxplot is can be seen in Fig. 5.2.6. One can notice that the $p0$ and $p1$ parameters, measured for the irradiated sensors, exhibit lower values in comparison to the pre-irradiated ones. Interestingly, the trends for both investigated parameters are reversed, the $p0$ increases as a function of fluence, whilst, the $p1$ decreases. The c parameter exhibits a more significant variance with slightly elevated values. T parameter also shows large variance, with a visible overall decrease.

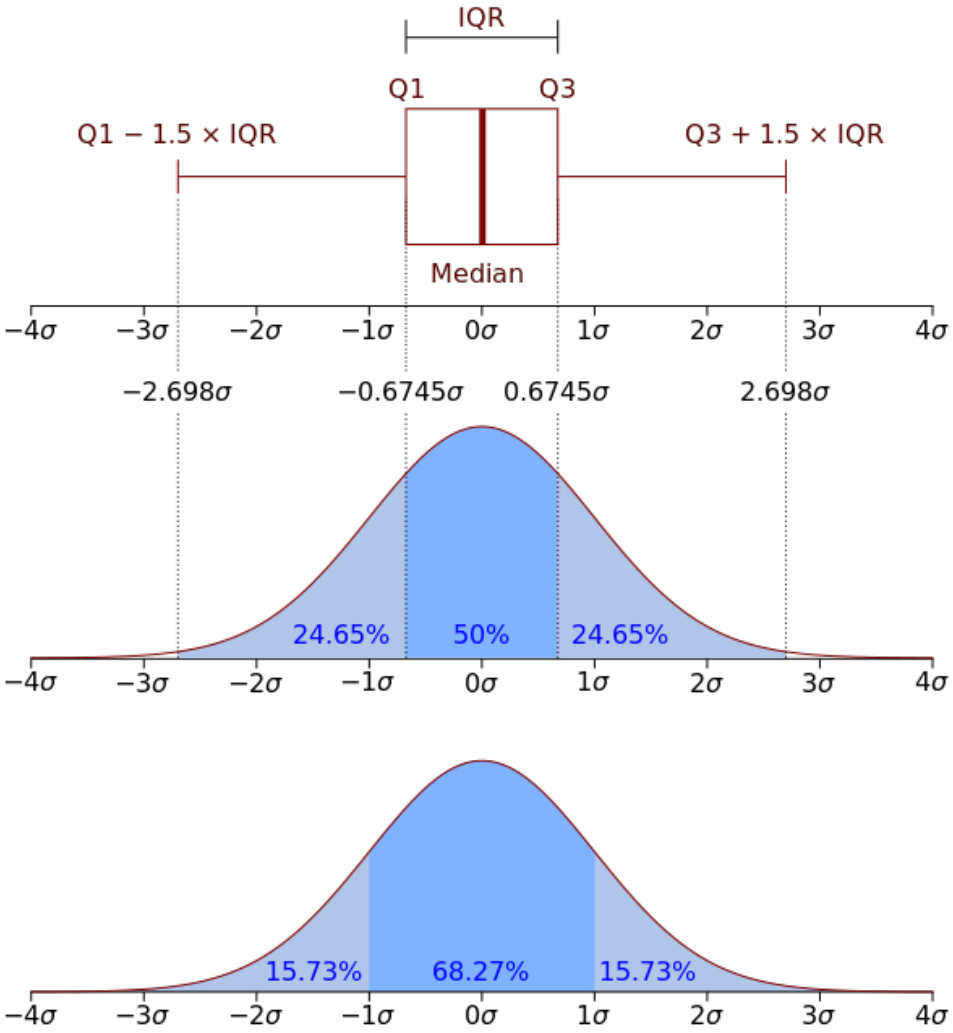


Figure 5.2.6: A plot explaining the scope of the boxplot. The top plot represent the boxplot, with median in the middle. The range of the box spans from Quartile 1 to Quartile 3. The interquartile region (IQR) is the width of the box. The whiskers span from the edges of the box to length of 1.5 IQR to 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.

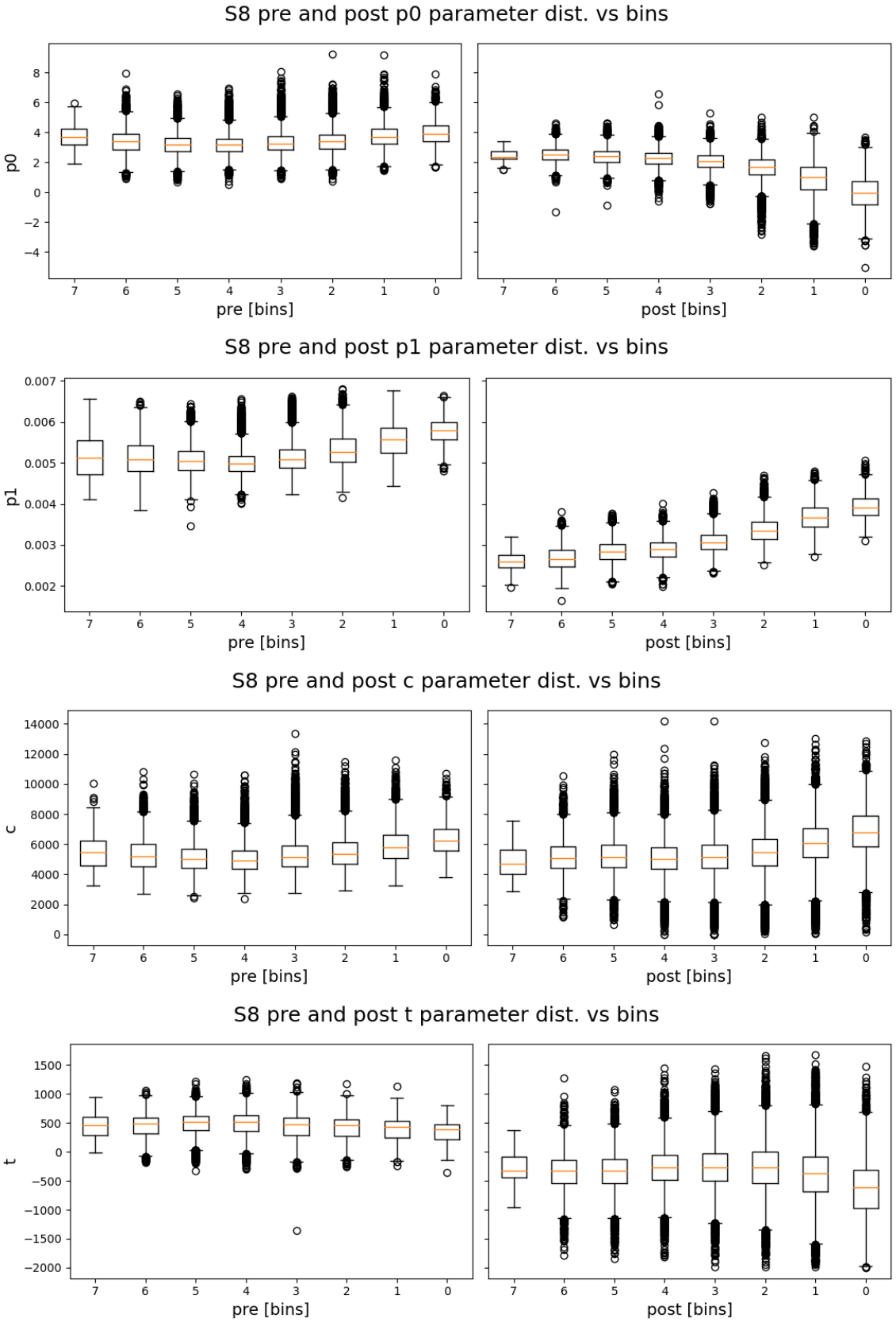


Figure 5.2.7: Surrogate model's parameters measured for sensor S8 plotted as boxplots for the respective fluence regions. Left-hand side plots represent parameters for non-irradiated sensor, whilst, the right-hand side ones after irradiation. Note, that the highest fluence bin, on the post-irradiation plots, is the far-left one.

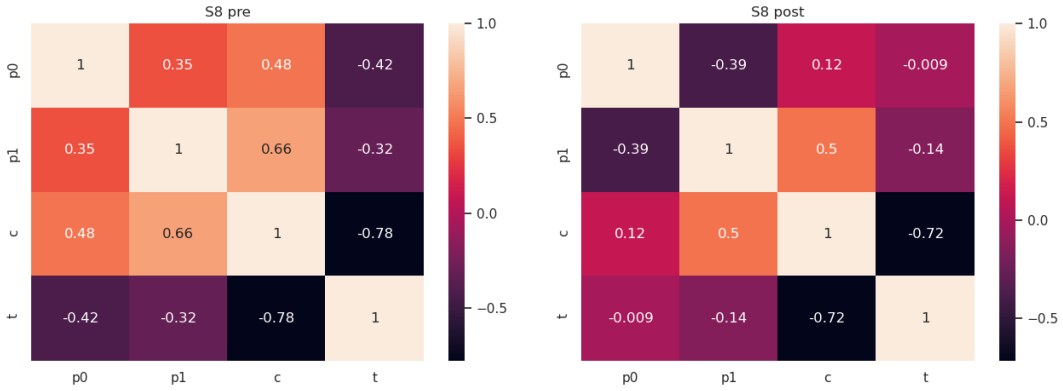


Figure 5.2.8: Pearson correlation coefficients between the surrogate parameters in pre- and post-irradiation cases.

When analysing the parameters, it is important to note their correlation. The Pearson correlation parameters presented in Fig. 5.2.8 show a high correlation between c and t pair, as well as c and $p1$. We do not explore the reasons for such correlation but acknowledge it in further studies.

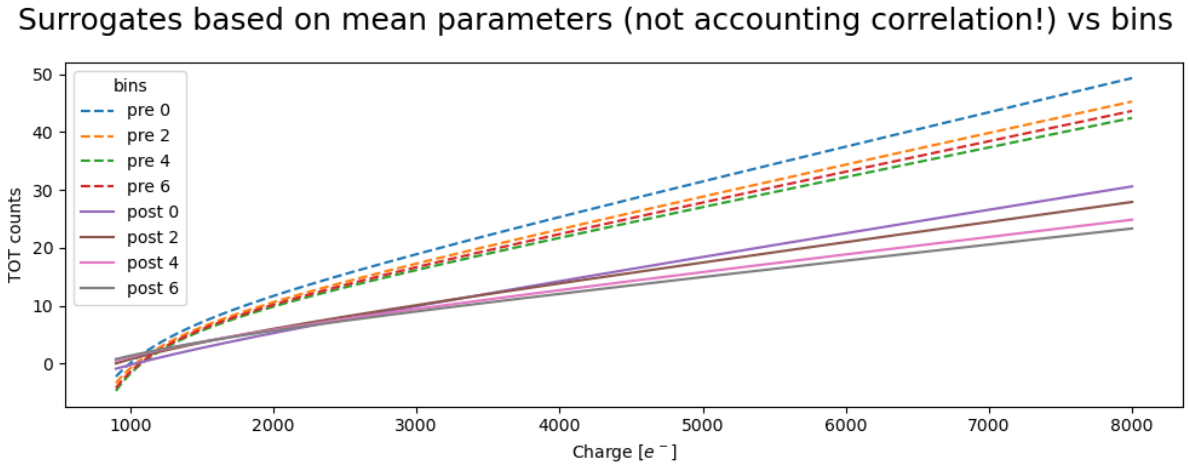


Figure 5.2.9: Average surrogate functions plotted for selected fluence regions. Broken lines shows the measured curves before the irradiation, the solid ones after irradiation.

To better understand the evolution of the surrogate functions, we present some exemplary activation curves, plotted using the mean value for each respective parameter in selected sensor regions corresponding to appropriate fluence bins. These exemplary surrogates, plotted for four sensor regions, showed in Fig. 5.2.9 are just approximations since we are not de-correlating the parameters, just using the mean values. In both cases we observe a similar spread of the curves, which is normal, since there is always a spread in physical properties of individual pixels. Interestingly, the spread seems to be approximately the same before and after the irradiation. The most pronounced difference is a significant change in the surrogates' slope after irradiation. This effect can be interpreted as a decrease in charge collection efficiency after irradiation. This is well reflected in the behaviour of the parameter $p1$ - the slope of the activation curve is decreasing as a function of particle fluence. Also, it is visible, alas less clear, that the charge necessary for pixel activation is higher after irradiation.

Given this change in pixels behaviour after irradiation, we can inverse the problem as follows, by measuring the change in the surrogates' parameters before and after the irradiation, it might be possible to estimate the fluence based solely on the data coming from the sensor. One should stress, that this technique may be a major innovation in the field of the silicon radiation studies. So far only unreliable methods of fluence estimations exists based only on Monte-Carlo modelling. Typically, the uncertainty of such modelling is assumed to be at least 100%

5.2.5 Modelling the surrogate curves evolution in the fluence domain

In order to discover the exact nature of the relation of the surrogate parameters with the fluence, we create a statistical model. As previously shown in Fig. 5.2.3, the four parameters have a distribution that can be aproximated with a gaussian. In case of one free parameter this could be done simply by aproximating it's distribution. With four parameters and partial correlation between them, we need additional steps. The model, used in our studies, for the surrogates evolution as a function of fluence consists of two parts, rescaling and application of the PCA (Principal Component Analysis). In this way we can generate a large set of surrogate curves that will allow to search for potential pattern of the fluence evolution. Because the parameters have different ranges of values, scaling is needed to bring them to the same range. We are using a standard scaler in form of ($z = \frac{x-u}{s}$, where u is mean and s is standard deviation). Next, because the Pearson coefficient indicates that the parameters are correlated, we will use the PCA to decorrelate them. Since it is easy to decorrelate the variables using the PCA, which assumes that the variables follow the normal (Gaussian) distribution, we will assume Gaussian distribution for the activation curves' parameters. The PCA internally fits the dimensions of the variables to a 1D Gaussian, and its transformation matrix is holding the width and mean information. And so the model can be summed in two steps

1. Scaling (z= (x-u)/s)
2. Decorrelation with PCA.

The PCA decorrelation allows then for the sampling of the parameters from the decorrelated space using four random Gaussian generators and the following procedure;

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

This process is useful for checking the loss of the quality of the parameters resulting from assuming that the parameters follow normal distribution. Fig. 5.2.10 shows the results of applying the model of the surrogates measured for the sensor S8, for all pixels pre-irradiation. It is clear that the generated parameters are of sufficient quality for our purposes (for more detailed studies a more sophisticated model, such as an autoencoder, may be used). An additional test for the model is presented in Fig 5.2.11. The plot depicts the combination of the generated parameters into respective surrogate functions, both using the real world data, and the generated output from our statistical model using learned distribution.

In the two previous tests of the model (Fig. 5.2.10 and 5.2.11) we were modelling all pixels of the sensor for either pre-irradiated sensor or post-irradiated sensor without the distinction of fluency. By

separating the sensor into 8 distinctive bin areas, we are able to relate the surrogate parameters to the fluence. For each of the two cases (pre or post-irradiated) and each of the 8 bins, we learn the distribution parameters separately. The Figs 5.2.12 and 5.2.13 depict the results for the full fluence model, split according to respective sensor regions, are presented. The real-world data and the simulation result are overlayed. Although we are showing all parameters (p0, p1, c, t), the p0 and p1 have the most substantial impact on the overall shape of the surrogate function. While in Fig 5.2.12 we can see that the distributions in the consecutive bins for the p0 and p1 do not change, in the Fig. 5.2.13 we can see a noticeable change depending on the bin. This change corresponds to the change in the fluence in the bins. What is also important is that in both cases, the real and generated data overlay quite well, showing that our model reflects the change influence.

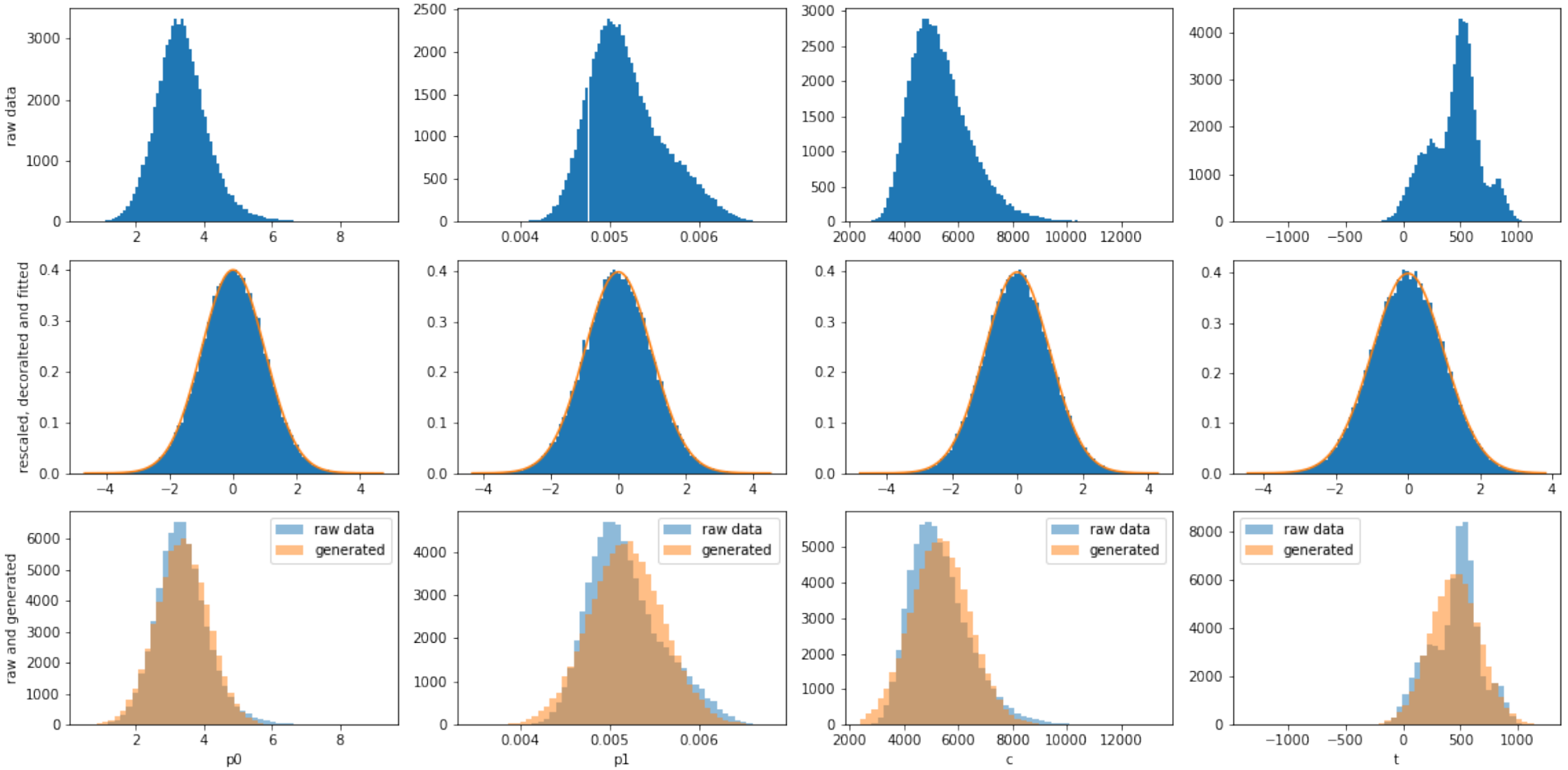
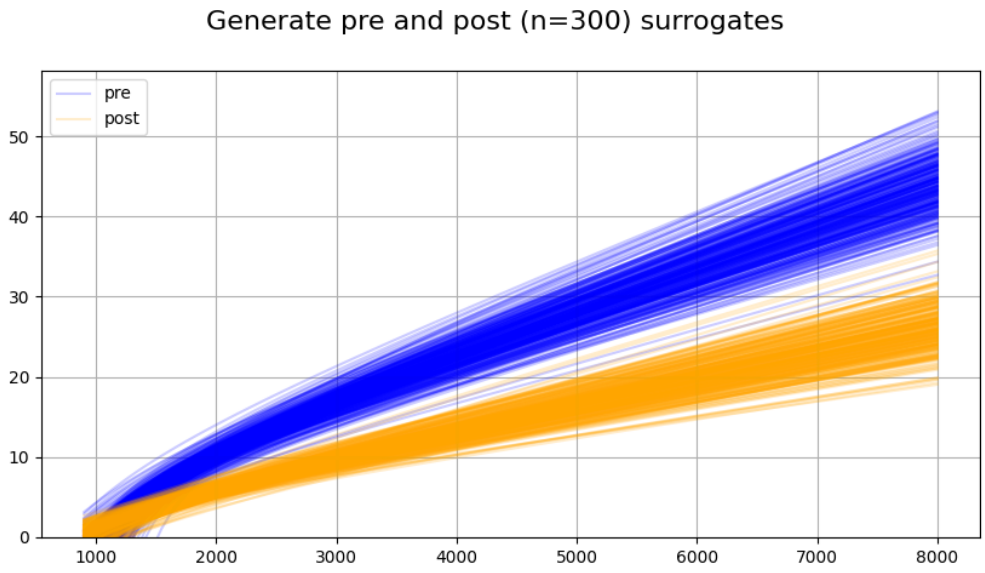
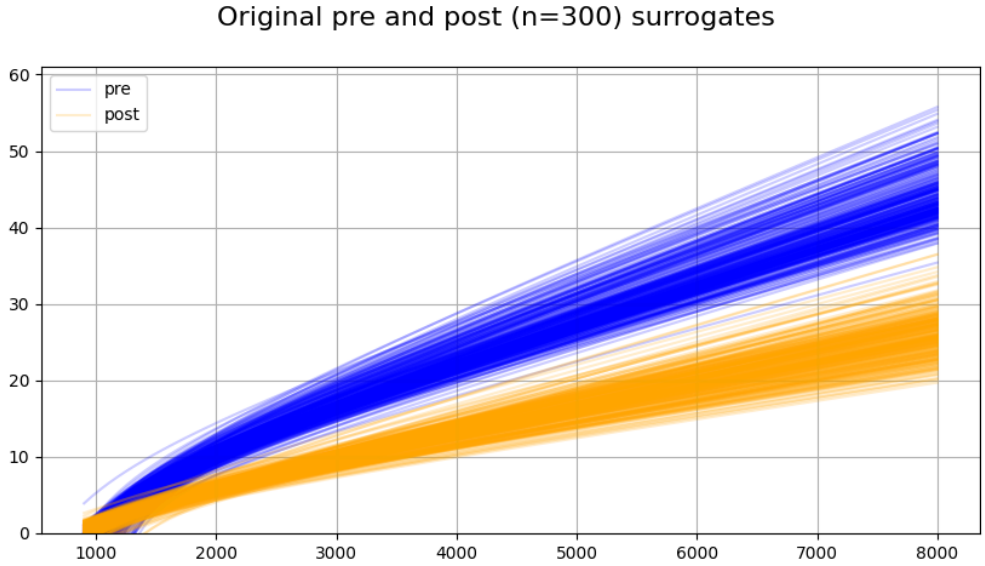


Figure 5.2.10: Exemplary results of the surrogate fluence model for the entire sensor. Each column of the plot represents a different parameter of the surrogate function. The first row of plot presents the distribution of respective surrogate model parameters as measured for the sensor S8. The middle row shows the distribution after scaling and applying the PCA, with the Gaussian function fit in orange. The bottom row shows the same actual distribution as in the first row in blue, and a distribution of generated parameters using the inverse model procedure.



(a) Real data



(b) Generated data

Figure 5.2.11: Comparison of true and generated surrogate functions. In case of real data a sample of 300 curves were selected on random from measured data set (plot a). Plot b presents a generated sample of the same size.

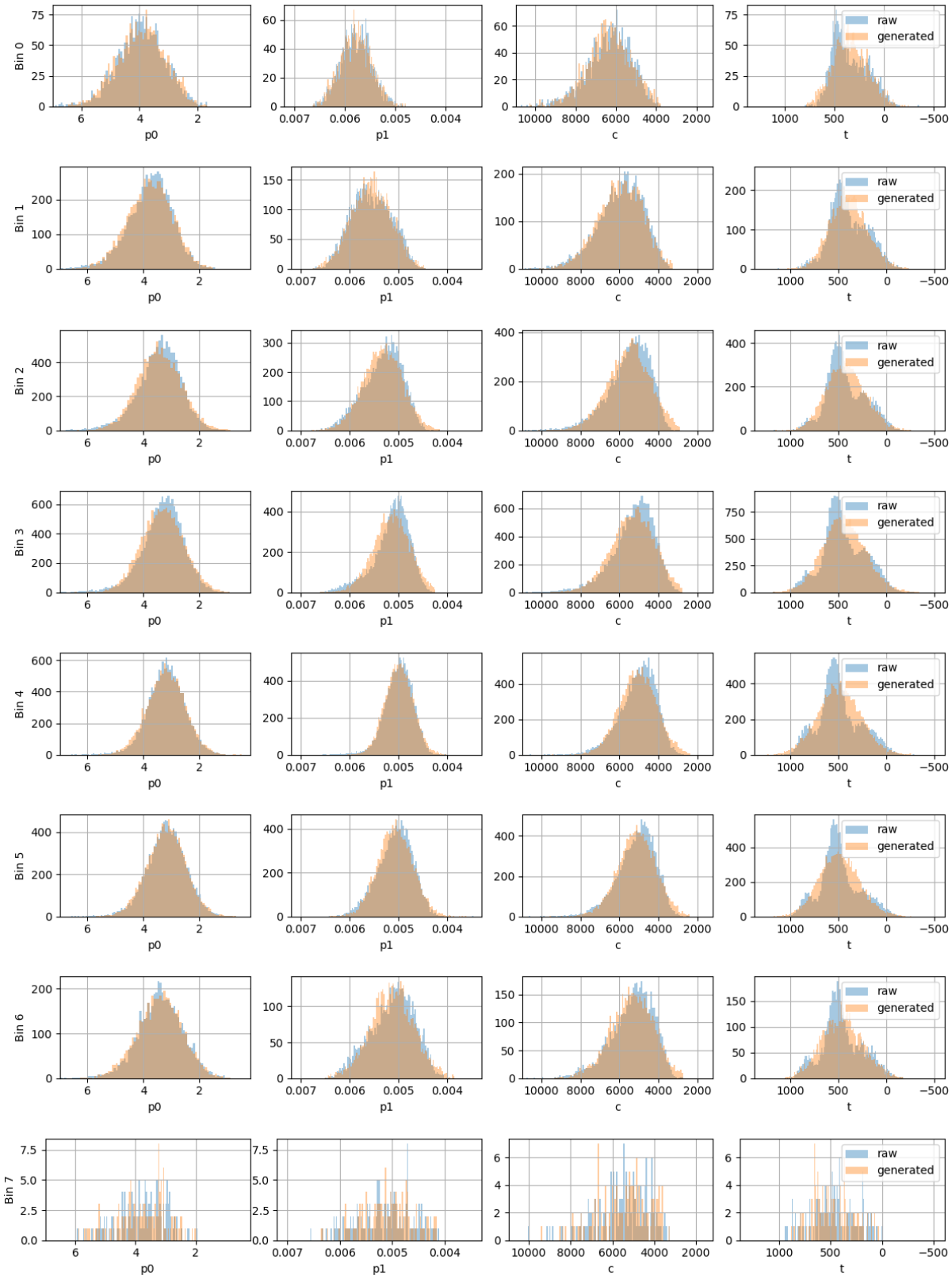


Figure 5.2.12: Comparison of real (blue) and generated (orange) parameters for all fluence regions before the irradiation.

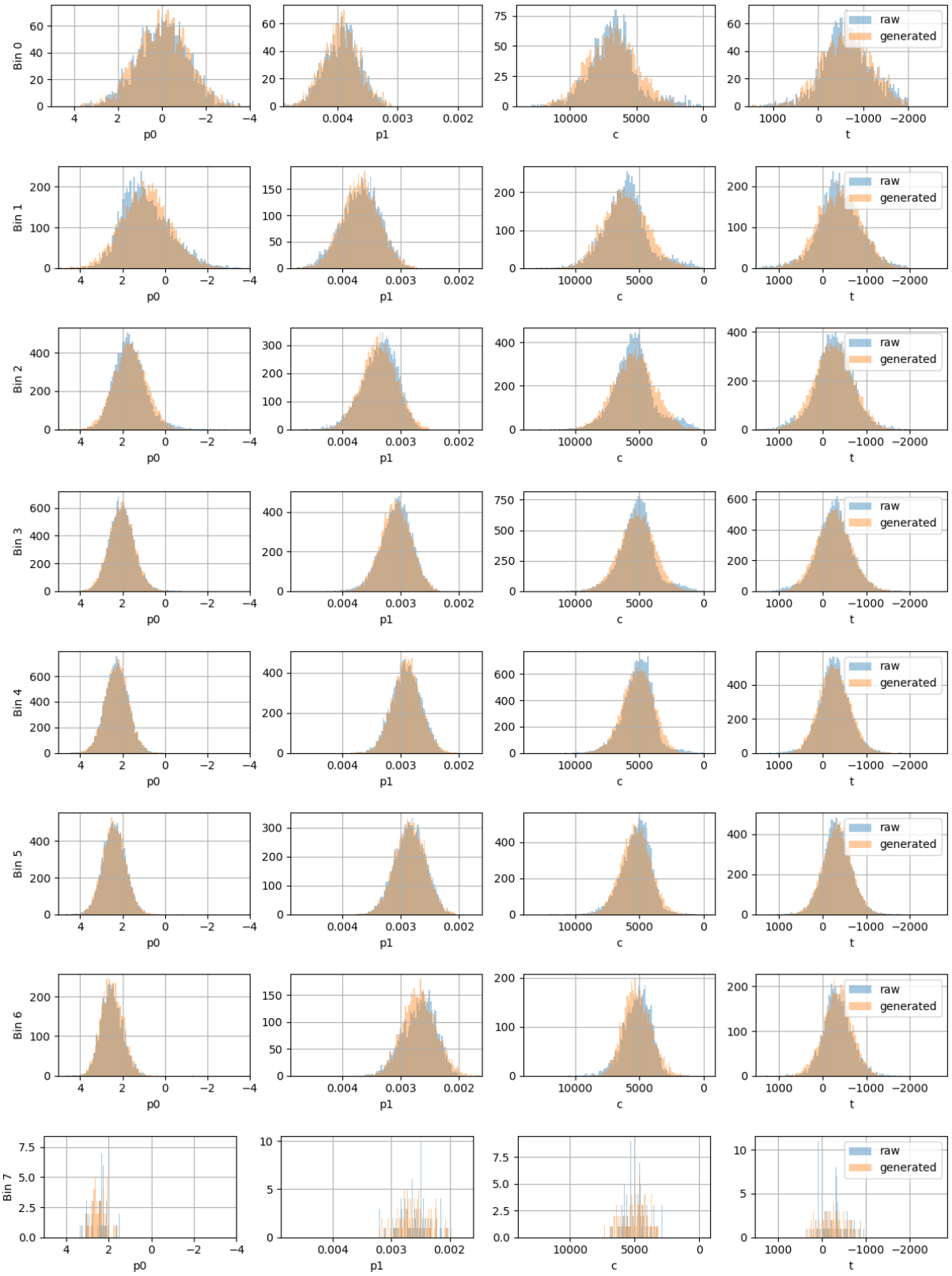


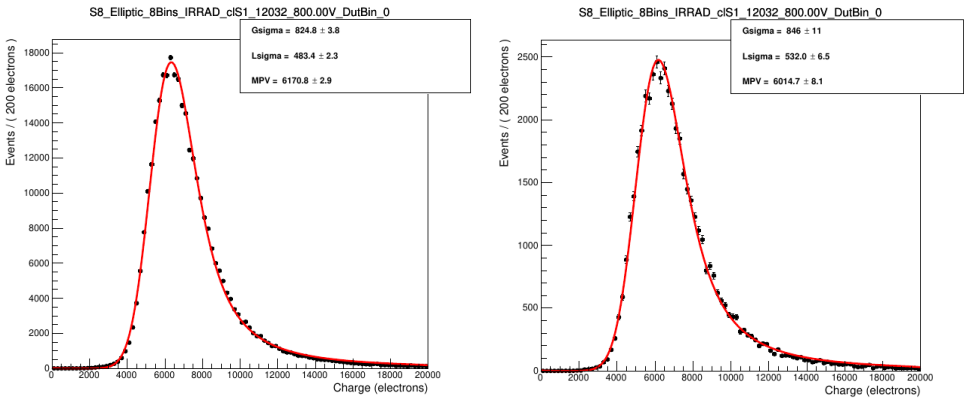
Figure 5.2.13: Comparison of real (blue) and generated (orange) parameters for all fluence regions after the irradiation.

5.2.6 Particle fluence estimation based on the surrogate curve properties

Using the model described in the previous Section we can correlate the fluence and the parameters of the surrogate curves. In other words, it should be possible, by analysing the activation curves of an irradiated sensor to recover the fluence level to what the sensor was subjected to.

Fig. 5.2.14 depicts the most important output of this work. Here we show the exact correlation between the surrogate parameters and the fluence. It depicts the mean and standard deviation of the parameter per fluence bin. In case of $p0$ and $p1$ parameters, there is a clear dependence on the fluence. We can see that the p0 (intercept) is increasing with the fluence, and p1 (slope) is decreasing. This in accordance with our expectations, as it means that with the same signal (eV) the lower ToT number is produced, which meanse the decrease in sensitivity of the sensor after irradiation. On the other hand, the dependency of the remaining two parameters c and t is not so clear. Based on that one can conclude that the fluence has much more impact on the linear part of the surrogate function than on the hyperbolic part. The hyperbolic part of the surrogate determines the minimum of the generated charge needed to exceed the activation threshold. The progression of the parameter change in respect to fluence is also fitted with a third-degree polynomial, which can be used for inter and extrapolation. This fit can be used for reverse estimation of the fluence based on the surrogate.

The fluence estimation was cross-checked in the following way. Using the surrogate function evolution model we generated a fake 'irradiated sensor'. These generated activation functions were then used in the simulation software to produce TOT response and reconstruct the Landau deposition functions. The appropriate MPV values taken from the simulated deposites and real ones have been compared in Fig. 5.2.15. As we see the value of measured and simulated MPV agrees very well. Additionally, we performed the procedure for all fluence bins to get the complete evolution of the MPV value. The results are presented in Fig. 5.2.16. Again a good agreement between the measured and simulated data is apparent.



(a) Ground truth **(b)** Generated surrogates

Figure 5.2.15: Distribution plots of the charge collected in bin 0. The left image shows the distribution made using the actual calibration surrogate functions, and the right image shows the distribution made using generated surrogate functions using our model.

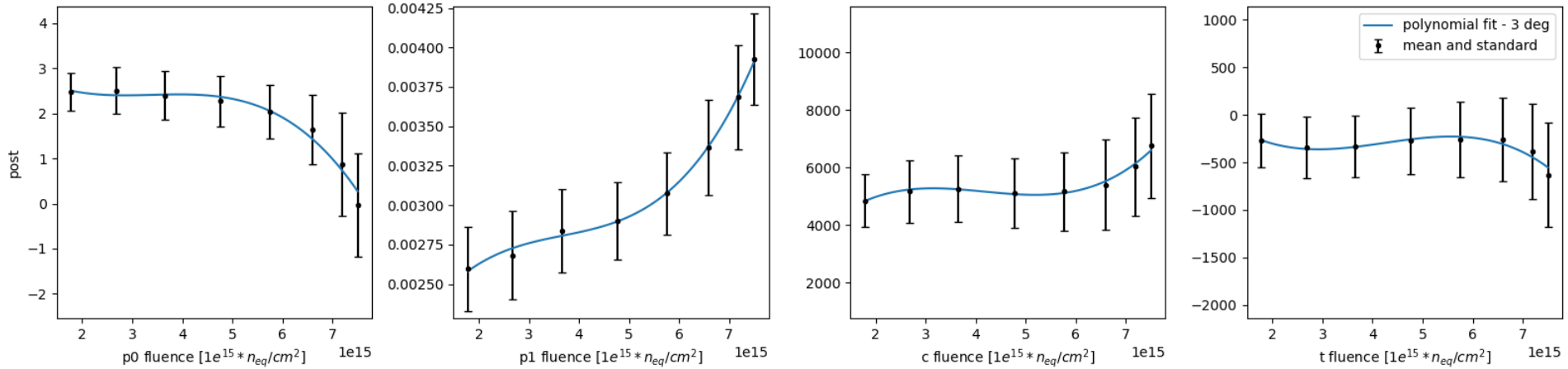


Figure 5.2.14: Surrogate parameters as a function of particle fluence. These values come from model of the sensor post irradiation. The errorbars denote the standard deviation of the distribution. The blue line is a third degree polynomial fit.

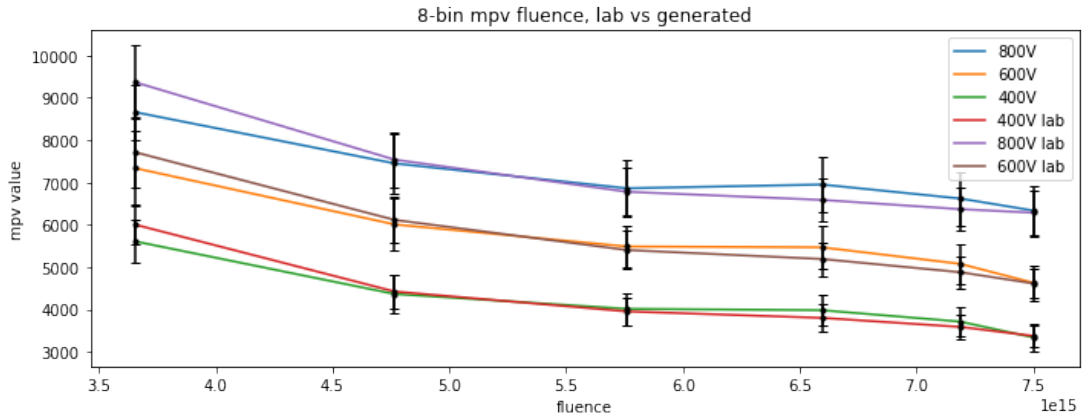


Figure 5.2.16: The plot of the MPV in different bins, using both the ground truth data and the generated parameters.

5.3 Disussion

In this Section, we discussed the purpose of the surrogate function and its meaning in the sensor. We have shown a correlation between fluence and surrogate parameter values. We also presented a simple statistical model that describes the surrogate parameter distribution. This model was later used to find the exact nature of the relationship between the surrogates and fluence. We have shown that this trend can be approximated with a third-degree polynomial.

The results of this work have the following implication: it indicates that it may be possible to use the surrogate calibration to asses the overall fluence experienced by the sensor. Unfortunately, using only a single sensor and single profile of fluence is not enough to calculate the exact coefficient that would lead to the immediate calculation of the fluence based on the surrogates for the entirety of the VeloPix detector.

However, these methods may be used after the commissioning of the detector by repeating our studies and correlating the surrogates with other means of calculated fluence.

Talk is cheap. Show me the code.

Linus Torvalds

6 Software for Velo daily operation

The development of the Large Hadron Collider was foreseen in steps, consisting of data-taking runs and technical stops. Since the radiation may cause damage to the parts of the accelerator as well as to the parts of the detectors, these interruptions are necessary not only to mitigate the radiation damage but to provide a window of opportunity for the upgrade of the composition of the detectors. Since many of the procedures that rely on the detectors are tailor-fitted to the devices, parts of the software created for previous versions of the hardware cannot be possibly used for the upgraded setup. The Storck project was created because of the lack of a system for handling and tracking of the calibration data. The Titania project was designed and implemented in place of the old monitoring system called Lovell.

6.1 Storck

Storck stands for "Data **Stor**age and Tra**ck**ing". It is a database system created to store Velo calibration data. It was developed using python**?** and Django framework**?** . At the time of writing, it is being adapted at CERN for use in the commissioning of upgraded Velo pixel detector and subsequently in its daily operation. This section will present its design. The implementation details can be referred to in the CERN's GitLab repository**?** and online documentation page**?** .

6.1.1 Motivation

The scientific workflow when preparing or managing a high energy particle detector requires taking vast amounts of different types of data. Apart from the immediate data coming from the device, other data types, such as detector's condition (sensors' temperature, date, and duration of the run), are also needed. The development of the monitoring process might be messy and evolve in time. This is also true for the

monitoring tasks when using the detector for taking physical data. Usually, data storage problems are solved with relational databases. But, the relational databases require a predefined structure that is not easy to be modified when the data have been uploaded. Relational and non-relational databases are both inefficient for storing large binary files. For those reasons, the Storck project was created. It utilises a relational database with non-relational elements using the filesystem storage. It is not oriented towards one type of file, nor does it expect any kind of predefined structure. Storck allows to share data between its users and track the changes in the data.

6.1.2 Software stack

The software stack of the Storck project reflects the newest trends in the enterprise software development, as well as best practices in the industry.

6.1.2.1 Django and Django rest API

Python language provides many web frameworks. One of the most popular of them is Django **?** . Django is a mature web design framework based on the Model-View-Controller (MVC) design pattern. Django is split between Model, View and Template. It is important to note that functionally Django View corresponds to Controller and Template to View parts of MVC. This design pattern allows for logical separation of the functional responsibility of the code. Functionality of a Django project are encapsulated in, so called, applications. Each of the application implements its own version of MVC, though it is possible to use any of the components of the MVC design from any other application. This framework also utilises object-relational mapping, which frees the framework user from writing queries using SQL language, and allows for high-level object-oriented usage of the model.

6.1.2.2 PostgresSQL

PostgresSQL **?** is a widespread implementation of a relational database. It is SQL compliant. Importantly, it is free and open source. A recent version introduced a non-relational capability, which is storing the JSON data format along with the ability to make queries using JSON data format.

6.1.2.3 Docker

Docker **?** provides OS-level virtualisation. It uses images and containers. Docker image is a software package that contains everything necessary to run an application. Images are defined in special files called Dockerfiles. Image definition is done by using Dockerfile commands. Images are usually based on operating systems, like Ubuntu Linux or Alpine Linux. The former is 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 referred to as containers.

6.1.3 Service Design Overview

The system's design at heart uses REST API communication for managing the access to the database. Users have multiple 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 convenience, a python wrapper to the REST API also exists, so users can easily create scripts that will interact with Storck. When the webserver receives commands via REST API, it interacts with PostgresSQL to validate the request and responds using an HTTP request, either by providing the requested information (like a list of files, file details) or by sending the file. Because HTTP servers may be a bottleneck when serving large amounts of data, we implemented the possibility of downloading the data using direct files access. When storing files, Storck doesn't write them directly to the database, but it saves them to 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. 6.1.1 contains a schematic diagram for the data flow in the Storck.

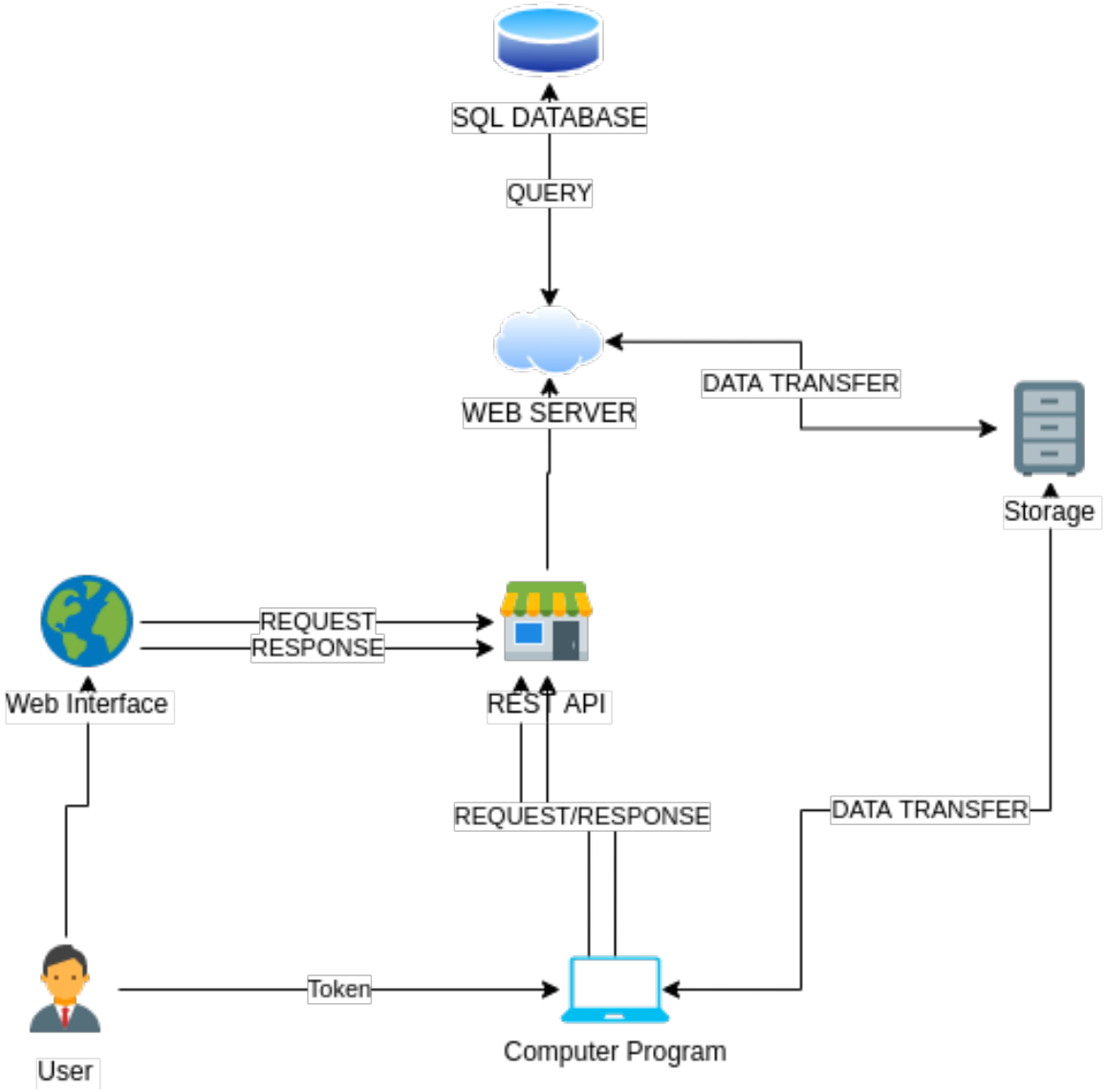


Figure 6.1.1: The conceptual chart of the data flow in the Storck service.

6.1.4 Implementation

The overall implementation spans over a few Django applications. I will not present parts of the code here, as I feel this would be excessive. Yet we provide an overview of the essential parts of the implementation of the service.

6.1.4.1 Files

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 6.1.1: Table of contents of the StorckFile model.

The most crucial element of the Storck is the StorckFile model/database. It keeps track of the uploaded files and stores additional information and metadata. The *id* field contains a unique identifier that identifies each of the files in the database. Next, the *file*, which is of Django's File field type, contains a path to a locally saved file. This property is not implicitly created as the request to file with its contents comes in, but instead, it is designed with the deduplication procedure described in Sec 6.1.4.3. *Hash* is the md5sum of the file's contents, calculated upon receiving. *Workspace* and *user* contain a foreign key connection to the list of all workspaces and users. The *User* field contains the 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 a given workspace with the same *fake_path* attribute. *duplicate_of* contains a reference to the file's 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 metadata property. It is JsonField, which makes it flexible, and allows to create an arbitrary structure of the data inside the 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.

6.1.4.2 Authentication

The CERN account system allows for OAuth 2.0**?** authorisation for third party services. Each person in CERN has a personal CERN account that identifies the user. It uses an (OIDC) OpenID Connect system built on top of the OAuth 2.0 framework. That means that instead of requiring a proprietary

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)

Figure 6.1.2: Workspace endpoints of the REST API and their parameters (left) and the endpoints for interaction with Storck files (right).

login password in Storck, we can redirect the login to CERN SSO (CERN single sign-on) service. There, user can use their CERN Account password and log in, 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 user's identity. Setting up the authentication with OIDC requires setting proper tokens on the administrator side. Those can be generated using the CERN application portal. Although we refer to CERN's authentication procedures in this section, for anyone willing to reuse Storck for 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, as OIDC can be interfaced with other services that provide authorisation.

6.1.4.3 Deduplication

The deduplication process is used when the file is about to be saved in the Storck. The server must receive the file's content, and when the process is finished, the server calculates the md5sum hash value. The database is then checked for the hash matching the file. If the same hash exists in the database, then the record containing 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 progresses normally, and the file path to the incoming file is set to be the same as the already existing file. Additionally, the attribute *duplicate_of* is also set to point to the *id* of the entry of the existing file record.

6.1.5 REST API

The REST API makes for a perfect interface for projects like this. Most modern programming languages are capable of making HTTP requests by themselves, and there CERN already utilises an extensive internal network. REST API is agnostic of an operating system and programing language. In my previous experience with LHCb monitoring, one of the biggest challenges was the compatibility of the database tools, which only interfaced with the 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.

Tab. tab:workspace-api, tab:file-api, present the listing of the REST API methods. The first endpoint

provides a way of creating and managing the workspaces. That includes the creation of the workspace, getting workspaces information, and its associated user list. The second endpoint is responsible for searching for the file, downloading and uploading, and getting detailed information about the files.

6.1.6 Deployment

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

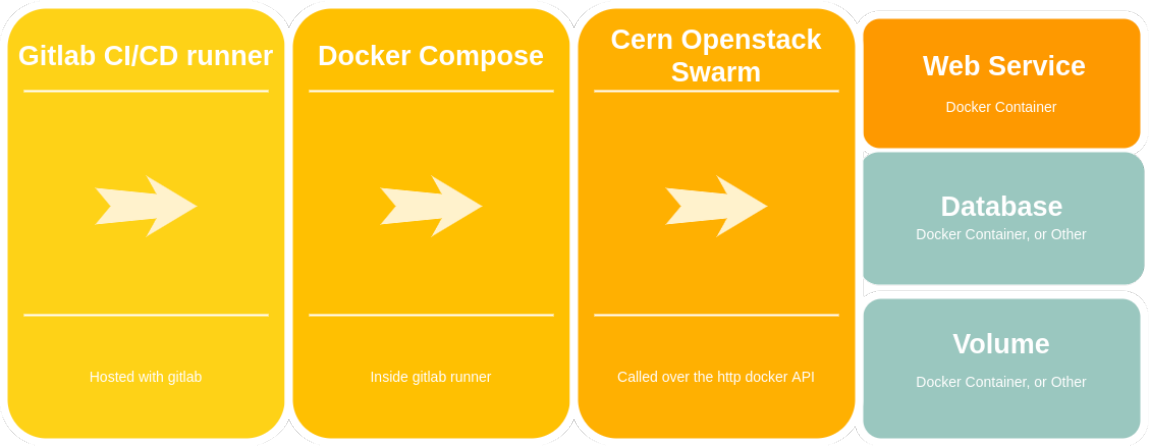


Figure 6.1.3: The schematic of the Storck deployment process. The light green colour means optional components.

The Fig. 6.1.3 depicts the flowchart 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. A database can be run simply by using a predefined docker image. Web App is created using a custom Dockerfile. The base image is the Alpine Linux image with python 3.8. During the building of the image, it set's up necessary environment variables, dependencies, and migrations, and in the end, runs the uwsgi server.

6.1.7 Gitlab CI

As visible in the Fig. 6.1.3, the Gitlab CI run environment is the first part of the deployment process. Although GitLab started purely as a service that allows for hosting and browsing the git repositories, like many other git services, it provides a mechanism for Continous Integration / Continuous Deployment. It means that there is no longer a need for a dedicated server that will be running a test environment and enables manual triggering of some deployment or testing jobs just after committing to the repository. It is now perfectly possible to do those things from the GitLab web interface.

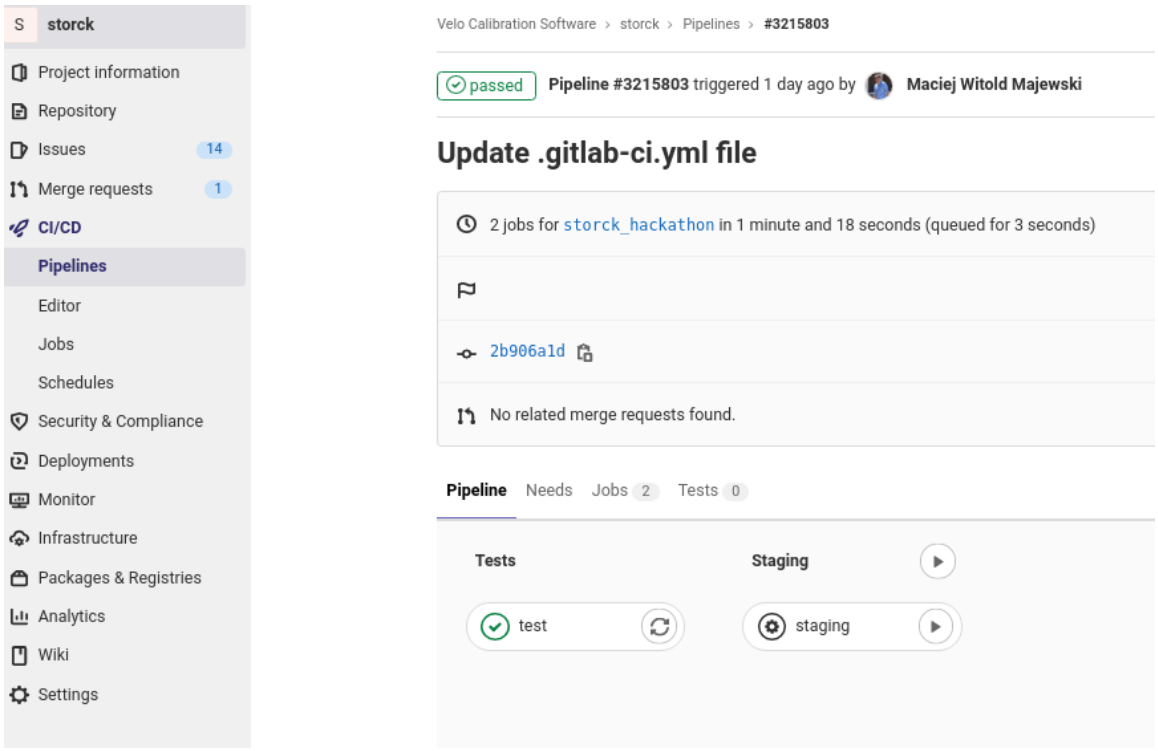


Figure 6.1.4: An example view of a GitLab pipeline

Automatic deployment can be set in various ways. In the case of Storck, the pipeline was set up to 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 the staging server. Only on the main branch, after a successful deployment of staging, it is possible to deploy to the production server. Fig. 6.1.4 shows a view of a single pipeline in GitLab.

6.1.8 Storck web interface

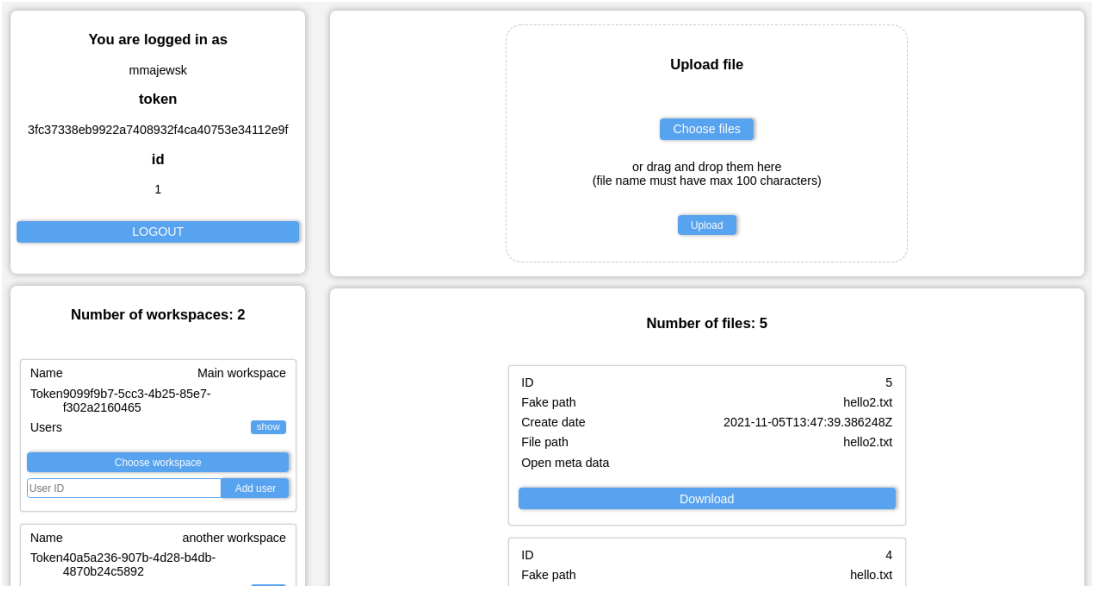


Figure 6.1.5: Storck's web interface.

Storck's web interface is very basic, as it is not meant to be the primary way of interacting with storck. In Fig. 6.1.5, on the left side, there is a space that contains the user's username, id and user token. Below that space, there is a list of available workspaces. Each of the workspaces shows its 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 a drag-and-drop area to upload files to storck. The lower area lists all of the files contained in the workspace and allows for the download of each using a dedicated button.

6.1.9 Storck python client

Storck python client was created as a separate 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
```

or

```
pip install storck-client
```

It is a very small and simple package. It is an object-oriented API in python, and users should create and operate on the Storck client object. The very basic usage is as follows:

```
from storck client import StorckClient

client = StorckClinet(api_host=..., user_token=..., workspace_token=...)
print(client.get_files())
```

Listing 3: A snippet of code presenting how storck python client can be used.

The Storck client constructor accepts three arguments: the HTTP address to the Storck server, user token, and workspace token. These arguments are optional as, instead the environmental variables can be set, in order to not provide the tokens in plain text. The client has several methods, which we present in the form of a table (6.1.2). For a more detailed description of all of the methods, we refer the reader to the Storck documentation page**?** .

Method	description
auth_verify	verifies the credentials t
set_workspace_token	sets the workspace token for the client object
create_workspace	creates new workspace
get_files	outputs list of files 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_user_to_workspace	adds user to workspace

Table 6.1.2: Table of available methods of Storck client

6.2 Titania

Titania is a monitoring framework built with python, Qt? and flask? . Similarly to Storck, Titania is also being adopted at the Velo group for the next LHC runs. This section will present the design and exemplary uses of the framework.

6.2.1 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 complicated in the development due to the amounts of different uncoordinated changes. The platform's source code was not well structured, so adding new plots was usually quite the challenge. This drove us to develop a new platform for monitoring Velo as a python framework.

6.2.2 Software Stack

Python is one of the most popular tools for plotting and visualisation in the scientific community, thanks to packages like Matplotlib? or Plotly? . Famously, the first image of the black hole? was created with python, and visualisation using matplotlib in the first extraterrestrial flight on mars played a significant role? . 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 an easy way to create cross-platform GUI applications. It is an object-oriented library and implements a system of sockets and connections for action triggering.

6.2.3 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 three key components to any monitoring task.

- **Data**
- **Plots**
- **Exploration**

Those are fundamental blocks of any monitoring tasks.

Data here is understood in terms of the source of data. Anytime we want to show some data, we need to access them. There are many ways that this can be done; reading files from a disk, reading a stream of data, receiving data as an HTTP request and others. This also means a format of the data. Data can be text files of CSV structure, binary files, images or sound recordings.

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

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

6.2.4 Using Titania

The detailed API can be found online**?** ; nonetheless, we present here some of the examples of the classes used. Drawing inspiration from the Django framework, the Titania package contains a script that enables the creation of a Titania project.

```
python -m titania start --name ProjectName
```

This command will create a folder with the following structure:

```
ProjectName
|  data/
|  panels/
|  plots/
|  views/
|  main.py
|  config.py
```

This script also contains an exemplary implementation of selected classes. The whole program can be run using the main.py file. Each of the subdirectories is devoted to different parts of the framework.

6.2.4.1 Data

All of the data classes created with Titania, are required to subclass **TitaniaDataInterface**. This requires an implementation of the **fetch**() method. The only requirement for the implementation of this method is that it will return any kind of data. This serves one purpose; all data coming in any shape or form after reading/processing can be sent from this class with a common interface. An exemplary class implementation that gets the temperature data can be found in the listing below (Listing 4).

The implementation above is requesting an open REST API for the weather data, processing it, and returns a list of measurements and a list of time points as python objects. It can be noticed that this could also be implemented in such a way that the data could be acquired in the constructor of the class and then statically returned from the **fetch**() method.

6.2.4.2 Plots

The implementation of the Plot in the Titania is slightly more complicated. While the Data component does not depend on the interface used (Web or Desktop GUI), some plots may differ in implementation depending on the interface.

The minimal requirement for a plot is implementation of **PlotInterface** class. This requires following methods: **draw_plot**(), **get_as_plot_widget**(). The first method is used for any drawing action that the plotting library requires. The **get_as_plot_widget**() requires returning the **QtWidget** object. The output from this function is then used to construct the GUI. There are already classes implemented in

```
import urllib.request
import json
import datetime
from titania.data.data_core import TitaniaDataInterface

class WeatherData(TitaniaDataInterface):
    def __init__(self):
        self.request_string = "https://www.7timer.info/bin/astro.php"+/
        "?lon=6.097&lat=46.241&ac=0&unit=metric&output=json&tzshift=0"

    def fetch(self):
        response = urllib.request.urlopen(self.request_string).read()
        data = json.loads(response)
        temp = [v['temp2m'] for v in data['dataseries']]
        timepoints = [v['timepoint'] for v in data['dataseries']]
        initial_date = datetime.datetime.strptime(data['init'], "%Y%m%d%H")
        exact_timepoints = [initial_date+datetime.timedelta(hours=h) for h
        in timepoints]
        return temp, exact_timepoints
```

Listing 4: Example of Data implementation

the titania framework that can be used for more specific cases that are described in the documentation of the project. In general, it is advised to use the **MplPlot** class for implementing a plot class using matplotlib.

```
from titania.plots.base_plot import MplPlot

class WeatherPlot(MplPlot):
    def __init__(self, parent=None, view=None, *args, **kwargs):
        MplPlot.__init__(self, parent=parent, view=view)
        self.color = 'blue'

    def draw_plot(self, data=None) -> None:
        self.figure.clear()
        ax = self.figure.add_subplot(self.plot_number)
        temp, timepoints = self.view.data.fetch()
        ax.plot(timepoints, temp, color=self.color)
        self.draw()
```

Listing 5: Example of Plot implementation

This implementation (Listing 5) makes a simple matplotlib line plot, using the colour defined in the "self.color" attribute.

6.2.4.3 Exploration

This part is strictly connected to the interface. The minimal requirement for implementing the "Exploration" in the framework is implementation of the **ControlPanelInterface** and it's **get_control_panel()** method. That method must return the "QWidget" object containing the widget of the panel.

```
from titania.panels import ControlPanelInterface
from PyQt5.QtWidgets import QVBoxLayout, QPushButton

class WeatherControlPanel(ControlPanelInterface):
    def __init__(self, data=None, widget=None):
        self.widget = widget
        self.control_panel = QVBoxLayout()
        self.red = QPushButton("Change to red")
        self.blue = QPushButton("Change to blue")
        self.control_panel.addWidget(self.red)
        self.control_panel.addWidget(self.blue)
        self.red.clicked.connect(self.change_color_to_red)
        self.blue.clicked.connect(self.change_color_to_blue)

    def change_color_to_red(self):
        self.widget.plot.color = 'red'
        self.widget.plot.draw_plot()

    def change_color_to_blue(self):
        self.widget.plot.color = 'blue'
        self.widget.plot.draw_plot()

    def get_control_panel(self):
        return self.control_panel
```

Listing 6: Example of Exploration implementation

The exemplary implementation (Listing 6) creates a straightforward panel that contains two buttons named "blue" and "red" (left side of the window in Fig. 6.2.1). It is connected using Qt signals to change the "color" property of the plot class to the according name of the button and executes the plot drawing action.

6.2.4.4 Views

The View connects Data, Plotting and Exploration. Views should implement **TitaniaTabInterface**, with **get_title()** and **initiate()** methods. Since we assume a tabular design of the interface, **get_title()**

should be returning the name displayed name of the tab. The **initate** method is used to start all of the things needed for the view to work. As it is a very basic requirement, many classes exist that already implement valuable features. The recommended class to use when making a tab for titania in Qt is **QtBaseLayoutTab**. It's implementation requires:

1. Passing a data class to **QtBaseLayoutTab** constructor using data argument.
2. Returning a Plot class from **make_plot** function
3. Returning title with **get_title()** function
4. Returning a Exploration class, which extends **ControlPanelInterface** with method **create_control_panel()**

The exemplary implementation of the View class is in Listing 7.

All of those exemplary implementation should be put together in the following directory structure:

```
ProjectName
|  data/
    - weather_data.py
|  panels/
     weather_panel.py
|  plots/
     weather_plot.py
|  views/
     weather_gui.py
|  main.py
|  config.py
```

All of this makes into a single tab that is depicted in the 6.2.1.

Figure 6.2.1: Exemplary code combined into weather plot view in Titania.

```
from panels.weather_panel import WeatherControlPanel
from data.weather_data import WeatherData
from plots.weather_plot import WeatherPlot
from titania.QtGUI.base_tab import QtBaseLayoutTab

class WeatherGUI(QtBaseLayoutTab):
    def __init__(self, parent=None):
        super().__init__(data=WeatherData(), parent=parent)

    def make_plot(self):
        return WeatherPlot(parent=self.parent, view=self)

    def get_title(self):
        return "Weather Plot"

    def create_control_panel(self):
        return WeatherControlPanel(widget=self)
```

Listing 7: Example of View implementation

6.2.4.5 Optimisation and building on top

The example in the section above may seem like an excessive amount of code for such a limited functionality. This may be true for such small projects, but when the number of features grows, such design allows for easy reusability of the already created code. A case for it may be the following example: The **WeatherData** class created in Sec. 6.2.4.1 has a minor flaw: the actual call of **fetch()** may be pretty slow due to the slow loading time of the HTTP request. In some cases, the rate of available REST API calls per amount of time is restricted. Additionally, each time the colour of the plot changes, the **fetch()** method has to be called one more time, which makes the colour changing feature slow. One of the methods to deal with this problem could be an implementation that uses the cache to store the API calls and creates new request when a certain timespan is exceeded. An exemplary implementation of such a class can be seen in the Listing 8.

The class **WeatherDataCached** extends the class **WeatherData**, and adds two attributes **data** and **last_check**. The **data** attribute stores the data coming from the parent **WeatherData.fetch()** method, and **last_check** contains the last time of the call of that method. When the time span of 5 minutes is exceeded, the data is refreshed. Otherwise, the data received with the last call of **WeatherData.fetch()** is returned.

Now, to run a new upgraded version of the program, we need only one additional change seen in Listing 9.

As you can see, the final change to the existing code is minimal, and other parts (exploration and plotting) do not need any changes. This proves the robustness of the framework.

```
class WeatherDataCached(WeatherData):
    def __init__(self):
        WeatherData.__init__(self)
        self.last_check = None
        self.data = None

    def fetch(self):
        now = datetime.datetime.now()
        if self.last_check is None:
            self.last_check = datetime.datetime.now()
        else:
            dif = now-self.last_check
            if  dif.seconds//(5*60) b>= 1:
                self.data=None
                self.last_check = now
        if self.data is None:
            self.data = WeatherData.fetch(self)
        return self.data
```

Listing 8: Example of Exploration implementation

```
...
class WeatherGUI(QtBaseLayoutTab):
    def __init__(self, parent=None):
        super().__init__(data=\textbf{WeatherDataCached()}, parent=parent)
...
```

Listing 9: Update to the WeatherGUI class to use cache data

6.2.4.6 Web app

Although the feature of enabling a dual interface (both desktop GUI and web) is not yet fully developed, the titania project enables such an option.

In order to do so, the following requirements must be met for a view class:

- It must extend both **TitaniaPlotTab** and the **WebTab** class
- It must contain an member object **self.plot.figure** of matplotlib's figure type

With the usage of example snippets of code from previous sections, the dual interface can be introduced by splitting the **WeatherGui**, as depicted by Listing 10.

Such split allows for the implementation of the **WeatherWeb** class (Listing 10), which will be used for the web interface view. The implementation in Listing 11 implementation inherits the **Weather** class and **QtTitaniaWebTab**. Both provide inheritance valid for the web view. Additionally, for easier creation of the web view, there is also a possibility of using **WebMetaclass** (Listing 12). Both Listing **??** and Listing 12 are good ways of creating the web interface view. To enable the web interface, a new configuration file must be created "config_web.py" (Listing 13). It should include the new web view class in the dictionary.

```
from panels.weather_panel import WeatherControlPanel
from data.weather_data import  WeatherData, WeatherDataCached
from plots.weather_plot import WeatherPlot
from titania.common.titania_tab import TitaniaPlotTab

class Weather(TitaniaPlotTab):
    def __init__(self, parent=None):
        self.parent = parent
        super().__init__(data=WeatherDataCached())

    def make_plot(self):
        return WeatherPlot(parent=self.parent, view=self)

    def get_title(self):
        return "Weather Plot"from titania.QtGUI.base_tab import QtBaseLayoutTab

class WeatherGUI(Weather, QtBaseLayoutTab):
    def __init__(self, parent=None):
        QtBaseLayoutTab.__init__(self, parent=parent)
        Weather.__init__(self, parent=parent)

    def create_control_panel(self):
        return WeatherControlPanel(widget=self)
```

.

Listing 10: WeatherGUI class split into two classes

```
class WeatherWeb(Weather, QtTitaniaWebTab):
    def __init__(self, parent=None):
        self.parent = parent
        Weather.__init__(self, parent=parent)
        QtTitaniaWebTab.__init__(self)
```

Listing 11: WeatherWeb class

```
from titania.web.base_tab import WebMetaclass

class WeatherWeb(Weather, metaclass=WebMetaclass):
    pass
```

Listing 12: WeatherWeb metaclass

```
from views.weather_gui import WeatherWeb
config = {
    "ExampleTab": [WeatherWeb],
  }
```

Listing 13: Config file for the web interface.

Next, the config dictionary should be imported and executed in a new "main_web.py" (Listing 14) file that will start a small web server and host the interface on the http://localhost:5000.

```
import config_web
from titania.web.main_window import MainWindow

if __name__ == '__main__':
    main_window = MainWindow(tab_config=config_web.config)
    main_window.run()}
```

Listing 14: The main file for the web interface example.

Figure 6.2.2: A screenshot of the exemplary titania web interface app

All of the code presented in this section allows for simultaneously running the Web and Desktop interfaces. This significantly reduces the amount of code needed to run both the desktop version and the web interface.

6.3 Conclusions

Titania is a versatile monitoring framework, and Storck should create a reliable backbone for the calibration data in the next runs of Velo at LHCb. As of the time of writing, it is being introduced, beside Velo, to another of the silicon systems at LHCb UT - Upstream Tracker. Both projects are open source

and fully available online. They solve not only the specific problems of the LHCb but also provide more general solution to the problems that might occur in other areas or physics experiments. Titania was presented in its earlier stages at the PyHep conference in 2020. Hopefully, the upcoming year will prove in-vivo the value of both of the projects and promote its uses in other experiments.

I have done a terrible thing: I have postulated a particle that cannot be detected.

Wolfgang Pauli

7

Neutrino induced processes

In this chapter a very short description of the possible reactions detectable via tracking detectors and induced by neutrinos are described. The Author did not perform any physics analysis regarding the neutrinos but contributed to reconstruction algorithm described in the next chapter. Neutrino observation is one of the most challenging problems in modern particle physics. Neutrino experiments come in many different shapes and sizes. The notable examples of the extreme lengths that scientists come through pursuing the science of neutrinos include;

- Super-Kamiokande**?** - a detector 1km deep underground, holding 50 220 tons of ultra-pure water
- ANITA**?** - radio pulse detector suspended on the helium balloon flying at the height of about 37,000 meters over the Antarctic
- ICE-CUBE**?** - contains 5 160 optical sensors placed directly on the South Pole, 1,5 - 2,5 km deep inside the Antarctic ice sheet

LARTPC (Liquid Argon Time Projection Chamber)**?** detectors are a family of detectors developed from TPC detectors (Time Projection Chamber)**?** . In general, such detectors consist of a volume of medium (radiator) between two plates (anode and cathode) that create an electric field. When a particle travels through the detector's active volume, it excites the electrons, which interact with the electric field and move toward the collection plate (see Figure 7.0.1). The TPC detectors used gas, and the LARTPC detector uses the same working principle, albeit the medium filling the volume is liquid argon.

*Source: https://en.wikipedia.org/wiki/File:RealSchematicTPC.png

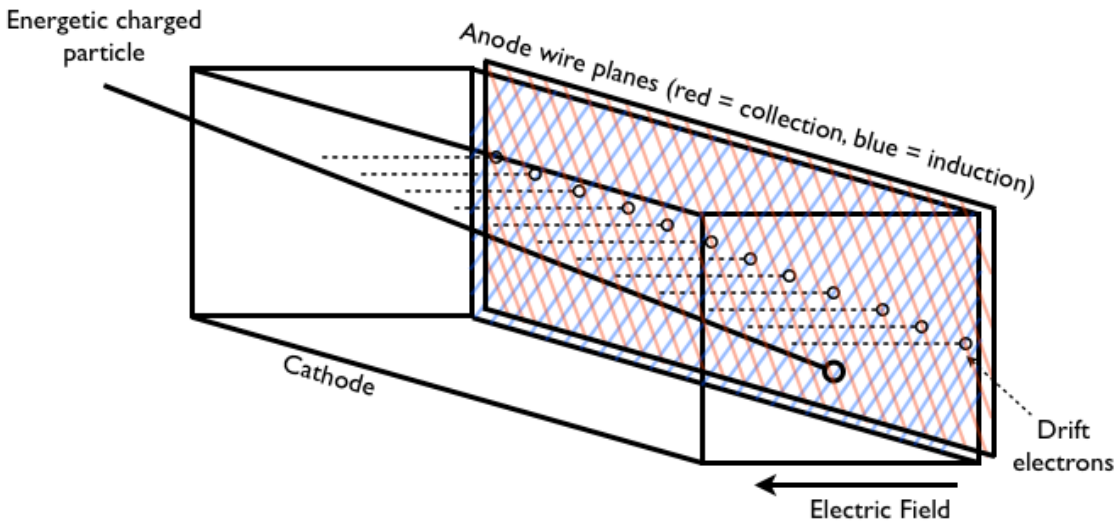


Figure 7.0.1: A diagram of LARTPC opration*.

The argon is an excellent choice for the medium as it has high electron mobility (electrons are easily pulled along by the electric field) but low electronegativity (which means that the electrons will be less likely to be absorbed into the material).

The critical part of the design of the detector is its collection plates. There are two popular solutions of the LARTPC anode planes: pixel-based and wire-based. While the pixel-based approach is still mainly in development, the wire-based one is more popular and has been implemented in MicroBoone, ICARUS, SBND, and the DUNE far detectors**?** . The anode for the wire-based design consists of multiple wire planes placed parallel to the cathode. Each of the planes consists of multiple parallel wires. The set of wire planes is kept at a bias voltage. Their electrostatic potential allows electrons to pass the first two planes undisturbed while inducing the signal. The electrons stop at the last plane called a collector. Each of the first two planes (also called induction planes) has a unique orientation of the wires. This orientation allows for a reconstruction of the 2D signal (Figure 7.0.2). Additionally, a 3D reconstruction is possible to obtain by calculating the distance travelled by the electrons, using the 2D and the constant flow rate of the electrons. This requires trigger information, which can either come from the particle source or can be achieved using the scintillation effect (light flash) of the argon. The trigger information provides the timing information about the event**?** .

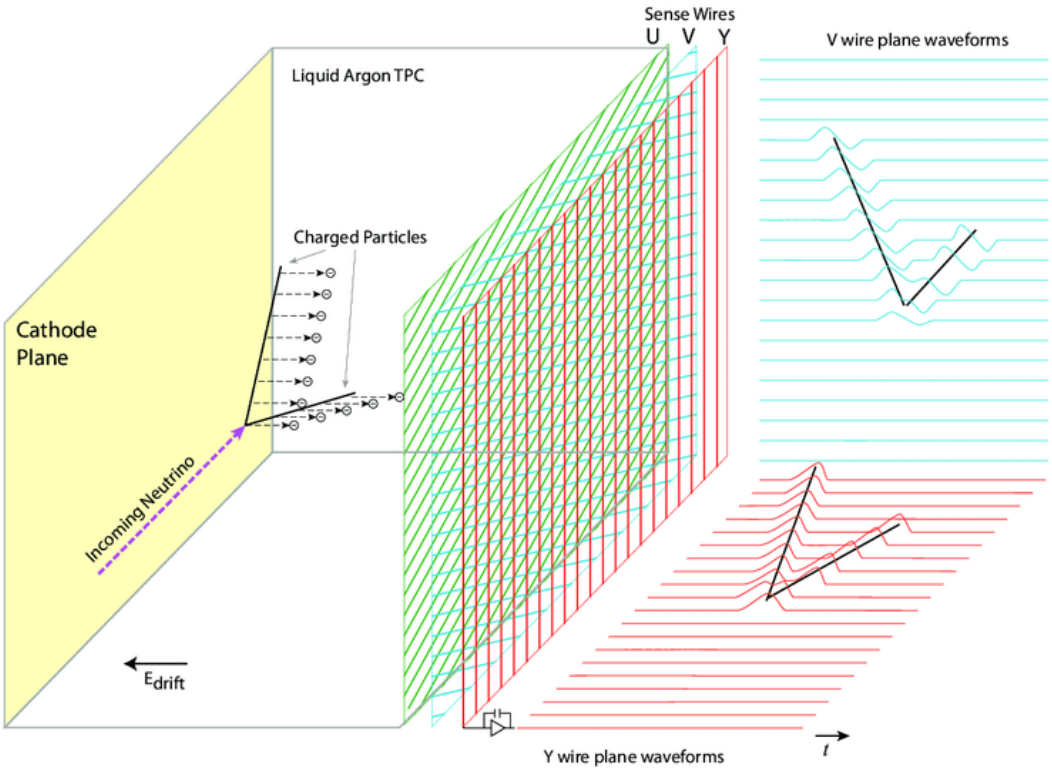


Figure 7.0.2: A diagram of LARTPC planes[?]

7.1 Neutrino production and observation

Neutrinos are notorious for their elusiveness. Their only known interaction with matter is through the weak force. One of their most popular interaction is in **beta decay**, otherwise the neutrinos in the range of medium and higher energies interact via: **Elastic and Quasielastic scattering**, **Deep inelastic scattering** and **Resonance production**.

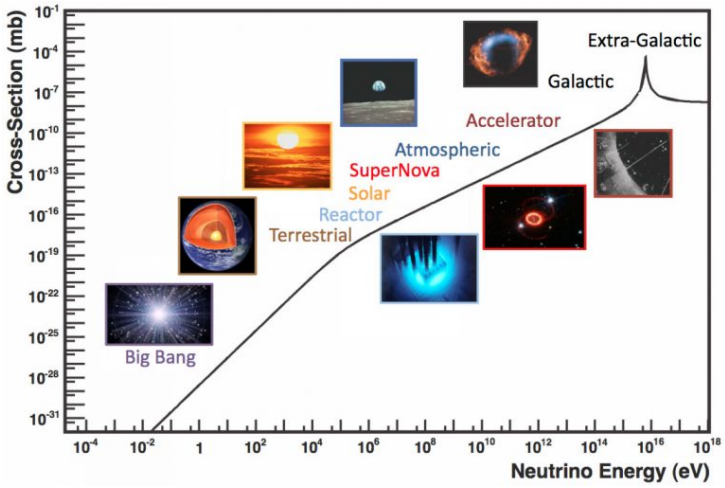
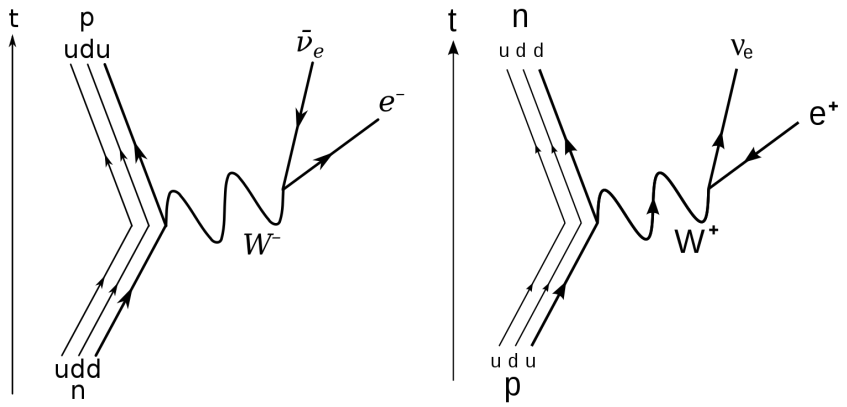


Figure 7.1.1: Neutrino energy scale[?]

7.1.1 Beta-decay

The first processes in physics, also known as nuclear decays, that involves neutrinos are B^- and B^+ decays. The common examples of these processes are $n \to p + e^- + \bar{\nu}_e$ and $n \to p + e^+ + \nu_e$ respectively, which on the quark level can be expressed as $u \to d + e^- + \bar{\nu}_e$ (Fig 7.1.2a) and $d \to u + e^+ + \nu_e$ (Fig 7.1.2b).

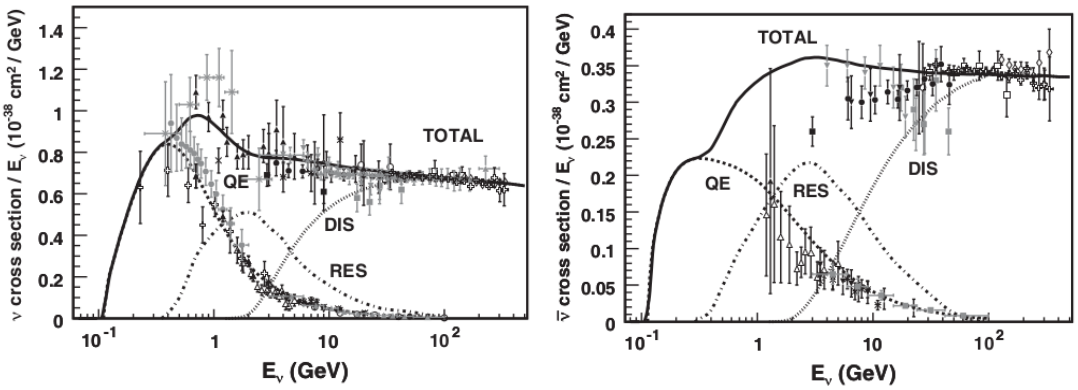


(a) Beta minus **(b)** Beta plus

Figure 7.1.2: Two types of beta decays involving neutrino*.

7.1.2 Medium and higher energy interactions

The most interesting range of energies, defined arbitrarily, for neutrinos spans from medium $(0.1GeV)$ to high $(1TeV)$ values. As previously stated, neutrinos only interact with the matter through weak interaction. This is why the neutrino is not directly detectable, but instead, detectors rely on secondary particles for evidence of neutrinos. Multiple probable processes can produce observable particles (see Figure 7.1.3); we will introduce some more probable ones in this subsection.



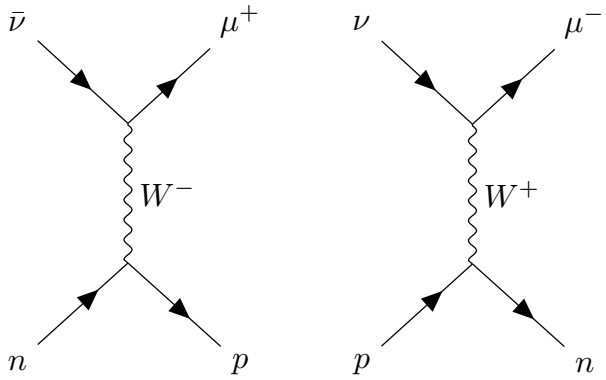
(a) Neutrino **(b)** Antineutrino

Figure 7.1.3: Neutrino (A) and Antineutrino cross-section in GeV section with contribution from different processes.[?]

*Image source: https://commons.wikimedia.org/wiki/File:Beta_Negative_Decay.svg and https://commons.wikimedia.org/wiki/File:Electron_Capture_Decay.svg

7.1.2.1 Elastic and Quasielastic scattering

Neutrinos can elastically interact with the atom's nucleus what leads to their excitation and possible emission of nucleon. It is as simple as that in the case of the neutral current elastic scattering. Neutrino interacts with the nucleus and frees a proton or neutron, which can later interact with matter more easily. In the case of the charged current quasi-elastic (CCQE) interaction, it changes the nucleon and emits a lepton. For the free nucleon interaction it is: $\nu + n \rightarrow l^- + p$ and $\bar{\nu} + p \rightarrow l^+ + n$ where l stands for lepton. Fig 7.1.4 depicts exemplary CCQE scattering. It is important to note that this interaction both releases a nucleon and produces charged particle. The quasi-elastic scattering is the most popular interaction in the medium range of energies.



(a) Beta minus **(b)** Beta plus

Figure 7.1.4: TODO

7.1.2.2 Deep inelastic scattering

In deep inelastic scattering, neutrino scatters off on a quark coming from a nucleon. The scattering is conveyed by a virtual boson (W or Z). In cases of a CC produces a μ or μ^-. The NC process doesn't produce additional charged particles. The examples below present the case with the ν_μ, but other types of neutrinos are also allowed in deep inelastic scattering. Some of the possible processes of DIS:

$$\nu_\mu N \rightarrow \mu^- X$$
$$\bar{\nu}_\mu N \rightarrow \mu^+ X$$
$$\nu_\mu N \rightarrow \nu_\mu X$$
$$\bar{\nu}_\mu N \rightarrow \bar{\nu}_\mu X$$

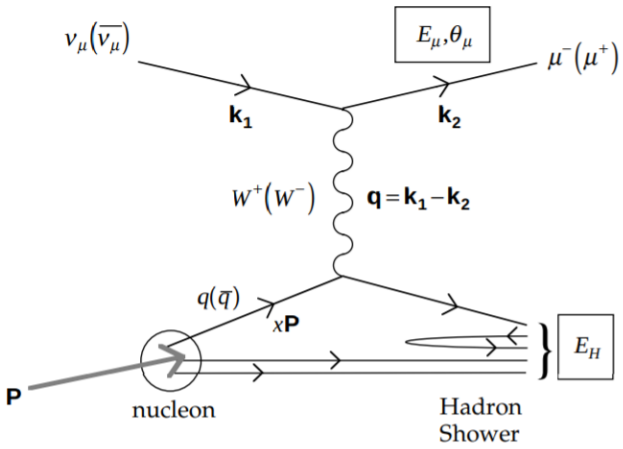


Figure 7.1.5: A diagram of deep inelastic interaction of a neutrino with a neutron (charged current)[?] .

7.1.2.3 Resonance production

With enough energy, neutrinos can cause the excitation of a nucleon. This excited state produces a baryon resonance along with a lepton, and then the resonance quickly decays into a nucleon and an accompanied meson:

$$\nu_\mu N \rightarrow \mu^- N*$$
$$N* \rightarrow \pi N$$

Other decays modes are also possible but less probable. The most common pion production is possible in this interaction. For ν_μ and p, n targets, there are seven possible resonant single pion reaction channels (seven each for neutrino and antineutrino). In those channels, only two of them do not produce a charged particle.

7.2 Ultra high energy neutrinos

The neutrinos of the ultra high energy are impossible to be produced on earth (yet), so they are searched in the cosmic radiation. All of the experiments mentioned in the list Section 7, are focused on registering ultra high energy neutrinos from space. The task is problematic not only because of the small cross-section of the interaction of a neutrino with the matter but also because of the other types of radiation that can obscure the discoveries. This is why they need a very thick shielding (Super Kamiokande and ICE-CUBE). Due to the rarity, ultra high energy neutrinos are one of the least experimentally researched branches of particle physics and thus are worthy of mentioning here.

7.3 Summary

Although the neutrinos themselves are limited to only weak interactions, after the deposition of the energy, a vast array of mostly charged particles can either be produced, excited or use the deposited energy to create a further chain of reaction. The selected processes depicted in Sec. 7.1.2.1-7.1.2.3, provide a short glance at the possibilities. It is worth noting that although there exist many channels of interaction for the neutrino, the cross-sections are very small compared to other branches of physics.

[...] to boldly go where no one has gone before [...]

Star Trek

8

Reinforcement learning for LARTPC detector

As this thesis centres around the intersection of particle physics and machine learning, it was quite natural to brainstorm the ideas for pushing forward this niche of science. One of the very conclusions from studying very different applications of machine learning to many of the particle physics problems was the lack of unification. Although many of the algorithms solve critical problems in the domain, a significant portion is highly dependent on the type of detector.

This is partially expected; after all, each of the detectors is designed for a particular branch of physics that they will study. Yet, all of those experiments study exactly the same universe and expect that the underlying rules of physics are exactly the same, and the holy-grail of particle physics, called the Standard Model aims to unify all of the branches of particle physics.

This notion is not reflected in the pursuit of machine learning algorithms for experiments.

This, and my participation in 2017 on "Machine learning for High energy physics" school in Oxford, pushed me to sketch an idea for a more general algorithm.

In this pursuit, I decided that although the vast sea of possible methods that stem from the supervised machine learning is ever-expanding, the nature of this branch of machine learning is too limiting. It expects concrete examples of input-output pairs, while many of the tasks at best can only provide an estimate of the overall performance. This is why I have chosen the Reinforcement Learning approach for further investigation. As someone who worked both in industry and in science with artificial intelligence, I know that the fewer constraints the problem has, the harder it is to solve it. To put it mathematically, the difficulty of the problem is inversely proportional to its number of constraints. This is the cost of the generalisation.

The conceptual model for a high-level reinforcement learning definition of the general problem of both identification and tracking of the particles can be brought back to the corpuscular model of parti-

cles. Intuitively this is how we perceive the particle, especially when considering the Feynman diagrams. In this model, a particle is an object travelling through space and being subjected to the rules of physics following decays and interactions.

Such a model can be translated into the reinforcement learning environment setting quite easily; a particle is an agent interacting with the environment while subjected to its rules. Just like a video game character moving through a game environment. The agent's ability to imitate reality can be assessed by checking how well it obeys the rules of the environment. If the agent obeys the rules, it is rewarded. If the agent does not obey the rules, it is punished (either by receiving a smaller reward, or even a negative reward). The ideal agent/model is then the one that maximises the reward.

The rules that govern the behaviour could also be approached as a generative task, in which the goal is to produce realistic record of the behaviour of the particle. This could be implemented as a GAN (Generative Adversarial Network) architecture, where the progress of a generative network is assessed by comparison to the simulation rules. But then this would mean that the task is purely about the generation of the data, and would not benefit from the conceptualisation of the physical rules in a machine learning model. The information gained from the training could be use for particle tracking and identification, which is far more useful. This is how we land on a reinforcement learning idea, where the data generated from the simulation is used to train a model for identification and (partially) tracking of the particle.

8.1 Simulations and Data

The inspiration for this research comes from a DeepLearnPhysics public dataset. That dataset contains simulation data, which includes pairs X and Y of both the simulations of records of particles in the LARTPC and their ground truth classification in each position in space. The environment for the reinforcement learning was prepared using that simulation data.. Although the code details of the simulation weren't explicitly mentioned for the dataset used in this study, the DeepLearnPhysics collaboration prepared a preprint article for the new version of the open dataset **?** , in which they mention that the simulation was prepared using LARTSOFT **?** and GEANT4 **?** .

The simulation generates particles inside a LARTPC detector **?** , and their progress in time is recorded by the detector. The source point of the generated particles is uniformly distributed in 3D. The total particle multiplicity is set to be uniform between 2 to 5. There is an equal and uniform probability of generating any particle from four categories presented in table:

Category	Particle Type	Energy range (uniform)	Multiplicity
0	light lepton (electron or muon)	50 to 1000 MeV	0 to 3
1	gamma ray	50 to 1000 MeV	0 to 2
2	charged pion (pion or anti-pion)	50 to 1000 MeV	0 to 2
3	proton	50 to 400 MeV	0 to 3

Table 8.1.1: Table of simulated particle properties for LARTPC data generation.

The energy deposition for an individual particle is stored in a 3D voxel of 0.5 cm^3 in MeV units, but

later it is saved with a scaling factor of 100**?** . Next, the 3D data is projected onto three 2D planes. Then the data that contains less than 5 pixels in any given projection with a pixel value lower than 0.1 MeV is rejected. The values of voxels lower than 0.1 Mev are trimmed to 0. For the purpose of this research we do not use full 3D voxelised information, but one of the 2D projections.

8.2 Dataset

The dataset generated from the simulations and used in this research is available at the DeepLearnPhysics collaboration website**?** . This dataset contains 15.000 training samples as well as 10.000 test samples. Each of the samples contains a single simulation of a signal event, as discussed in the previous subsection. The simulations are saved as two 2D images - a source image where only information is the energy deposit in the pixel and a target image - where pixels hold the information about the type of the particle in the given pixel. Also, the target image contains 3 types of pixels: background, EM-shower particles, and track particles (i.e. not EM-shower).

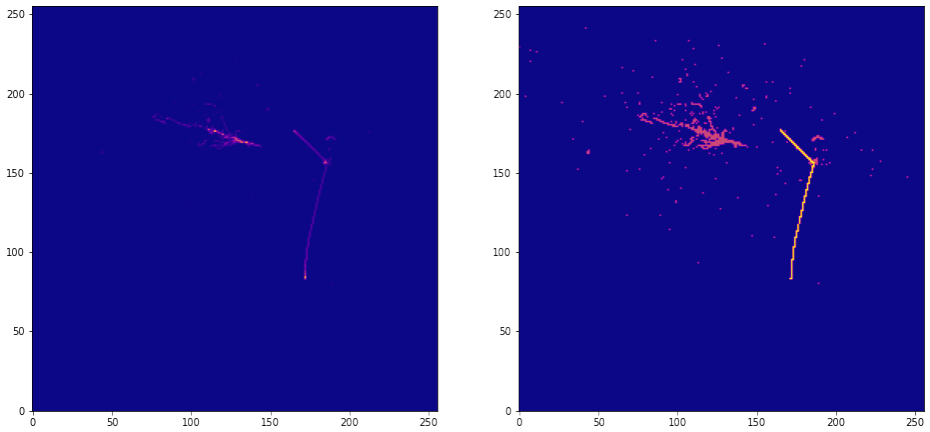
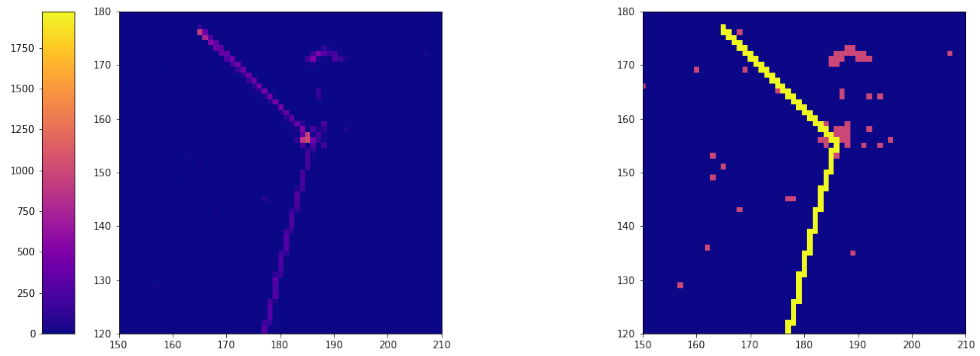


Figure 8.2.1: Exemplary data from the LARTPC dataset. **Left** image is the source input, and the **right** image is the desired output (target)



(a) Source (X) **(b)** Target (Y)

Figure 8.2.2: Examplary data zoomed into a smaller region.

8.3 Environment

The reinforcement learning approach requires a "game-fication" of the problem. That means that we essentially want to turn the problem into a game, preferably one that allows for gaining points along with the progress of playing. For this purpose the gym-lartpc environment was created**?** . It is based on the openai-gym library**?** . The game should consist of an environment and an actor. The actor can interact with the environment and modify it. In the case of lartpc data, the actor will be an analogue to a particle, and the environment is analogue to the space inside LARTPC medium. Before we detail the implementation itself, it is vital to present a formalisation of the problem.

8.3.1 Formalisation

The usual framework for mathematical formulation of the reinforcement learning problem requires a definition of the problem in terms of Markov Decision Process (see Sec. 3.5.0.1) or MDP. This subsection will present steps towards representing the gym-lartpc environment as an MDP, along with introducing key objects that comprise the lartpc-gym package.

8.3.1.1 Maps

In the sense of translating the LARTPC data into game, the images of the dataset can be thought of as maps, which the agent will explore. Although original dataset samples contains source and target, for the purpose of creating the game environment we add an additional map called canvas. The agent will explore the source map (S, 512x512) and by using it the agent will try to approximate the target map (T, 512x512). An example of source and target can be seen at 8.2.1. The output of the approximation is saved on the canvas map (C, 512x512x3), which is the actual output of the algorithm. The canvas map C is used to create an image containing categorised pixels. It is This image then can be compared with the target of a given sample. The majority of the environment states are not available for the agent, so we only consider its nearest neighbourhood. This neighbourhood can be expressed as a quadratic window

n x n. In total there are three types of window objects source map window W^S, canvas map window W^C and target map window W^T.

It is useful to mentally imagine this task and the usage of those three maps as learning to paint a picture, using a sketch. The source map S, can be thought of as an sketch of a ready painting T. Then, the task is to paint a picture C which will be the most resemblant the T, using only S as reference.

8.3.1.2 States

We assume that the actor occupies a position on the map $P(x, y)$, with each step, this position may change. There is only a small subset of states from an environment available for the actor, defined by the window of the map. The state in the current timestep t can be denoted as $s_t = (W_t^S, W_t^C)$. These states create state space S of MDP.

8.3.1.3 Actions

In such defined environment the total action can be expressed as composed of two sub-actions:

Movement action - a decision on the direction of the next step in the game, expressed as a 2D vector m_t, where $[1, 1]$ is a movement by one pixel right (x-axis) and one pixel up (y-axis).

Put action - a KxKx3 tensor denoted by c_t. This matrix will be pasted into the canvas map C, on the coordinates of $P(x, y)$. The the last dimension of the tensor denote probability of given pixel belonging to one of the three categories (background, EM-shower particles, and track particles). The centre coordinate of the c_t has the same coordinates as the Source Window centre.

The action can be expressed as a tuple $a_t = (m_t, c_t)$. If we would go back to analogy of painting the picture, then you can imagine, that the m_t would be direction of a stroke, and c_t would be set of colors to use to paint. The actions create action space A of MDP.

8.3.1.4 Rewards

Finding a correct reward function in reinforcement learning is a challenge. Therefore, this part is open for interpretation by the user in the implementation of the environment. That being said, the default implementation for the reward R_t calculation follows the algorithm:

$$R_1 = N_{discovered}(W^c) \tag{8.1}$$

$$R_2 = N_{\text{nonzero}}(W^s) + 0.9 \times R_1 \tag{8.2}$$

$$R_3 = \begin{cases} R_2 - 5, & \text{if } R_2 == 0 \\ R_2, & \text{otherwise} \end{cases} \tag{8.3}$$

$$R = \begin{cases} R_3 - 15, & \text{if } Center(W^s) == 0 \\ R_3, & \text{otherwise} \end{cases} \tag{8.4}$$

Where $N_{discovered}(W^c)$ return the number of the pixels in W^c which haven't been categorised (painted) yet. This encourages to find the pixels that haven't been visited yet. The term $N_{\text{nonzero}}(W^s)$ is the number of non-zero (empty) pixels in source window W^s. The term $R3$ is a punishment when both $R1$ and $R2$ are zero, meaning that the source window is empty, and target window is already fully processed. Finally, R adds a punishement if the central position of the source window is empty.

In summary, this reward system encourages exploration of both non-empty pixels on source map, and non-processed pixels using canvas map. The reward highly discourages to stay in one place, or in locations in which the source map has zero valued pixels.

8.3.2 MDP and POMDP

The elements of states, actions and rewards, along with the logic of the lartpc-gym environment, give rise to a Markov Decision Process Formulation (see Sec 3.5.0.1), and allows us to use standard reinforcement learning methods to solve the problem posed by the environment.

But careful readers can notice that the term R_1 may be breaking the Markov property. Indeed, depending on the defined reward function, the environment can break the Markov property. And with the reward defined as in Eq. 8.4 it does indeed brake the Markov property, because the new state with the canvas map is constantly updated by the agent, canvas window W^c_t contains the result put action c_{t-1} from the previous step. It is important to note that in this case, only the number of already visited locations on the map, that are apparent in the W^c_t is contributing to the reward, and not the values contained on W^c_t.

Yet, this does not mean that the environment is unsolvable with reinforcement learning. Many of the novel and state of the art solution to gaming environments, such as AlphaStar**?** , or OpenAI five**?** are known to be rather non-markovian. In this case, another formulation of the problem may be useful: POMPD (Partially Observable MDP). POMDP can be expressed as 6-tuple (S, A, P, R, Ω, O) , where (S, A, P, R) are the same as in MDP, but now the agent doesn't observe the states directly, but rather through observation $o \in O$. This observation is sampled according to some distribution $o \sim O(s)$. The $O(s)$ is a function that provides an observation given the state s with some probability. In essence what that means is that the full underlying state of the environment may be obscured by the O. In a case of non-markovian environments the state can be understood as state with it's past history, and the observation O provides only the most recent state, obscuring the past. A classical Deep Q-Learning has no explicit mechanism to solve the problem stated in this way, but some works**?** show that by adding the recurrency to the network, this becomes solvable. In the case of the model presented here, there is a hidden recurrent dependency of the model, as it reads and writes values to the target canvas.

8.4 Gym-lartpc package

The environment presented in the previous section was implemented in a Python Package called "gym-lartpc"**?** , which contains openai-gym interface. Openai-gym**?** is a popular open-source project containing reinforcement learning environments. Each of the environments provides exactly the same style of the interface. The minimal requirement for creating compatible environment with openai-gym is to

extend the **gym.Env** class, and provide **step(action)**, **reset()** and **render()** methods.

Although the package has the openai-gym environment interface, it also provides a separate custom interface for more sophisticated options for users' convenience.

Although it does not contain official API documentation, the code itself is very clean and reads well enough.

8.4.1 Classes

The most important classes in the package are LartpcData, DetectorMaps, Lartpc2D, and Cursor.

Cursor denotes a region on any map (Source, Canvas, Target). Its x and y coordinates and can be used to get or set a window of values on a map.

DetectorMaps contains Source Canvas and Target maps together. It allows getting random positions on maps and nonzero positions.

Lartpc2D contains the logic of the game. Holds both Cursor instance and DetectorMaps instance. It's **start()** method allows to start a new game. **step(action)** can be used to make a step in the game environment.

LartpcData uses either a path to a folder containing the data in the ".npz" format or a list of files that contain training data. It can be used to get a tuple of source and target maps as a python generator object.

8.4.2 Example game

The listing in 15 shows an exemplary use of the Gym-lartpc package. The Lartpc2D class accepts a dimension of the result and a maximum number of steps as arguments. Here the result dimension means the size of the two-dimensional window that will be used on the output map. When using the **game.detector.set_maps()** method the maps are set to a random source target pair coming from LartpcData class. The **game.start()** resets the state of the environment and assigns a random, non-zero, non-visited location to the cursor in the environment. Then, the current observation of the game can be accessed by the **game.get_observation()** attribute and action can be passed using **game.step()** method. This will move the cursor according to the movement contained inside the action and will change the value on the canvas map at the scope occupied by the cursor. It is worth noting that three types of objects are exchanged between the environment and agent (see implementation in 16);

Action2Dai which embodies an action $a_t = (m_t, c_t)$. Contains **movement_vector** attribute (m_t), which is expected to be a numpy array of shape 1x2 (a vector of length two), and a **put_data** (c_t) which is expected to be 2D array of a shape compatible with result dimension. This is the action described in Sec. 8.3.1.3.

State2Dai which contains the state of the game, as expected by the openai-gym. The state of the game in the gym-lartpc is a tuple consisting of current observation of the environment, reward, "done"

```
def bot_replay(data_path, viz=True):
    max_step_number = 20
    data_generator = data.LartpcData.from_path(data_path)
    result_dimensions = 3
    game = Lartpc2D(result_dimensions, max_step_number=max_step_number)
    vis = Visualisation(game)
    agent = BotAgent(
        game
    )
    for iterate_maps in range(30):
        map_number = np.random.randint(0, len(data_generator))
        game.detector.set_maps(*data_generator[map_number])
        for iterate_tries in range(10):
            game.start()
            for model_run_iteration in range(game.max_step_number):
                current_observation = game.get_observation()
                action = agent.create_action(current_observation)
                state = game.step(action)
                vis.update(0)
                if state.done:
                    break
```

Listing 15: Example of usage of Lartpc2D class. This function implements visualisation with a random agent.

flag, and empty string. The empty string is a space-holder for any additional information coming from the environment required by the openai-gym implementation. "Done" flag is true if the game has ended. This condition may occur if the agent has exceeded the allowed number of moves, or cannot make the action (steps outside the map). The reward is the float value as described in the previous section.

Observation2Dai consists of three windows in a given position W^S W^C W^T.

The main loop of the game works in the following way:

1. The observation of the game is made using the **game.get_observation()**.
2. The agent uses the observation with **agent.create_action(observation)** method, and creates an instance of **Action2Dai** object containing the action to take in the next step.
3. The game environment processes the action using **game.step(action)**. First, it applies the change specified in the classification, then it changes internally the position of the cursor, and returns a new state of the game.

8.4.3 Visualisation

The package also allows for visualisation of the game, implemented using PyQt. This feature allows for step by step basis watching of the progress of the algorithm, using the space key. The Fig. 8.4.1 contains

```python
@dataclass
class Observation2Dai(StaticTyped):
    source: NDArray[(typing.Any, typing.Any), np.float32]
    result: NDArray[(typing.Any, typing.Any, 3), np.float32]
    target: NDArray[(typing.Any, typing.Any), np.int32]

@dataclass
class State2Dai(StaticTyped):
    obs: Observation2Dai
    reward: float
    done: bool
    info: object

@dataclass
class Action2Dai(StaticTyped):
    movement_vector: NDArray[(1,2), np.int64]
    put_data: NDArray[(typing.Any, typing.Any, 3), np.float32]
```

Listing 16: Impelemntation of **Observation2Dai**, **State2Dai**, **Action2Dai**.

a screenshot of the visualisation running in 6 different windows. The top row of windows depicts the the parts of the maps visible to the the actor, and the lower row of windows shows the position of the actor on each of the maps.

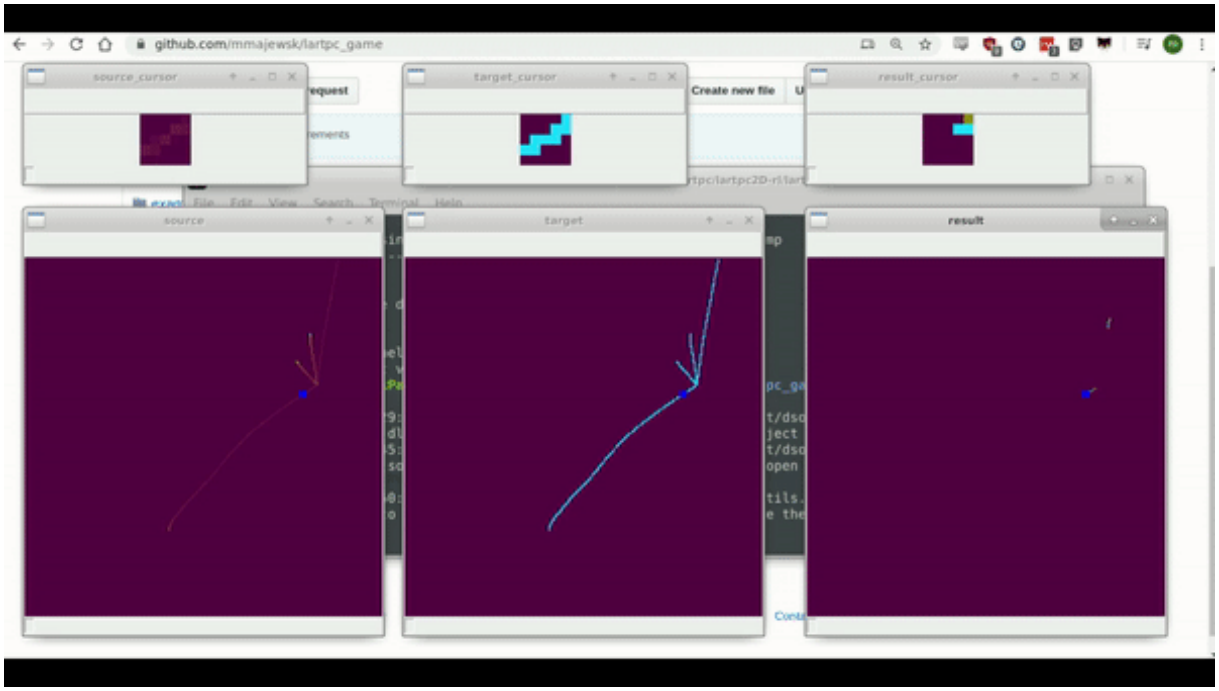
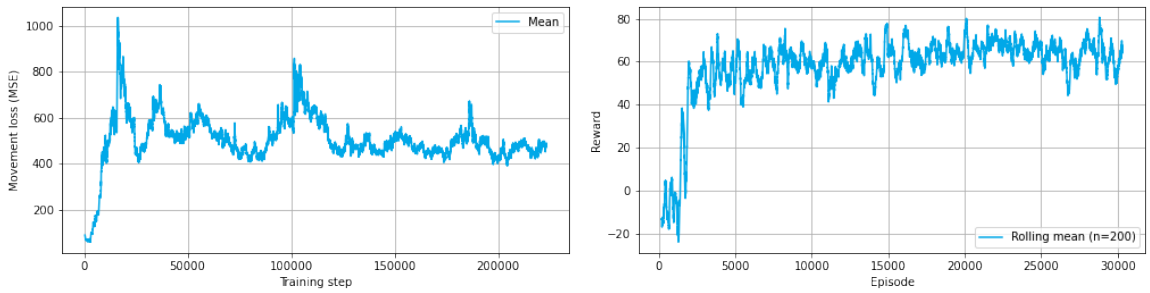


Figure 8.4.1: Visualisation in the gym-lartpc package.



(a) Mean square error loss **(b)** Reward curve

Figure 8.5.1: The learning process of the baseline DDQN.

8.5 Deep Q-learning solution

With all of the accompanying code of the lartpc game, it is much easier to brainstorm possible solutions. There is a vast range of reinforcement learning algorithms. Probably the two most popular approaches are those based on DQN (Sec. 3.5.2) and policy gradients *. The most successful solution to the problem posed by the environment in our research was DDQN, with Memory Buffer.

In the Fig. 8.5.2 the hyper-parameters of the model are defined. The environment, consisting of three maps, outputs an observation state, consisting of three smaller windows, and after each action provides the reward. The agent uses the *canvas* and *source* information in two blocks, called movement block and classification block. The higher level conceptualisation of the input and output to the agent can be seen in Fig. 8.5.3. The movement block is responsible for selecting the direction in which to move in upcoming action (Movement Decision). The classification block chooses the values that it should update the canvas window with (Canvas Output Window). Additionally, both blocks outputs undergo the operation of concatenation and are fed to a unifying block, which allows for the mixing of the information, and then creates appropriate outputs.

The classification block was pre-trained using a random sampling of the dataset images, with an appropriate window used as an output. It was pre-trained to the level of 86% of accuracy. The hyper-parameters of the network can be seen in Tab. 8.5.1.

The Fig. 8.5.4 presents the final version of the agent neural network. The Canvas window input, and source input are both binarised and concatenated together. Then they are presented as input to the movement block, which consists of 5 fully connected layers intertwined with ReLU activation function. The Source input is also presented as an input to the classification block. The classification block consists of smaller blocks of fully connected layer, batch normalisation, ReLU, and dropout. These blocks of four layers are repeated four times, ending with a fully connected layer that scales the output. Output from both blocks is concatenated and passed to three fully connected layers, the output of which is divided into Movement output and Classification output.

Parameter	Value
batch_size	128
droput_rate	0
input_window_size	5
outpu_window_size	1
result_outpu	3

Table 8.5.1: The parameters used for training the convolutional neural network for simple classification

Parameter	Value
gamma	0.99
batch_size	32
reaplay_size	10 000
learning_rate	1e-4
epsilon_start	3
epsilon_final	0.01
max_step_number	8
trials	8

Table 8.5.2: The parameters used for training the convolutional neural network for simple classification

8.5.1 Learning

The details of the training process are out of the scope of this description and may be found in the github repository**?** . The table 8.5.2 shows some of the parameters used for the training of the DQN model. From the training quality metrics, depicted in the Figure 8.5.1, one can see that the value of the reward function is growing, which proves that the model is maximising the reward. You can also see that the MSE loss function grows and then drops. We attribute this to a high epsilon value at the start and the memory buffer being filled with more uncomplicated episodes. Then the complexity of the task grows when the agent learns to explore, and then overall slowly decreases and stabilises as it learns. It is by no means trivial to evaluate the performance of a reinforcement learning agent by only basing it upon the loss metrics and rewards defined in the previous section. The difficulty of the problem may fluctuate, as some parts of the images are easier to solve than others, the actions of the agent may influence future difficulty, and the epsilon-greedy * actions introduce randomness into the process. The evaluation by observation of the agent behaviour was necessary. We observed that the agent indeed was following the tracks of the particles, but in the case of a more significant number of possible steps, the agent tended to "escape" an area with only an isolated non-zero pixel visible.

*Policy gradient methods in contrast to DQN (or value based methods), do not seek a best action, given states, but learn the probabilistic mapping of from state to value

*epsilon-greedy algorithm chooses either an action generated by the model, or completely random action based on a ε probability

The proposed DDQN solution to the problem posed by the gym-lartpc environment is meant to serve as a baseline for future research. DDQN has proved to be capable of performing very basic navigation through the decay tree. The goal of this baseline study is to show how the gym-lartpc environment can be used. We invite other researchers to use our environment for future experimentation with reinforcement learning, which is openly available in the GitHub repositories**? ?** . The future course for this work may include removing canvas as the feedback input and using a recurrent layer in the network. The other exciting possible extension could be rewarding the model to predict the direction and classification not for the current step but rather for the next step. This prediction could be applied to the environment and then fed back as an input to the model, generating new predictions and forming a feedback loop. Suppose one would translate the data read from any other experiment to the same voxel representation and scaled similar energy deposits (maintaining the same domain of particles). In that case, it might be possible to reuse the same model, or expand the simulation data, to create a single model for different detectors.

The presented environment has the potential to be the groundwork for a more unified machine-learning-based model of physics coming from high energy physics experiments. A cross-experimental framework could shift a focus from researching particular branches of physics (neutrino physics, b-hadron physics, particle showers, etc.) to more holistic research that accounts for all of the effects met in these particular areas. A model suitable for multiple experiments could help us better understand the link between the parts of physics studied by those experiments. I believe that the reinforcement learning-based model also has the potential to be used for creating simulations, as it may be computationally less expensive.

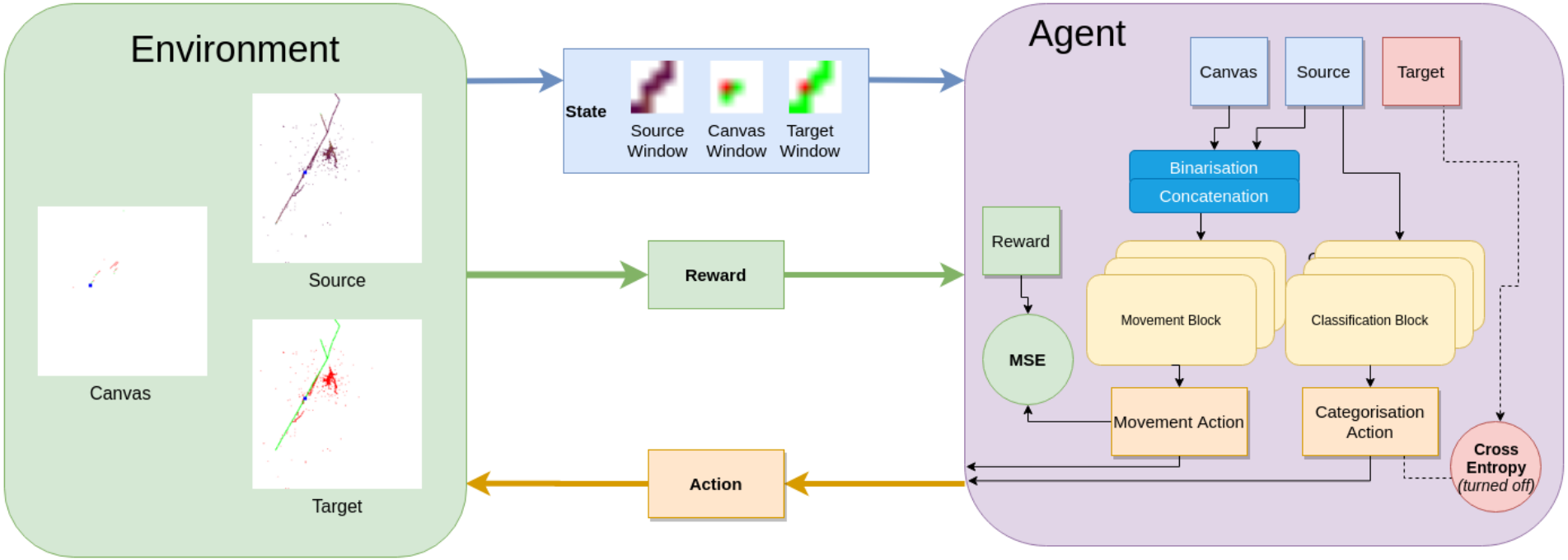


Figure 8.5.2: Reinforcement learning setting for learning the DDQN algorithm.

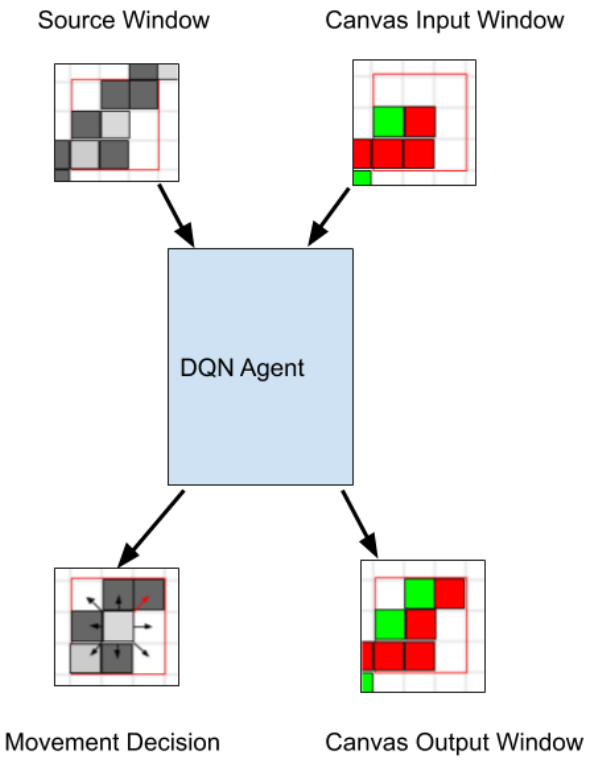


Figure 8.5.3: Exemplary input and output states for the neural network used in the analysis. The network receives both Source and Canvas input windows, and outputs a movement and put actions.

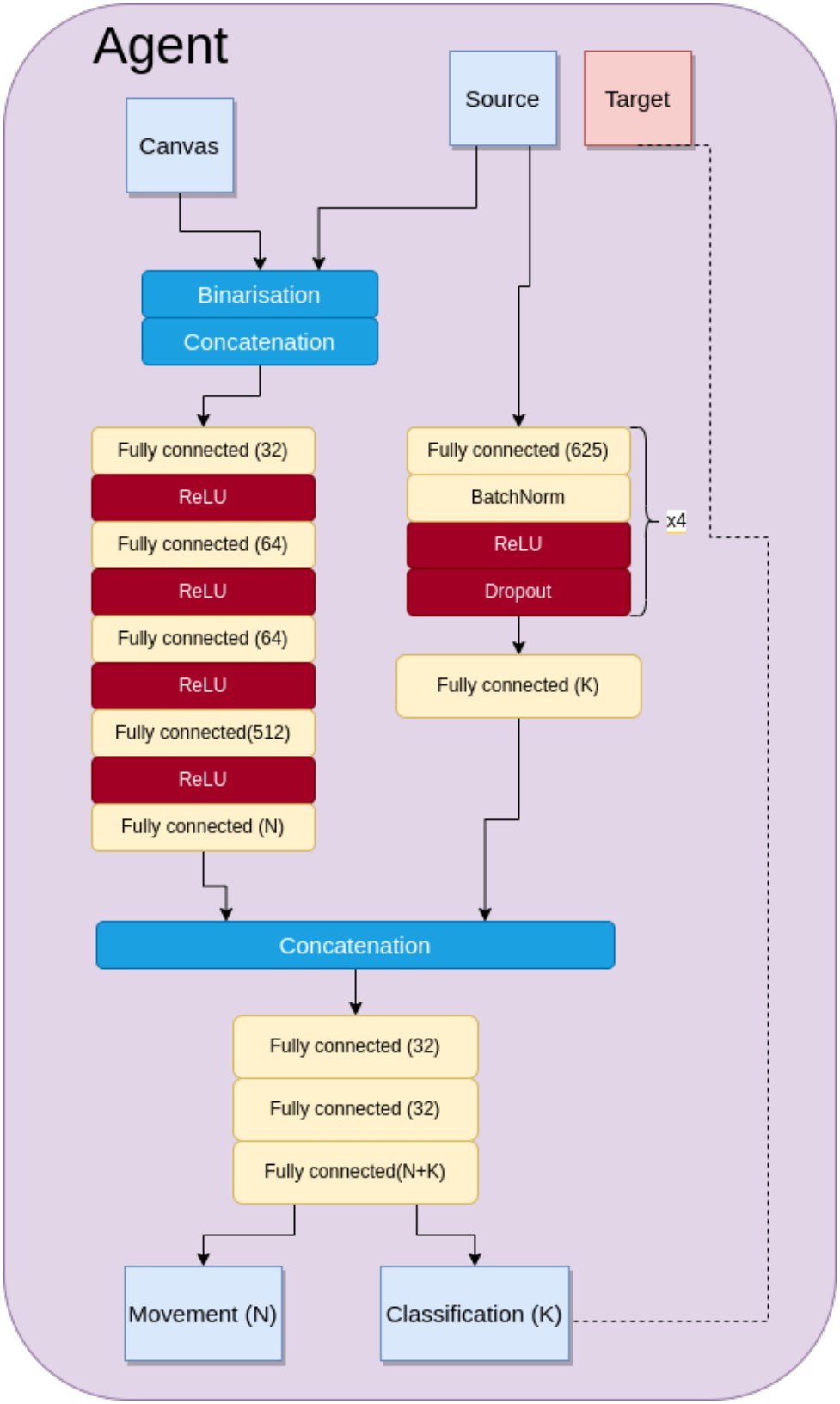


Figure 8.5.4: The complete model of the employed agent neural network. Source and Canvas input windows enter the movement block after binarisation and concatenation. The Source input alone also enters the classification block as is. The movement block returns a vector of size N. N is the number of possible movement directions (usually N=8). The classification block returns a vectorised matrix of classified pixels, of length K. Usually K=1 or K=9. Both blocks outputs are then concatenated and passed to the joint layer, which outputs a vector of size N+K.

9 Conclusions

This thesis combines multiple projects oriented around machine learning and software applications for high-energy physics detectors.

The projects related to Velo in runs 1 and 2 started with analysing the calibration data. This analysis inspired other projects and validation of detector maintenance. The essential information acquired in this process is that properly controlling the bias voltage and other factors have primarily protected the detector from continuous radiation damage.

Importantly Pedestal calibration parameter has shown no overall trend. Some of the effects stemming from the manufacturing of the sensors (faulty channels and coupling to the digital clock line), as well as the progression of the masked channels in time, have been noticed and discussed. Overall, these effects were expected to be found.

Another insight was that the Threshold parameters, which play a crucial role in hit detection, are prone to improper calibration. Such essential factor needs are guarded with care. That need has evolved into an anomaly detecting algorithm that is capable of early assessment of the calibration. This novel ”outlierness” algorithm was created using probabilistic programming and was capable of noticing anomalies and fluctuations in the calibration. The algorithm presented in the thesis was developed and deployed successfully within the Lovell monitoring framework in 2018.

Velo detector has a large number of individual channels (about 170 000 strips), which will be only increased in Run 3 (about 3 500 000 pixels). This makes it impossible for a human to assess the state of the detector at one glance. Hardships in visualising such high dimensional data led to an investigation of the dimentionality reduction methods (PCA and autoencoder), which have shown promise in conveying crucial insights from the calibrations.

Calibration is vital for the detector and demands special conditions (no-beam). The beam at LHC is

centrally steered and shared with other experiments. Thus scheduling of calibration must be deliberate. The comparison of the noise read out in during beam collisions and the Threshold parameters have shown a connection. In this unique study, a recurrent neural network model capable of predicting the need for calibration can help with the scheduling.

Although testing the application of the methods developed for the Velo in Run 1 and 2 on the Velo in Run 3 is impossible (as it is still in commissioning), this presents studies for the VeloPix, based on the insights from the test beam.

The VeloPix sensor was made to be a careful eye that will recognise the particles passing through and reconstruct their path. However, this eye can have blind spots due to the masked pixels. The study presented in this thesis utilises a simulation of VeloPix masked pixels to devise a breakthrough method for associating the clusters of the masked pixels throughout the calibration. The methods utilise a similarity matrix made of custom cluster features and tests both DBSCAN and OPTICS for clustering. The resulting algorithm (using DBSCAN) shows 76% of accuracy in cluster recognition.

One of the key metrics for monitoring the detector is the fluence level. The methods used so far, which include the geometry simulation, are not trivial in calculation and are burdened with low precision. The study of the relationship between the surrogate function parameters and the fluence in the test beam data has shown a correlation. This opens a door for a novel method of assessing the fluence levels experienced by the sensor. We present an approximation of this relationship as a polynomial fit.

An experience of close work with the detector calibration has led to the creation of a framework solution to two of the challenges of maintaining the detector: calibration monitoring and calibration data storage. The Titania framework helps to navigate the obstacles created by the monitoring tasks. It enforces structured writing of the code needed for plotting and exploring the data. The Storck database system solves the other problem: calibration data storage. Simple disk space on the filesystem has proved to be a not sufficient way of storing calibration data. This kind of data requires a structure, and it is a description while not limiting the end user to any type of data. The Storck project provides all of those features and is being deployed along the commissioning works for the upcoming Run 3. Both of the projects were created using an open-source MIT licence.

The last project in this thesis diverges from the LHCb and touches the LARTPC detector. It is an innovative approach to the problem of tracking and particle identification of neutrino-induced processes in the LARTPC detector simulation using Reinforcement Learning. It presents an environment interface for experimentation and implementation of the Reinforcement Learning based methods, along with the neural network model (DDQN) for baseline solution. This study shows that this approach is a viable choice for further investigation in this area.