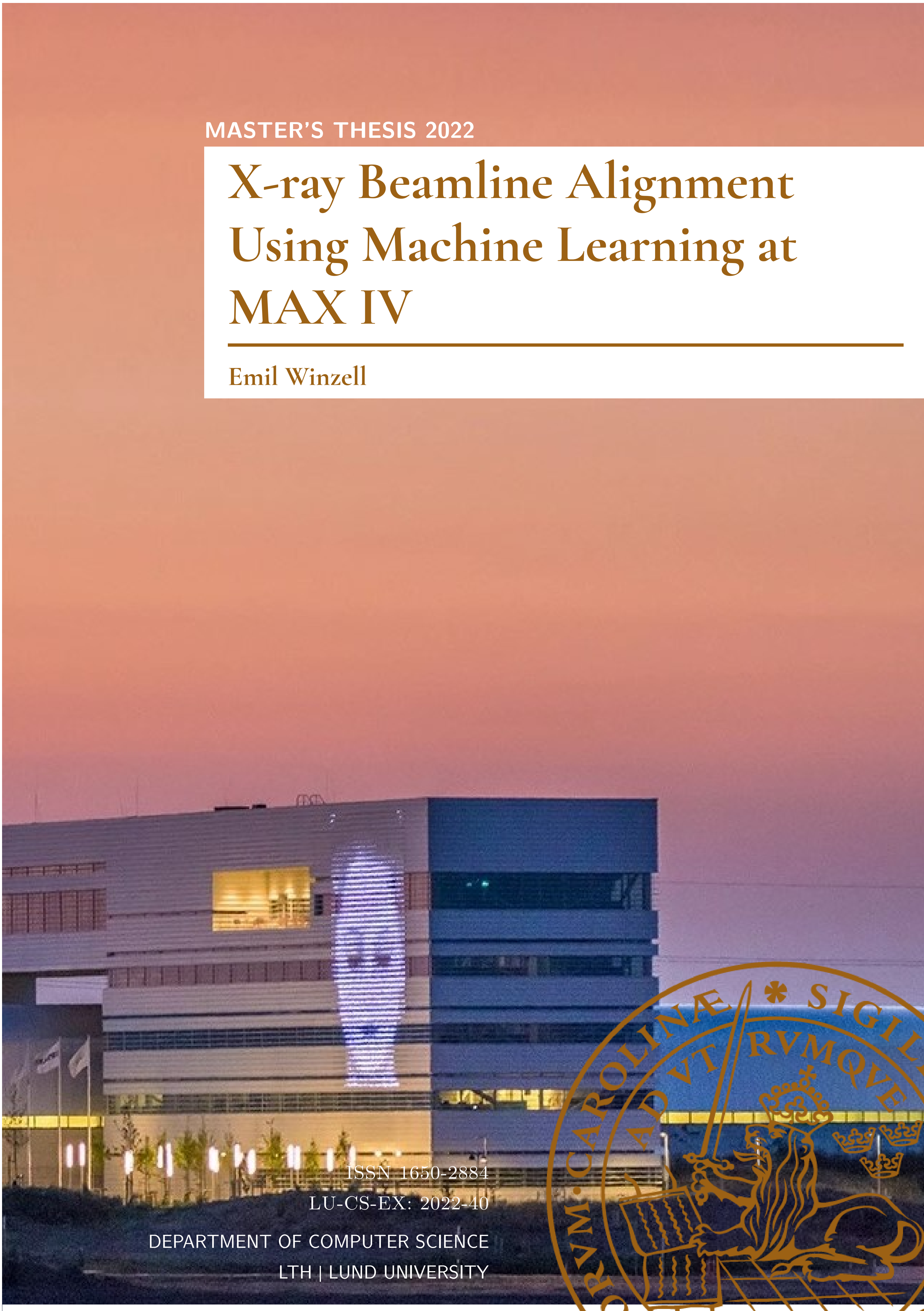
MASTER'S THESIS 2022

X-ray Beamline Alignment Using Machine Learning at MAX IV

Emil Winzell



ISSN 1650-2884
LU-CS-EX: 2022-40

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2022-40

X-ray Beamline Alignment Using Machine Learning at MAX IV

Emil Winzell

X-ray Beamline Alignment Using Machine Learning at MAX IV

(Master Thesis)

Emil Winzell
`emil.winzell@gmail.com`

June 23, 2022

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Elin Anna Topp, `elin_anna.topp@cs.lth.se`
Louisa Pickworth, `louisa.pickworth@maxiv.lu.se`

Examiner: Marcus Klang, `marcus.klang@cs.lth.se`

Abstract

The alignment process of beamlines at MAX IV takes up a lot of the scientists time, which instead could be used for experiments. In the beamline, mirrors are modeled at micro scales in order to produce the correct light. This thesis explores possibilities and pitfalls in using machine learning for aligning the optical elements of a beamline. Three different approaches are evaluated: Variational autoencoders, reinforcement learning and Bayesian optimization. Where promising results were obtained from the variational autoencoder and the Bayesian optimization, while the results from the reinfrocement learning methods were inconclusive. The autoencoder managed to quite well reconstruct and create new data, and the optimizer could improve a poorly aligned beam substantially. The project contributes with valuable insights for future work of this complex problem.

Keywords: MAX IV, beamline, alignment, machine learning, optimization

Acknowledgements

First I would like to thank my supervisors Louisa Pickworth and Elin Anna Topp who gave me the opportunity to do this thesis and who have helped me throughout the project. At LTH, I want to thank Alexander Dürr, for help with the VAE, and Erik Hellsten, for help with HyperMapper. At MAX IV, I would also like to give thanks to Zdenek Matej and Isak Lindhé.

Contents

Chapter 1
Introduction

This report presents work carried out at the MAX IV laboratory in Lund, Sweden. It includes an initial approach to applying machine learning in the beamline alignment process and suggests what work could be done in the future.

1.1 Background

Light is one of the most important tools for the scientific pursuit of understanding nature. From Röntgen's discovery of X-rays in 1895, and the later foundation of crystallography, which can unravel the atomic structure of materials, the use of electromagnetic waves in experiments have only increased. Scientists tool of choice has more and more become synchrotron radiation, which is the bright emission of photons when charged particles travel in curves and experience the centripetal force. These curves are called storage rings and can have diameters ranging from ten to several 100 meters [Altarelli and Salam, 2004].

MAX IV is the first of a new generation of storage rings, providing scientists with state-of-the-art X-rays ranging from 0.1 keV to 100 keV in photon energies. The facility consists of two storage rings which accelerate electrons at different energies, one at 1.5 GeV and one at 3 GeV. They are both fed by a 3 GeV linear accelerator which also connects to a short-pulse facility (see figure 1.1). The 1.5 GeV ring has a circumference of 98 meters and is used for lower photon energies (soft X-rays and ultraviolet light). The 3 GeV has a circumference of 528 meters and optimized for hard X-rays (greater than 5 keV in photon energy) . Every new project requires exact properties of the rays, which are achieved by sending them through the beamline. In the beamline, light travels through a series of precisely modeled, measured and aligned X-ray optics and aperatures [Tavares et al., 2014]. This project will be focused on the VERITAS beamline, which is located on the 3 GeV ring operating in the soft X-ray domain (see figure 1.1).

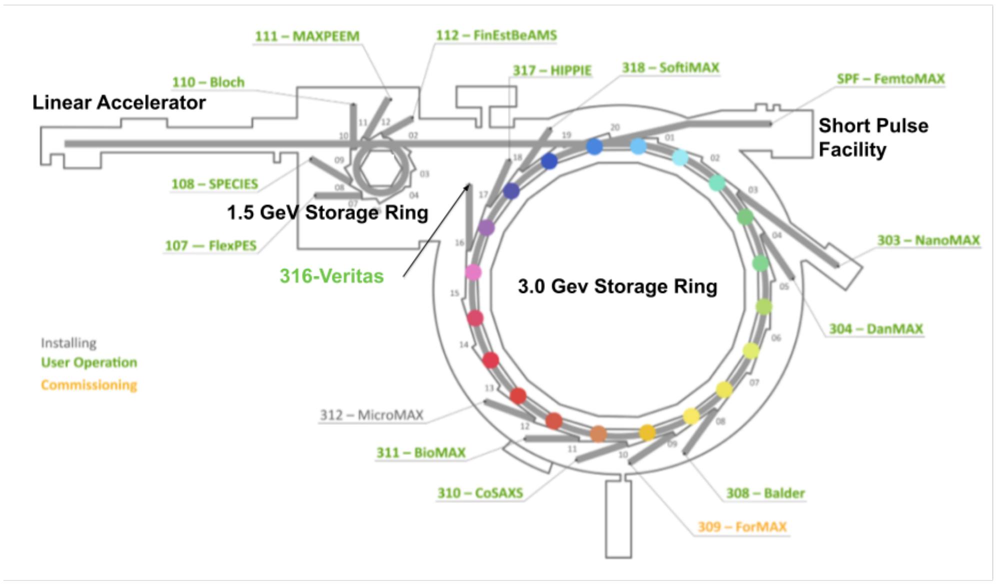


Figure 1.1: The MAX IV facility [MAX IV, nd]

Just like visible light, X-rays are electromagnetic waves with the only difference that the photons have higher energy and travel with down to a millionth of the wavelength of visible light. This will cause the X-rays to interact very differently with matter, and accordingly the optical elements will look very different from how they would look for visible light. Due to the high frequency of the X-rays they will have a refractive index very close to 1.0 for most materials, which means that they will penetrate without changing direction much. To aviod the X-rays from eventually getting absorbed they have to meet the conditions for total external refraction which occurs at very small “grazing" angles [Cremer, 2012]. To achieve small focal spots, the beamlines can become very long (the longest at MAX IV is over 100 meters) and they also have to be in vacuum since the rays can get easily attenuated in air. When well aligned, the light will have high energy resolution and hit the target sample at the smallest focal point. The placement and settings of the optical elements are typically designed through geometric ray tracing in computer simulations.

The mirrors of the beamline are operated in vacuum where all motorized axes are run by stepper motors. The motion is monitored by a set of linear absolute and incremental encoders. The steps that the mirrors are moved by are on scales down to 0.01 mRad and nanometers in linear motion. Vibrations, temperature and the state of the electron beam in the accelerator are all external parameters that will impact alignment and introduce an uncertainty in the setting of the mirrors. This means that the variable space for a single mirror is large and when multiple mirrors are introduced the alignment problem becomes complex. As there are practically no fixed, absolute reference points, the alignment process of the beamline can take many hours depending on the quality of light required. To achieve good performance, a beamline scientist will have to gain a lot of experience of working with the beamline which may take several months of daily tweaking to obtain. Today, the scientists spend a big part of their time on the alignment, time that they could be using to do the actual experiments. With the application of even more complex optical systems[1] the topic of automatic alignment with

[1]e.g. CoSAXS with 4 Kirkpatrick Baez mirrors [Kirkpatric and Baez, 1948]

computers is discussed frequently.

1.2 Aim

The area of machine intelligence has increased rapidly in recent years. Naturally, the question of *if and how machine learning can aid the optical alignment at MAX IV* rises. Which, given the complexity of the problem, is a very broad question and could never be answered fully by a master thesis project. The aim of this project was rather to answer it at least partially. Each of the 16 beamlines have different optical instruments and achieving a general model for all of them would be impossible. This project focuses thus on one beamline, VERITAS, and uses only simulated data for training and predicting. The input parameters are the physical settings of the optical elements and the output consists of intensity maps of the light (see section *Beamline and Simluation*). An ideal model would give answers to:

- *Where are we?* - Given the output intensity maps, what are the settings of the optical instruments in the beamline?
- *Where are we going?* - What should we change to achieve the desired focal point

Achieving a robust model, fully integrated to the operational beamline would have been optimistic for a master thesis project. This project was rather aimed at giving indications in what directions to move. What sort of models would be fit for the problem and what are their pitfalls and possibilities? This would contribute with valuable insights in using machine learning in the alignment process at MAX IV and pave the way for future work.

1.3 Methodology

The aim of the project was to conduct an empirical evaluation of a couple different machine learning approaches to the alignment problem. In order to be sure that the ML methods were well chosen, the first step of the project was to do a system identification.

- “What do we want to predict?”
- “How do the available data look?”
- “Which parts of the data are relevant for our purpose?”

were some questions raised. The next step was to select appropriate data and create a suitable dataset for training and evaluating our models (see section *Dataset*). Then the models were built, trained and evaluated. Based on the evaluations, conclusions were drawn to if the models were good or bad approaches to the problem and how this project could be built upon in the future. Three types of models were tested and evaluated: variational autoencoder, reinforcement learning and Bayesian optimization.

1.3.1 Contribution

This thesis contributes with an initial approach to achieving an automatic alignment process at the MAX IV facility. It shines a light on the three methods, how they could be implemented for the problem and how well they work. The groundwork for setting up a machine learning algorithm is done, with a system identification, data analysis and creation of data sets. Suggestions for future work are made along with ideas of how the project could be scaled up. This project is an initial look into a field which could have big impact on, not just MAX IV, but for storage rings in general.

1.3.2 Outline

The basic concepts and theory behind the methods are presented in chapter 2 along with prior work on the subject. In the next chapter (3) the approach to the problem is addressed, with details about the beamline and data in section 3.1 and how the architectures of the different models were designed. Specific implementations and evaluations of the models performance are presented in the following chapter (4). Lastly, conclusions and ideas for future work are presented.

Chapter 2
Theory

In this chapter all the relevant theory behind the evaluated methods to apply ML to the problem of beamline alignment is presented. It includes the scientific ground from which this project is built upon with background information on where automation and machine learning have been attempted at other synchrotron facilities.

2.1 Prior Work

Machine learning in the calibration process had not been explored before at MAX IV. Similar projects have been done at other facilities, but since MAX IV is one-of-a-kind, finding prior work was difficult. [Nash et al., 2020] at RadiaSoft LCC in Boulder, USA presented work on a reduced model for beamline control. The paper was mostly about creating a model for simulating the beamline but in the final part, the model was used to form a ML algorithm. The algorithm learned the misalignments from intensity data following a simple mirror system using a CNN. They give ideas for future development of the model and plan to expand it to use more paramaters of the beamline [Nash et al., 2020]. The paper was seen as inspiring in constructing the Variational Autoencoder (VAE), in which both the encoder and decoder consists of convolutional layers (see section 2.2.1). [Maffettone et al., 2021] also at the NSLS-II, presented work about 'gamifying' the beamline using reinforcement learning methods. However, in their case, the focus is not on the alignment process but on analysing the samples of materials. They managed to classify bad seeds based on their scattering characteristics right in almost 100% of possible cases [Maffettone et al., 2021]. The paper showed the powerful potential of RL to increase the scientific output of beamlines and was an inspiration to try reinforcement learning on the alignment process.

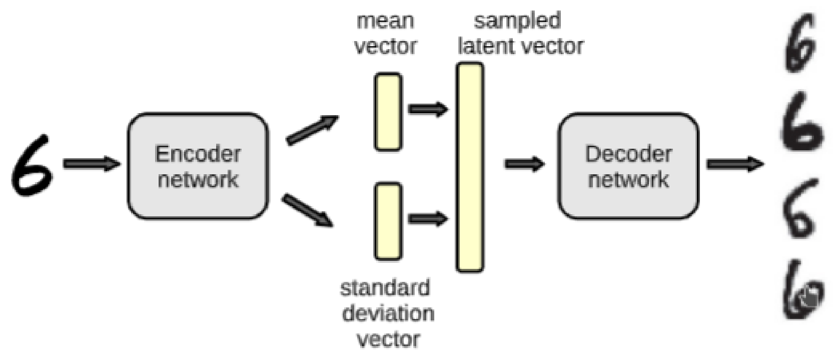


Figure 2.1: General design of a variational autoencoder [Ohlsson and Edén, 2020]

2.2 Variational Autoencoder

The variational autoencoder is a generative model which means that it can create its own data. The general design can be seen in figure 2.1. The VAE is a generalization of the autoencoder where the data is downsampled through a series of layers into the "bottleneck" (also called latent space) and then upsampled to the same size again. What is special about the variational autoencoder, compared to the regular autoencoder, is that the VAE learns a probabilistic distribution in the latent space which then can used to generate new data synthetically. The latent space is represented by a mean and standard deviation vector, which represents a normal distribution. The training is conducted in the following steps:

- A data sample is fed into the encoder network and downsampled to form the distributions of the latent space (the mean and standard deviation).
- From the resulting distributions, a point is sampled. This is the sampled latent vector.
- The sampled latent vector is fed into the decoder which scales it up to the same shape as the input.
- The reconstruction error between the input and decoded data is calculated and backpropagated through the networks.

After training, data can be sampled directly from the distributions in the latent space and then reconstructed in the decoder to generate new samples [Ohlsson and Edén, 2020].

2.2.1 Layers

A deep learning model consists of several layers with, in many cases, millions of nodes. The basic idea of how a node works can be seen in figure 2.2, weights are applied to the input values which are then summed up and put through an activation function. The activation function introduces a non-linearity and without it the network would just be a series of linear transformations which could be collapsed to one single layer [Ohlsson and Edén, 2020].

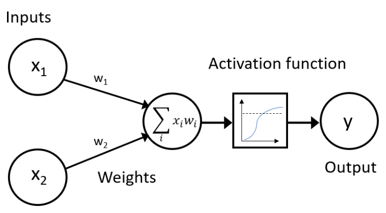


Figure 2.2: Basic idea of a node in a neural network [Bennström and Winzell, 2021]

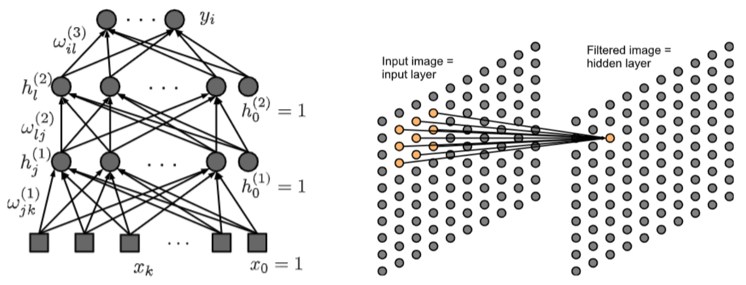
A popular activation function in deep learning is the rectified linear unit (ReLU), it just a ramp function

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}. \tag{2.1}$$

Some reasons of its popularity lies in the simplicity of implementation and that it has been shown to accelerate convergence. However, it can cause neurons to “die". The “dead neuron" problem can occur when a large gradient is flowing through the neuron which can cause the weight to update in a way that the output is never greater than zero. The output of the neuron then irreversibly always becomes zero due to the ReLU. A solution to this problem is the Leaky-ReLU

$$f(x) = \begin{cases} x, & x \geq 0 \\ kx, & x < 0 \end{cases}, \tag{2.2}$$

for some small, positive k. It allows the neuron to calculate the gradient even if the output is negative and prevents it from “dying" [Xu et al., 2020].



(a) MLP **(b)** Convloutional layer

Figure 2.3: Two common types of layers in a neural network [Ohlsson and Edén, 2020]

The layers of the VAE can be fully connected, dense layers (multi-layer perceptron, MLP) or convolutional layers. Usually the encoder and decoder networks consist of a combination of the two. An example of a multi-layer perceptron can be seen in figure 2.3a, all nodes are connected from one layer to the next and the information is passed through while weights are applied at each node. The convolutional layer can be seen in figure 2.3b, the input layer is convoluted by a kernel with a defined size. Each node in the hidden layer is connected to the same amount of input nodes as the size of the kernel. The size of the hidden layer is detmined by the size and stride of the kernel. For fast data manipulation, the weights of the kernel can be fixed and not trained, then the layer becomes a pooling layer. A common way of downsampling is using a max pooling layer which just takes the max value in the kernel for each node.

2.2.2 Loss

The loss plays a key role in any ML algorithm, it is through minimizing the loss the model “learns” by optimizing the weights. Any training dataset can be seen as a sample from a distribution $p(D)$, which, assuming indipendent samples, can be written

$$p(D) = \prod_n p(t_n, \boldsymbol{x}_n), \tag{2.3}$$

where $p(t_n, \boldsymbol{x}_n)$ is the joint distribution for the target t_n and input $\boldsymbol{x}_n$. The network produces an output $y(\boldsymbol{x}, \boldsymbol{\omega})$, where $\boldsymbol{\omega}$ denotes the parameters (weights) of the model. The probability of getting sample D from the model (given $\boldsymbol{\omega}$) is called the likelihood and gives

$$p(D|\omega) = \prod_n p(t_n, \boldsymbol{x}_n|\omega). \tag{2.4}$$

The learning process means basically determining the unknown parameters of the model by maximizing this likelihood [Ohlsson and Edén, 2020]. When working with images, which are big input spaces, the product of all the probabilities can be numerically difficult for the computer to handle and might lead to underflow. This is solved by taking the logarithm which converts the product to a sum. This leads us to our reconstruction loss function which is that we want to minimize the negative log-likelihood, we get the cross-entropy

$$E(\omega) = -\log p(D|\omega) = -\sum_n t_n \log(y_n) + (1 - t_n)\log(1 - y_n). \tag{2.5}$$

Ensuring zero-loss reconstruction for every possible data point is in most cases near impossible and will lead to overfitting, where some reconstructions will not make any sense. To prevent the VAE from overfitting, it is important that the distributions in the latent space are kept close to normal. Thus sampling from the latent space will give data that makes sense in the context. To enforce the normal distributions, a regularization term on the latent space is added to the loss function. The loss then consists of the reconstruction error and the regularizer. This regularizing term is expressed as the Kullback-Leibler divergence [Odaibo, 2019]. The KL divergence is defined as:

$$D_{KL}(p||q) = \int p(\boldsymbol{x}) \log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right) d\boldsymbol{x}, \tag{2.6}$$

which, when both p,q are Gaussian distributions can be simplified to

$$\begin{aligned} D_{KL}(p||q) &= \int p(\boldsymbol{x}) \log p(\boldsymbol{x}) d\boldsymbol{x} - \int p(\boldsymbol{x}) \log q(\boldsymbol{x}) d\boldsymbol{x} \\ &= \frac{1}{2} \log(2\pi\sigma_2^2) + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}(1 + \log(2\pi\sigma_1^2)) \\ &= \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}. \end{aligned} \tag{2.7}$$

In our case, the distribution $q(\mu_2, \sigma_2^2)$, which we are comparing to should have zero mean and unit variance. This gives

$$D_{KL}(\mu_1, \sigma_1^2) = -\frac{1}{2}(1 + \log \sigma_1^2 - \mu_1 - \sigma_1^2). \tag{2.8}$$

The total loss is then taken as a sum of the cross-entropy and KL-divergence [Bishop, 2006].

To update the weights (while minimizing the loss) methods using gradient descent are common, i.e.

$$\Delta\omega_i = -\eta \frac{\delta E}{\delta \omega_i}.$$

However, for efficient and reliable optimization the update formula has to be a bit more advanced. In this project the adaptive moment estimation (Adam) optimizer will be used in all VAE and RL models. The method runs averages m_k and v_k, which depend of the gradient g_k as:

$$\begin{aligned} m_k(t+1) &= \beta_1 m_k(t) + (1 - \beta_1) g_k(t) \\ v_k(t+1) &= \beta_2 v_k(t) + (1 - \beta_2) g_k^2(t). \end{aligned}$$

With $\hat{m}_k(t+1) = m_k(t+1)/(1 - \beta_1^t)$ and $\hat{v}_k(t+1) = v_k(t+1)/(1 - \beta_2^t)$ we get the update formula:

$$\Delta\omega_k(t+1) = -\eta \frac{\hat{m}_k(t+1)}{\sqrt{\hat{v}_k(t+1)} + \epsilon}. \tag{2.9}$$

Typical parameter values are $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$ [Ohlsson and Edén, 2020].

2.3 Reinforcement Learning

Reinforcement learning is a specific area of machine learning which consists of an abundance of methods for various tasks. The basics of a RL model is that we have an *agent* which takes *actions* in an *environment*. The environment gives the agent rewards and penalties as a result of the outcome of the agent's actions. The agent then tries to learn how to maximize its rewards. The way the agent acts at a given time is governed by the policy. The policy is a mapping from the observation space to the action space and can be anything from a simple look-up table to complex computations. The policy makes up the core of a reinforcement learning agent with the general goal to find the optimal policy for completing the given task [Sutton and Barto, 2018].

2.3.1 Policy Gradient Methods

Policy gradient methods is a set of RL methods that learn a parameterized policy without consulting a value function[1]. This means that the model predicts the action directly instead of first predicting the reward. The policy is based on the expected future rewards from the action using gradient descent. Some policy gradient methods are methods that also learns an approximate value function, these are called *Actor-Critic* methods. The actor part is a reference to the learned policy and will give probabilities for each action in the action space. The critic part is the learned value function which estimates future rewards. An advantage of policy gradient methods is that they can deal with continuous action spaces [Sutton and Barto, 2018]. The *Deep Deterministic Policy Gradient* method is an extension of the *Actor-Critic* which allows the use of continuous action spaces. The DDPG also has actor and critic with the same responsibilities. However, it also has two target networks which are updated slowly. The target networks give stability to the learning process and allows the model to learn a bit at a time instead of trying to learn the whole policy for every move. How fast the targets are updated is governed by the hyperparameter τ. A discount factor, γ, is used for weighting the immediate next predicted states higher than the ones far in the future. The following pseudo code describes the algorithm: [Lillicrap et al., 2016]

Algorithm 1 DDPG Algorithm

```
Randomly initialize critic $Q(s, a)$ and actor $\mu(s)$ networks
Initialize target network $Q'$ and $\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random noise process $N$ for exploring the action space properly
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu(s_t) + N_t$ according to current policy and exploration noise
        Execute action $a_t$ and retrieve reward $r_t$ and new state $s_{t+1}$ from environment
        Store the transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random batch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu(s_i + 1))$
        Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i^N (y_i - Q(s_i, a_i))$
        Update the actor using the sampled policy gradient
        Update the target network weights, $\omega$:
            $\omega^{Q'} \leftarrow \tau\omega^{Q} + (1 - \tau)\omega^{Q'}$
            $\omega^{\mu'} \leftarrow \tau\omega^{\mu} + (1 - \tau)\omega^{\mu'}$
    end for
end for
```

The state can consist of images and the actor and critic networks can have convolutional layers, in this project they were implemented as dense layers (see section 2.2.1). ReLU was

[1]A value function returns the expected reward from doing a certain action, in a certain state and then following the current policy [Sutton and Barto, 2018]

used as activation function and Adam as the optimizer. The noise process N is important for forcing the model to explore the action space properly.

Another policy gradient method is the *Proximal Policy Optimization* [Shulman et al., 2017], which has been shown to perform well on continuous control tasks. It is an extension of trust region methods in which the optimal policy is found though minimization of a surrogate loss function constrained in a region of trust [Shulman et al., 2015]. In PPO the region of trust constraint is dropped for a penalty on the loss function. Given policy π with weights θ, we denote the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}.$$

The surrogate loss is then defined as

$$L(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta),\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\right)\hat{A}_t\right] \tag{2.10}$$

where ϵ is a hyperparameter and $\hat{A}_t$ is the estimation of advantage (future rewards) for timestep, t. The expectation $\hat{\mathbb{E}}_t$ indicates the empirical average over a finite batch of samples. The clipping in the second term means that we penalize changes to the policy that move $r_t(\theta)$ away from 1, thus keeping the changes to the policy small. The algorithm is decribed through the following pseudo code [Shulman et al., 2017]:

Algorithm 2 PPO Algorithm

```
Initialize policy parameters $\theta$ for policy $\pi$
for iteration=1,2,... do
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
    Compute advatage estimates $\hat{A}_1, ..., \hat{A}_t$ through some method of advantage estimation
    Optimize surrogate $L$ w.r.t. $\theta$ with $K$ epochs and minibatch size $M \leq NT$, typically via Adam
    Update policy parameters $\theta_{\text{old}} \leftarrow \theta$
end for
```

2.4 Bayesian Optimization

Bayesian optimization is a class of optimization methods, based on machine learning. The BOA (Bayesian optimization algorithm) is derivative free for global optimization of black-box[2] problems. BOA is typically well-suited for objective functions that take long time to evaluate and for which only a couple of hundred evaluations is possible. The algorithm is focused on solving the problem:

$$\max_{x \in A} f(x) \tag{2.11}$$

Where A should typically be a hyperrectangle $\{x \in R^d : a_i \leq x_i \leq b_i\}$ and $d \leq 20$. f should also be a continuous function, in the sense that any two values x_1 and x_2 which are

[2]Where the structure of the objective function and/or constraints is unknown

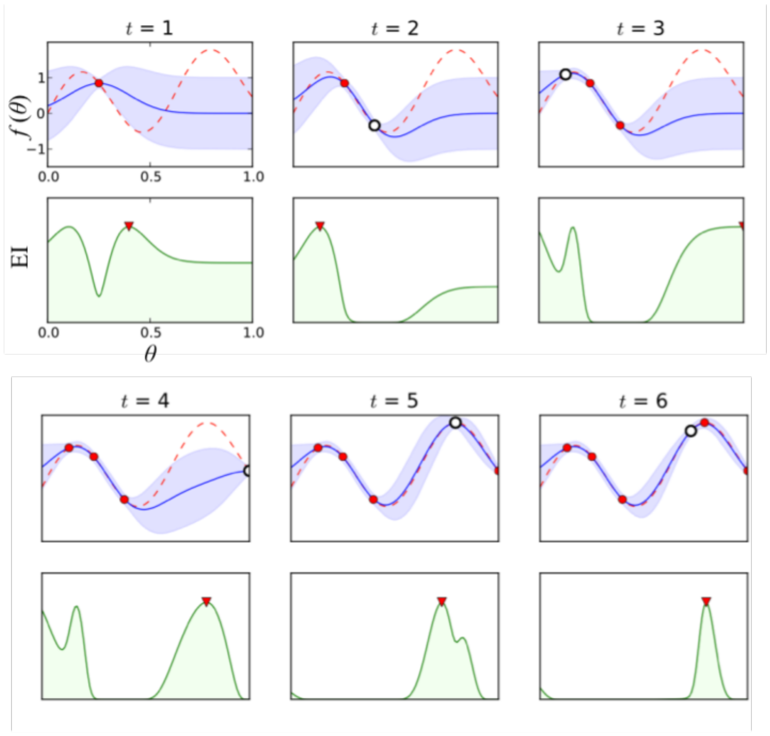


Figure 2.4: Example of objective and acquisition functions through iterations of Bayesian optimizaion. The red, dashed curve is the objective. The blue is the approximated function with the shaded region being the Gaussian process posterior. The green curve is the expected improvement.

close should mean that $f(x_1)$ and $f(x_2)$ also are close. The BOA works in two steps: first, a Bayesian statistical model is applied to model the objective function, then an acquisition function decides where to evaluate next. For the statistical model, a Gaussian process is used to provide a posterior probability. The posterior is updated every time a new observation of the objective function is made. The acquisition function measures evaluation of the objective function based on the posterior distribution. There are many alternatives for the acquisition function, the most commonly used is the expected improvement. We define it as:

$$\mathrm{EI}_n(x) := E_n[\max(f(x) - f_n^*, 0)] \tag{2.12}$$

Where $f(x)$ denotes a new evaluation of the objective function at x, f_n^* is the current observed maximum value. $E_n[\cdot] = E[\cdot|x_{1:n}, y_{1:n}]$ is the expectation taken under the posterior distribution given previous evaluations of f at $x_1, ..., x_n$. The posterior is a Gaussian process with mean $\mu_n(x)$ and variance $\sigma_n^2(x)$. The expected improvement is evaluated in closed form using integration by parts, resulting in the following expression:

$$\mathrm{EI}_n(x) = \max(\Delta_n(x), 0) + \sigma_n(x)\phi\frac{\Delta_n(x)}{\sigma_n(x)} - |\Delta_n(x)|\Phi\frac{\Delta_n(x)}{\sigma_n(x)} \tag{2.13}$$

Where $\Delta_n(x) = \mu_n(x) - f_n^*$ is the expected difference in quality between the new point and the previous best. The algorithm then evaluates at the point with the largest expected improvement. A simplified example can be seen in figure 2.4. At t=1 an initial guess is made. At that point we know the exact value of the objective function and will have low uncertainty around it, the expected improvement is low for points that are close. The acquisition function gives the point with the highest probability of improvement and we sample there. The algorithm keeps iterating and eventually finds the maximum [Frazier, 2018].

2.4.1 HyperMapper

HyperMapper 2.0 is a optimization tool, based on Bayesian optimization [Nardi et al., 2019] . It is multi-objective which gives the opportunity to optimize functions with more than one output, where there might be a trade off between them. Since it is based on Bayesian optimization it is derivative free, which is positive since derivatives can be tricky to work with. The mono-objective formulation of HyperMapper is the problem of finding a global minimizer of an unknown (black-box) objective function f:

$$x^* = \arg\min_{x \in \mathbb{X}} f(x) \tag{2.14}$$

where $\mathbb{X}$ is the decision space of interest and $f : \mathbb{X} \rightarrow \mathbb{R}$ is a deterministic function over a domain of interest that imposes lower and upper bounds on the input variables. HyperMapper allows the user to add prior knowledge on the distribution of the input parameters, e.g. if the user expects the optimal choice to be towards the middle but still wants to explore the whole domain the parameter can be set to follow a Gaussian distribution [Nardi et al., 2019].

Chapter 3
Approach

Here follows the approach taken to address the questions raised in section 1.2. Detailed information about the beamline and how the alignment process works is presented along with which data was chosen for the different methods and how they were designed.

3.1 Beamline and Simulation

The alignment process is aided though sophisticated computer simulations. This project was focused on simulated data where the MAX IV developed Python package *XRT* is used for all the simulations [Klementiev and Chernikov, 2014]. In *XRT* all parameters of the beamline are set and a specified number of rays are traced though the beamline. Anywhere along the beamline, 2D intensity maps and 1D histograms can be generated by counting each traced ray. The output from a simulation run can be seen in figure 3.1. The simulator is built upon geometric raytracing, which means that each individual ray will be traced through the beamline. In order to achieve realistic results a lot of rays[1] have to be traced through the system which will require computational power and time[2]. The real data will look different from the simulated since it is not possible to take images of the real beam, due to the small size. The 2D images have to be constructed from intensity data (see figure 3.5) and will have lower resolution (in the simulation there is no resolution limit). The 1D histograms of the beam intensity will, on the other hand, have comparable values (with some uncertainty due to temperature, vibrations etc.) to the simulated.

[1]10000 is acceptable but should preferably be more than 100000 rays

[2]≈ 5 s per tracing with 10000 rays running on several CPU cores

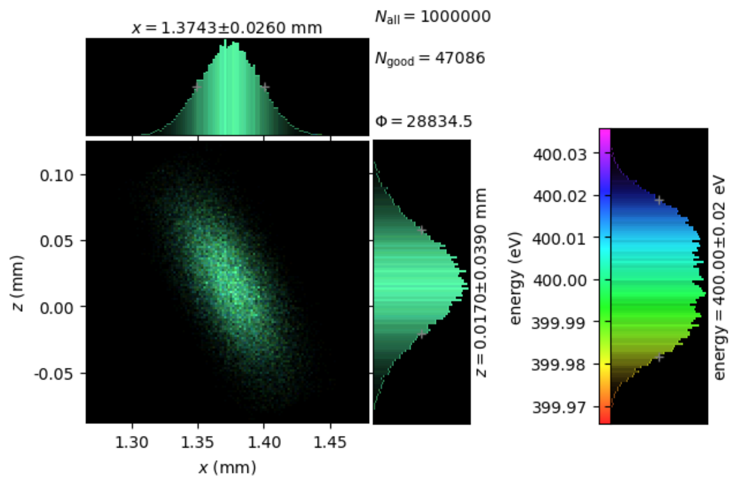


Figure 3.1: Typical output from a xrt simulation. The three plots to the left display the 2D intensity map and the two histograms on the axes (Notice that the z-axis here will be the y-axis in our coordinate system). The labels on the axes also shows the location of the beam with respect to the nominal axis through the target. The plot to the right shows the energy spectra of the beam, and looking at the colour of the beam in the 2D plot we can see that the different energies of the light is distributed evenly.

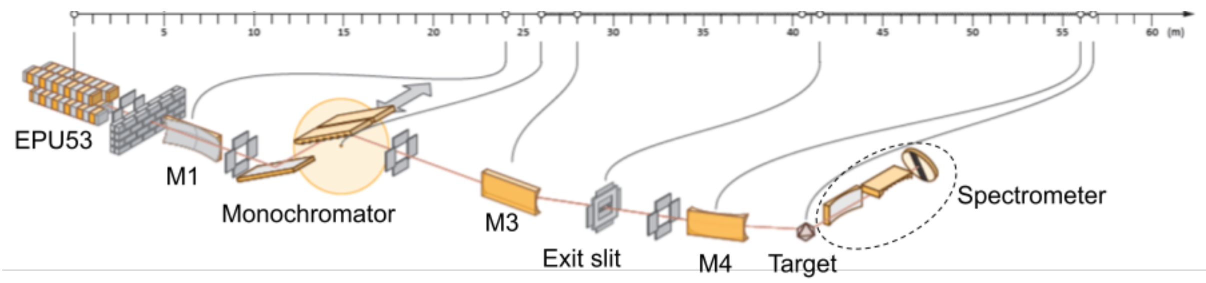


Figure 3.2: The optical elements and their position in the VERITAS beamline at MAX IV. EPU stands for elliptically polarazing undulator, which is the source from which the radiation from the storage ring is fed. The monochromator filters out different energy levels such that all light that hits the target is within the desired energy range. M1-M4 are the mirrors in the system. The beam hits the sample at the target and the resulting radiation is collected in the spectrometer for analysis [Samuli et al., 2017].

All simulations of this project were from the VERITAS beamline. VERITAS operates in the 275-1500 eV energy range (soft X-rays). It is almost 60 meters from the source to the target, the positions of the optical elements can be seen in figure 3.2. The experimental goal of the alignment process is to find the minimum focal spot size at the desired target position. The achieved focal spot from the real mirror depends on several factors: the position of the mirror, the size of the source and its angular distribution, the optical surface shape, size and the surface errors and the coating roughness. For this investigation the impact of the upstream

Table 3.1: Parameters and their limits of the M4 mirror

Parameter	Pitch	Yaw	Roll	Lateral transl.	Vertical transl.
Unit	mRad	mRad	mRad	mm	mm
Limit	±3.0	±1.0	±1.0	±1.5	±2.5

elements from M4 was not included. Similarly, the vertical size of the exit slit[3] was fixed in this investigation at 50 µm. The distance of M4 from the exit slit was also fixed in modeling (14500 mm) and the target position was defined by the ideal optical placement at 700 mm. The shape of the mirror is absolutely critical for the performance of the beamline. In this case the surface curvature was calculated from the given ellipse geometry (see appendix D for details), figure error and surface roughness parameters were added from measurements of the real surface. These measurements have an error of magnitude 1%, so even if they potentially could introduce a difference between the simulation and reality, it will be small. The shape parameters were fixed, thus leaving only the position of the mirror for exploration. This gave the system five degrees of freedom: Three angles of rotation, namely pitch, yaw and roll; two directions of translation, namely lateral and vertical. See figure 3.4 for a overview of the M4 mirrors placement in the coordinate system. By keeping all the upstream optics fixed, we are limiting ourselves to a fine tuning scenario, which will restrict the possible movement of the M4. The parameters were limited to prevent the mirror from moving off the beam. Each parameter and limit can be seen in table 3.1. The coordinate system that was used can be seen in figure 3.3, where the z-axis was fixed along the optimal direction of the beam. It is worth noting that the labels of the axes are different in *XRT* where the y and z have switched places.

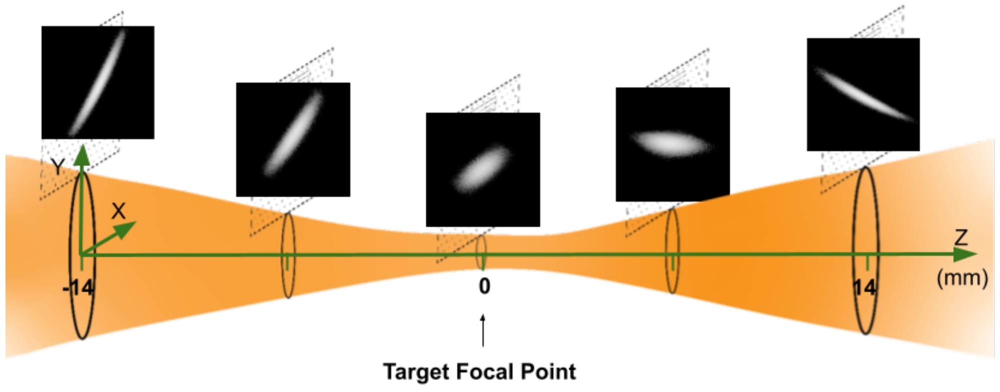


Figure 3.3: 3D visualization of the beam around the target, with xrt simulated 2D intenisty maps at the cross sections. The images are from a perturbed alignment.

[3]The exit slit acts here as a secondary source to be imaged onto the target by M4 (see figure 3.2)

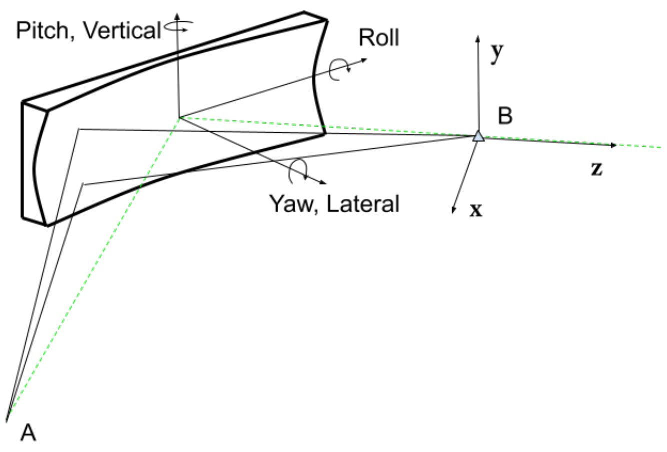


Figure 3.4: The position of the M4 mirror in the defined coordinate system. The green, dashed line represents the nominal axis directly from the mirror to the target (B). The five parameters are set as offsets from the pre-calculated ideal position which is handled by *XRT*, thus they have no meaning in the defined coordinate system in the image.

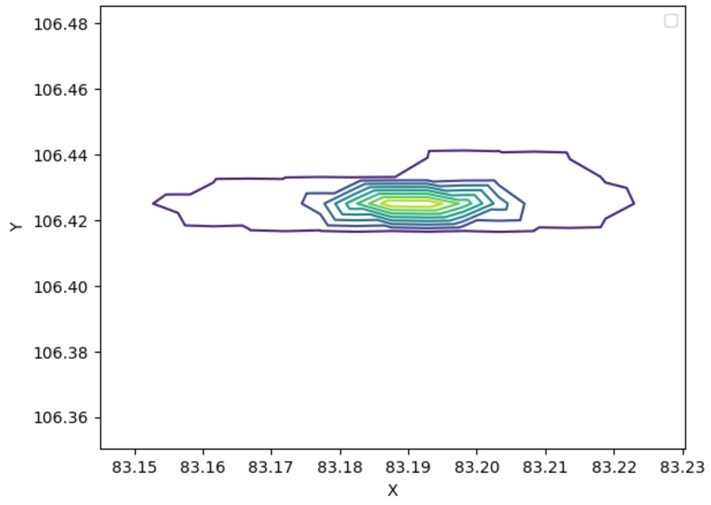


Figure 3.5: Resulting 2D image of beam intensity from a pinhole scan at the focal spot of the VERITAS beamline [Ekholm and Tokushima, 2022].

3.1.1 Dataset

In experiment, the scientists reconstruct the beam profile in three dimensions using a scanned blade or small pinhole which is moved through the beam. This is very different from in the simulation where the exact shape of the beam can be instantly retrieved by defining screens at desired positions, however, they can yield comparable properties. Just like in figure 3.3, screens were placed along the beam for recording data from the simulation. A total of nine screens[4] were used at equally spaced distances along the z-axis (y in *XRT*) starting from 14 mm ahead of the mirror focal point to 14 mm behind it. Each screen would record the following information about the beam: two 1D histograms of the intensity of the light along both x and y axis, one 2D intensity map of the beam on the screen. A total of 256 bins were used for both axes, resulting in images of 256x256 pixels, which was deemed to be an adequate resolution. The nine 2D maps of a data sample from one single ray tracing are displayed in figure 3.6. Corresponding histograms of the middle screen (at focal point) can be seen in figure 3.7. More 2D plots and scaling data can be seen in appendix B.

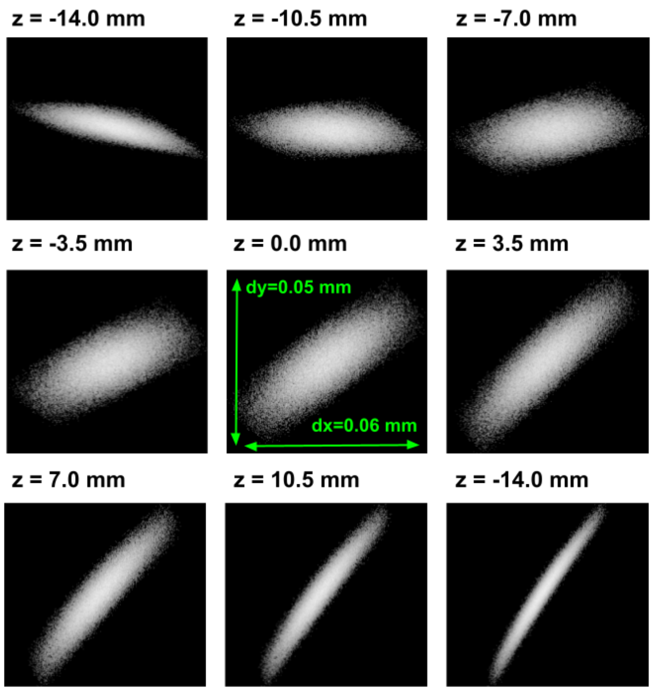


Figure 3.6: Result from simulation in *XRT*. The grayscale images of the 2D maps, later used in training, taken from 9 screens along the z-axis, where z=0 is the target focal point. All 2D maps are rescaled to fit to the image, the scaling of the middle image is printed as dx and dy. The alignment is slightly perturbed with offsets: pitch = 0.6 mRad, yaw = 0.7 mRad, roll = -0.9 mRad, lateral = -0.3 mm and vertical = 0.7 mm

[4]This would give us a good view of the beam in all dimensions, without having to save too many images per sample.

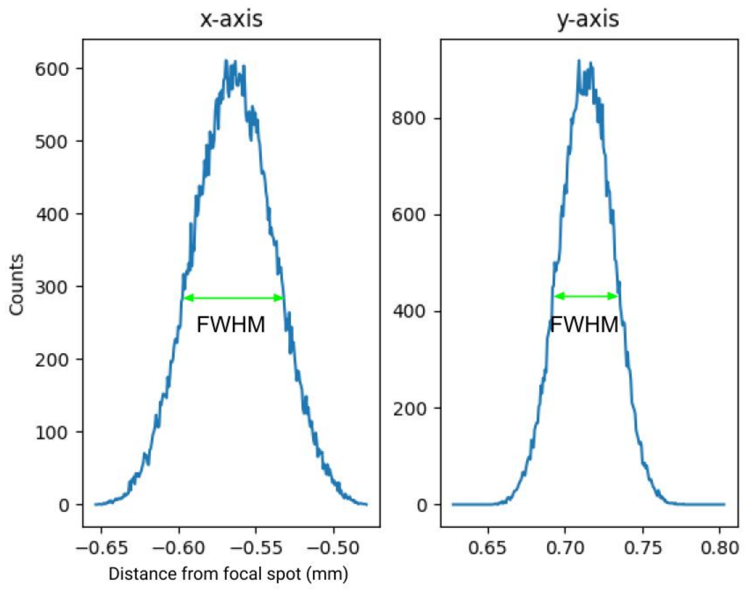


Figure 3.7: Full width half maxima calculated for the 1D histograms of the middle image (at z = 0.0 mm) in figure 3.6, from simulation in XRT.

The 2D maps were used as gray scale images for direct input into the desired model. The 1D histograms were not used as is, but could be reduced to one single number, without loosing any relevant information. The number is the *full width half maxima* which will be referred to as FWHM. The FWHM is the width of the peak at the half of the maximum, and is calculated for a gaussian curve as:

$$\mathrm{FWHM} = 2\sqrt{2\log 2}\sigma,$$

where σ is the standard deviation. An example is visualized in figure 3.7. Plotting the FWHMs along the beam provides us with valuable information about the quaility of the alingment in a very comprised way. One sample would consist of only 18 values[5], compared with the 2D maps which would add up to 256x256x9=589824 values.

Where and how big the impact of the beam on the target is varies a lot for different settings of the mirror. When perfectly aligned, the dot will be smaller than 0.01 mm in diameter, and when it is not, it could spread more than 2 mm. In addition to this the displacement in space could be as much as ±6mm in the lateral direction and ±3mm in the vertical. Thus, to contain all possible tracings within images of fixed sizes in space, the images would need to be around 16 mm wide and 10 mm high, such an image would need a resolution of 1600x1000 pixels for the perfectly aligned dot to be visible, even as one pixel. This clearly would not work, the networks would have to be very big to handle that big input space and would not be able to recognize one white pixel. The solution for this was that all images were re-scaled to fit only the beam within the image and we kept using images with a 256x256 resolution. The four[6] scaling parameters were then also saved with the images.

[5]FWHMs for both x- and y-axis with nine z-positions each

[6]Width, height and center location in space (x,y)

For training and evaluating the VAE a total of three data sets were used, two for training and one for testing. All sets were created by setting the mirror position at random settings and then tracing 100000 rays through the virtual beamline. The first training set was created with parameter limits slightly bigger than what is presented as reasonable in table 3.1. Yaw and roll were set between ± 20 mRad and the lateral translation between ± 5 mm. This was done to give the model a clear initial view of the differences in the data. Uniform distributions were used for the sampling of the pitch, yaw and roll offsets while the lateral and vertical translations were sampled from normal distributions. A total of 1064 samples were generated, resulting in 9576 different images for this set. For the second training set the limits were all reduced to the ones in table 3.1, uniform distributions were used for all parameters. A total of 366 samples were generated which gave 3294 images. In training, the sets were split 80-20 to training and validation. For accurate, unbiased evaluations you need a separate test set which the model never "sees" during training. The test set was generated using the same settings as for the second training set and consisted of 266 samples (2394 images). No pre-made sets were needed for the reinforcement learning or Bayesian optimization algorithms since the ray tracing was integrated into the learning process. A summary of the folder structure in which the data was ordered can be seen in figure 3.8. All information about the beamline and M4 settings for each sample was collected in the file named "data.xml", see appendix A for an example.

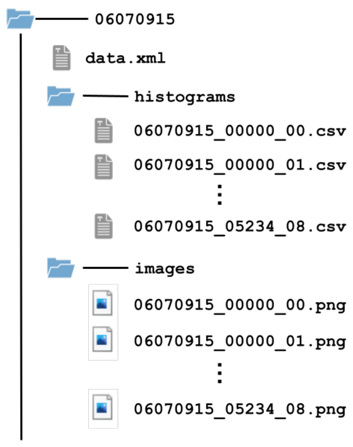


Figure 3.8: Structure of the data sets. Each data folder was named with a timestamp for when it was created, the histograms and images were saved with naming convention <timestamp>_<sample number>_<screen number>.

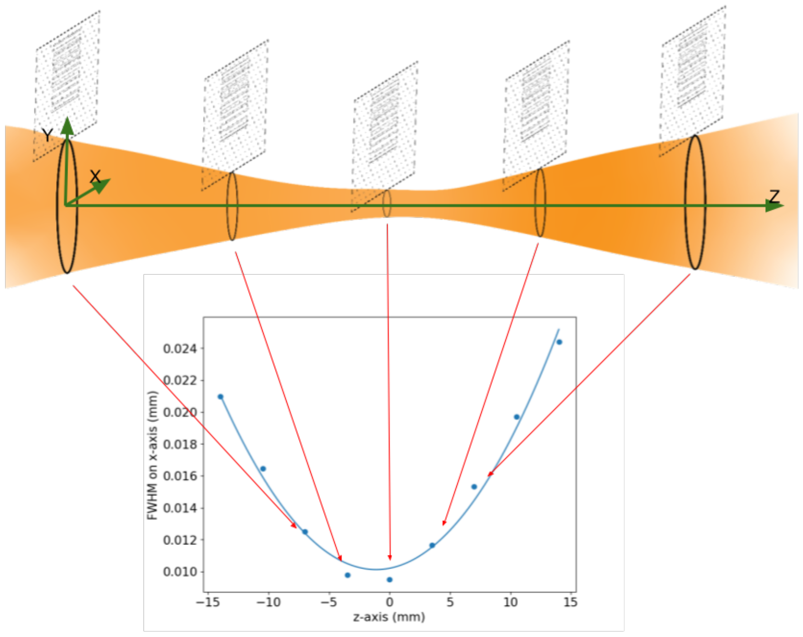


Figure 3.9: How the FWHMs of the beam from the x-axis are plotted around the target focal point. The image is from a well aligned beam.

3.1.2 Alignment

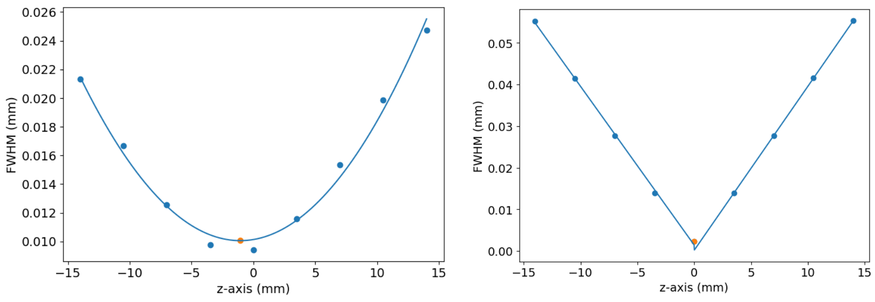
From looking at figure 3.4, the purpose of alignment becomes clear: we want to redirect the rays coming from the source (point A) to become centered again at the target (point B). This is done though reconstructions of the beam around the focal spot. Even if the process of reconstructing the beam is very different from the real to the simulated case, the measurement of quality of alignment is similar. The FWHM of the light intensity histograms plays a key role in both real and simulated cases. As the beam is reconstructed (see figure 3.9), the FWHM for the lateral and veritcal axes are calculated for points along the z-axis and collected in plots. We end up with two plots as in figure 3.10, where we want both of the minima close to the center. Typically, the points in the lateral direction should follow a second degree polynomial while in the vertical they should be in a V-shape. The plots in figure 3.10 display a well aligned beam. To quantify the measure of how good an alignment is, the following values could be used:

- The minimum value of the FWHMs for the x-axis (lateral).
- The minimum value of the FWHMs for the y-axis (vertical).
- The *gap* on the z-axis, between the occurrence of the two minimum FWHMs (x and y).
- The distance from the origin (target point) to the center of the current focal point of the beam).

In our case (for VERITAS beamline with one M4 mirror) the FWHMs should be below 0.01 mm and 0.005 mm for the lateral and vertical axes respectively. The gap should not be bigger than 1 mm (not a hard limit) and the origin should be within 2-3 standard deviations of the histogram for both axes. If all values are close to these boundaries the beamline can be considered well aligned, the parameters are presented in table 3.2. These boundaries result in constraints on the input parameters, as can be seen in table 3.3, the beam becomes well aligned at around 1% of the maximum offset.

Table 3.2: The parameters of the alignment space, their ranges and criteria for a beam to be considered well aligned. All parameters are in mm

Parameter	min. lat. FWHM	min. vert. FWHM	FWHM gap	Target distance
Range	$\sim 0.007 - 0.12$	$\sim 0.002 - 0.2$	$0 - 1000$	$0 - 5$
Well aligned	< 0.01	< 0.005	$\lesssim 1$	$\lesssim 0.2$



(a) The lateral x-axis **(b)** The vertical y-axis

Figure 3.10: FWHM plots from a well aligned beam. The blue dots are the FWHM values, the curves has been fit to follow their typical pattern. The orange dots are the calculated minima from the curves

To get an idea of the shape of the alignment space the five parameters of the M4 mirror were iterated over their ranges, while keeping the others at zero. Ray tracings were made at each step and the alignment parameters were calculated and presented in figure 3.11.

Table 3.3: Resulting parameters of the alignment space from different offset levels in the input parameters. The offsets are set as percentages of their maximum value (see table 3.1). Green values are below the well-aligned limit in table 3.2, red are above.

Offset	FWHMx	FWHMy	Gap	Target dist.	Well aligned
10%	0.0139	0.0049	11.69	0.3882	No
3%	0.0097	0.0066	4.010	0.1165	No
2%	0.0094	0.0056	0.0090	0.0776	No
1%	0.0093	0.0034	0.5175	0.0388	Yes

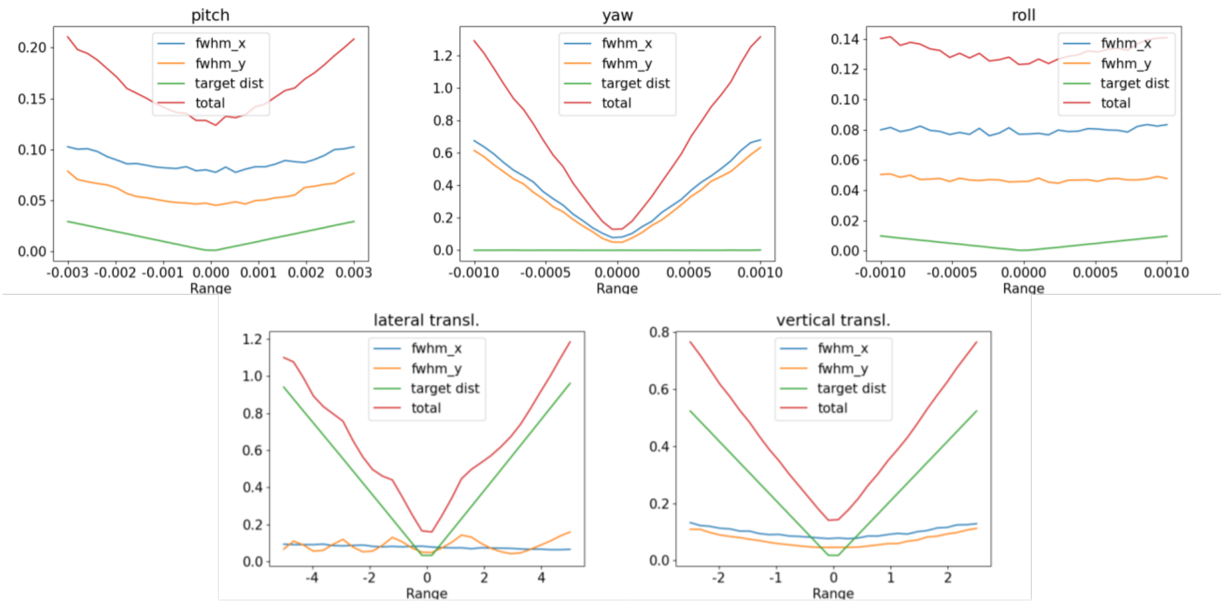
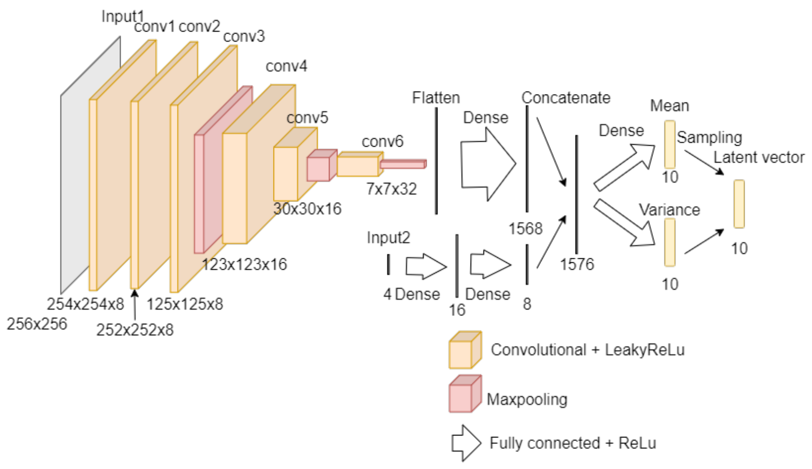
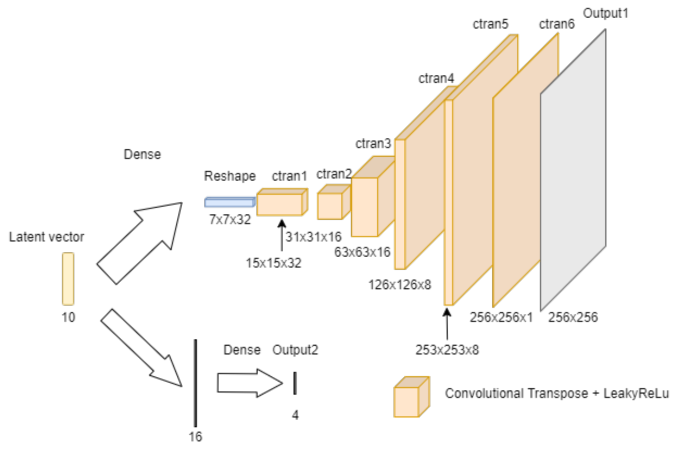


Figure 3.11: Plots of the FWHMs, distance from target and their sum, for the complete ranges of the five parameters of the M4 mirror. The output parameters have been scaled to produce similar values, which is why the y-axis was left unlabeled.



(a) Encoder



(b) Decoder

Figure 3.12: The architecture of the 2D VAE

3.2 Variational Autoencoder

A functioning variational autoencoder could be useful for our purposes in many ways. The generative aspect could help us to extend our data sets and to get new data samples much faster than doing the ray tracing. Data could also be stored in encoded format to save space. Another possibility is to map the latent space to the actual parameters of the mirror through another network and thus get an answer to the question *Where are we?*

The variational autoencoder can be implemented to use images in three dimensions which would be tempting for our problem since the beam contains valuable information in all directions. However, as an initial approach the VAE was implemented to take only 2D images. The implementation was adapted from [Chollet, nd] and the architecture was inspired by [Gad, nd]. The full architecture can be seen in figure 3.12 (see appendix C for a full summary). To avoid the "dead neuron" problem, Leaky ReLU was used as activation function for all layers, ADAM was used as optimizer (see section 2.2.1).

The attentive might realize that the model has two inputs and two outputs, one of which is the image of the 2D intensity map, while the other is not quite as obvious. The second term is due to the scaling issue discussed in the 3.1.1 Dataset section, that images of fixed sizes would be impossible due to the different sizes of the beam. The resulting re-scaling of the images means however, that a lot of important information is lost, both the spread and the location of the rays in space. Input 2 represents this lost information, it is a vector of 4 values: the x and y coordinates of the center of the image in the global coordinate system (the origin is at the ideal focal point) and the scale of the x and y axis i.e., how high and wide the image is in the global coordinate system. Input 2 is therefore very important for the model to have meaning.

To achieve good image reconstructions, we need the non-zero pixels to be correct. However, since a lot of the images are black, zero-valued pixels, we do not want to reward our model as much from getting them right. Therefore a weighting of the reconstruction loss was added where the term handling targets close to one was weighted much higher. This gives the weighted cross-entropy in equation 3.1 which was used for the reconstruction loss of the images [Brock et al., 2016].

$$\epsilon = -\frac{1}{N}\sum_{n}^{N} \alpha t_n \log y_n + (1-\alpha)(1-t_n)\log(1-y_n) \tag{3.1}$$

The VAE is regularized through a second loss term, the Kullback-Leibler divergence which keeps the probabilistic distributions close to normal with zero mean and unit variance (see section 2). An important part of designing a functional VAE is to have correct weighting between the KL-divergence and the reconstruction loss. If the KL-divergence is weighted too highly, the model will be heavily regularized and the decoded images will be averaged into the same image which fits to all the inputs. An example of how this would look for our case can be seen in figure 3.13. On the other hand, if the KL-divergence is too low, the model will overfit and a lot of the output will not have any reasonable connection to the input. The optimal weighting was found through trial-and-error.

In our case the model converged when the KL-divergence was multiplied by a factor of 0.1. The α parameter of the weighted cross-entropy (see equation 3.1) was set to 0.95, which means that 95% of the reconstruction loss would be constituted of pixels with values close to 1 (white). The value was chosen from balancing noise (too high) and neglected shapes (too low) in the reconstructions [Brock et al., 2016].

The network was first trained on the bigger dataset with 9500 images (see section 3.1.1 Dataset) for 200 epochs. Then it was retrained on the smaller, 3300 image, set for another 300 epochs. In the initial set, the optical parameters had bigger ranges than what was eventually decided as reasonable in our case. However, the data set was used as a pre-training step, to give the model a better initial view of the differences in a good and bad alignment. For both training sets, 20% of the data was taken for the validation set, which then is used for testing the model during training[7].

[7]The validation loss is what we try to minimize

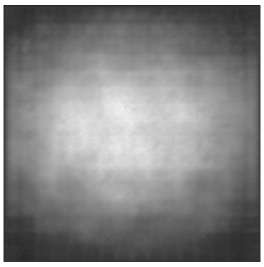


Figure 3.13: Decoded 2D intesity map from highly regularized VAE. All decoded images would have a similar appearance, independent of the sampled latent space

3.3 Reinforcement Learning

The idea behind the reinforcement learning models was to answer the question *where are we going?* To improve an alignment by tweaking the parameters of the optics. The first step in building a RL model is creating the environment. The environment will consist of an action space, an observation space, and a reward function. The action space is spanned by the parameters of the beamline and their limits, in our case the M4 mirror (see table 3.1). The observation space contains all the information that the agent 'sees' before choosing what action to do. The initial approach to the observation space was to use the quality measurements of the alignment (see section 3.1.2) i.e. give the model the minimum FWHMs, the gap between them and the distance from the target center. This would remove a lot of information about the beam away from the target, which could be important. Thus, the observation space was changed to take all 18 FWHM values from the screens. The reward function was determined from the alignment parameters in table 3.2

$$r = \begin{cases} 300 & \text{, if all parameters below alignment limit} \\ -f_x - f_y - g - d & \text{, else} \end{cases} \tag{3.2}$$

Where f_x,f_y denotes the minimum FWHMs, g the gap and d the distance from target center. In this way the agent would receive small negative rewards for each step that did not lead to a well aligned beam. The rewards would go towards zero as the agent got closer to the solution.

The learning process was set up as the following, giving one episode:

- Initialize mirror in random state within parameter ranges (see table 3.1)
- Let the agent evaluate the state and take steps to counteract the initial offset
- Calculate rewards from actions and update weights of the agent network that chooses what action to do next
- Let the agent keep taking steps until a well aligned beam is achieved, or a predefined limit of steps is reached
- Reset the mirror and try again

The algorithm was then run for a defined number of episodes.

Due to the nature of the problem, a model with discrete actions was deemed to be unfit. The size steps of the parameters would have to be very small which in turn would require a lot steps to find the optimum, and since each step requires a new ray tracing this would take very long time for the model to learn. Policy gradient models are good with dealing with continuous actions, of which two models were implemented.

The fist choice of model was the *Deep Deterministic Policy Gradient* (see section 2.3.1). The model architecture was taken from [Singh, nd] and modified to take multiple actions. [Lillicrap et al., 2016] used the Ornstein-Uhlenbeck process as the noise process, which generates a temporally correlated process, suitable for physical problems with inertia. The beam does not depend on previous settings, i.e. we have zero inertia, which is why another noise process was implemented. The second process was chosen as white (uncorrelated) noise from a normal distribution with zero mean and decreasing standard deviation. The algorithm was run for 20 episodes with a maximum of 1000 steps per episode. It resulted in 20000 ray tracings which took more that 24 hours to complete.

The second model was the *Proximal Policy Optimization*, of which the implementation was taken as is from Stable Baselines3. Stable Baselines3 is a set of open-source implementations of deep reinforcement learning methods in PyTorch. All implementations are well tested and have been benchmarked against reference codebases [Raffin et al., 2021]. The PPO model was loaded with policy model: “MlpPolicy", which means that the policy was decided through a multi-layer perceptron (see section 2.2.1). It was trained, without resetting the environment for 12000 timesteps (or until the conditions for a well aligned beam were met).

3.4 Bayesian Optimization

The Bayesian Optimization was done with the tool HyperMapper [Nardi et al., 2019] (see section 2.4.1). The same principle like in the RL models were used, where the mirror was set in a random state and the goal of the algorithm was to align the beam again by counteracting the offset. The input parameters were the M4 settings which were normalized with respect to their ranges. For simplicity, the objective function were chosen to only have one output value. The objective function has a crucial role in the Bayesian optimization and is the main design area. For a good optimization we want it to have a clear global minima with steep slopes leading to it, but since we are working in a multidimensional space visualizing it is hard. The metrics presented in 3.1.2 was a staring point in forming the function. The gap parameter was eventually dropped in the optimization since gave rise to a non-smooth curve with lot of local minima. It was just used as a validation for classifying the beam as well aligned. The optimization problem was then formulated as the following:

$$\begin{aligned}
\text{Minimize} \quad & q(p, y, r, l, v) = \alpha_1 f_x + \alpha_2 f_y + \alpha_3 t \\
\text{Subject to} \quad & |p| \leq 0.003 \\
& |y| \leq 0.001 \\
& |r| \leq 0.001 \\
& |l| \leq 1.5 \\
& |v| \leq 2.5
\end{aligned} \tag{3.3}$$

Where p, y, r, l, v denotes the pitch, yaw, roll, lateral- and vertical translations respectively, and f_x, f_y, t denotes the minimum FWHMs and the distance from the target (see section 3.1.2). α_i is a pre-defined weighing parameter, which was picked as normalization values towards the range defined in table 3.2. They were chosen using the graphs in figure 3.11 and such that their values would all fall in approximately the same range: $\alpha_1 = 8.3$, $\alpha_2 = 20$, $\alpha_3 = 0.2$. The prior knowledge of HyperMapper was kept at the default uniform distribution since the offsets also are picked from uniform distributions.

3.5 Evaluation

The variational autoencoder was evaluated on the separate test set with 2400 images. The images and their scales were fed into the encoder, then one sample was decoded from the latent space. The resulting lateral and vertical FWHMs and scales were compared, the resulting errors are presented in table 4.1. The FWHM errors are calculated as

$$e = \frac{|f_t - f_p|}{f_t},$$

where f_t is the true FWHM of the original image and f_p is the calculated FWHM from the reconstructed image. Along with the FWHM errors are also the absolute errors from the scaling parameters presented.

The reinforcement learning models were only evaluated through their average reward curves (which should increase) and plots of their actions taken. Since the models never achieved their goal (a well aligned beam) in training, no further testing was made.

The Bayesian optimization was evaluated though how well the found parameters and the original offset would cancel each other. The errors were calculated as the absolute value of the resulting offset and are presented in table 4.3 along with the objective function error. New tracings were made with the resulting offsets and the alignment parameters from table 3.2 were calculated, their averages are presented in table 4.4, with the results from the best points found by the DDPG as comparison.

Chapter 4
Implementation and Results

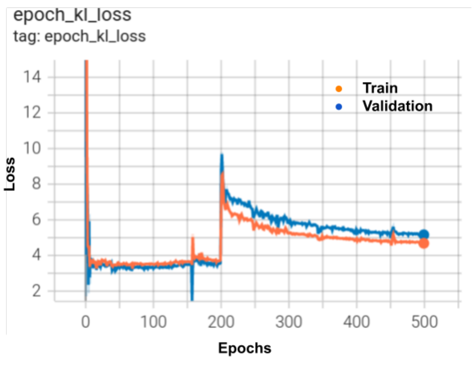
In this chapter the technical choices for the implementations of the methods (VAE, Reinforcement learning and Bayesian optimization) are presented along with the evaluations of their performance.

4.1 Variational Autoencoder

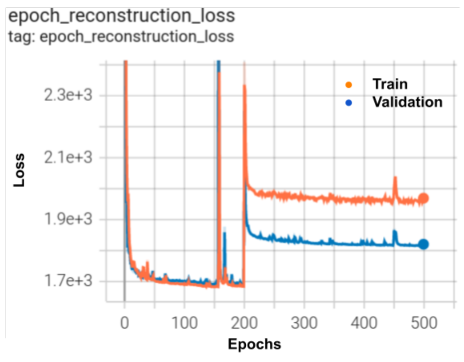
Plots of the reconstruction-, scaling-, KL- and total loss propagation throughout training is displayed in figure 4.1. The model has clearly reached a stationary point without getting overtrained. The jump in the loss at 200 epochs is from the switching of training data set. After training, the model was evaluated on the test set (see section 3.1.1 for details). Error data from the reconstructions of the test data is presented in table 4.1. Some of the reconstructed images can be seen in figure 4.2 while the histograms corresponding to sample number 1 are plotted in figure 4.3.

Table 4.1: Absolute reconstruction errors from 2D VAE on the test data. FWHM error presented as fraction of true value, center and width values are in millimeters.

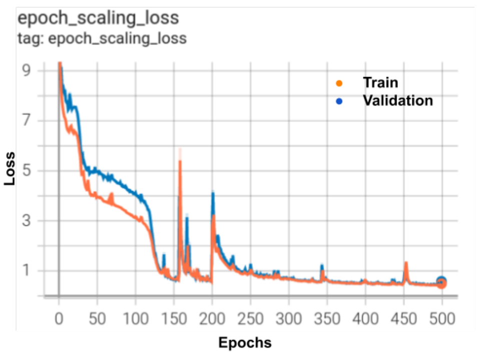
Lateral (x)			Vertical (y)		
FWHM	Center	Width	FWHM	Center	Width
0.4624	0.1056	0.0165	1.0854	0.0698	0.0305



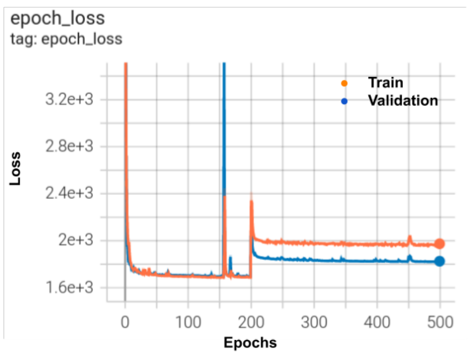
(a) KL loss



(b) Reconstruction loss



(c) Scaling loss



(d) Total loss

Figure 4.1: Loss throughout training of the 2D VAE

Table 4.2: Average FWHM errors from 2D VAE on test data. Values taken as the difference between true and predicted values (no absolute value), divided by the true value

Lateral FWHM	Vertical FWHM
−0.3527	−1.015

From looking at the reconstructions two things were concluded, firstly; the reconstruction was working well and managed to capture most of the shapes of the input images, secondly; there was a constant error in the FWHM since the reconstructed images seems brighter with sharper edges. The constant error in the FWHM was further confirmed when the absolute value of the differences between true and reconstructed FWHMs was removed, the results are presented in table 4.2. The error could possibly originate from the high penalization in the weighted cross-entropy (see C), but could be fixed in a post-processing step. Again, letting f_t represent the true FWHM, f_p the predicted and ϵ the average constant error, the following mathematical trick was applied:

$$\frac{f_t - f_p}{f_t} = -\epsilon \quad \Leftrightarrow \quad f_t - f_p = -\epsilon f_t \quad \Leftrightarrow \quad f_t = \frac{1}{1+\epsilon} f_p$$

$$\Rightarrow \frac{f_t - (f_p + x)}{f_t} = 0 \quad \Leftrightarrow \quad \frac{1}{1+\epsilon} f_p - f_p - x = 0 \quad \Leftrightarrow \quad x = -\frac{\epsilon}{1+\epsilon} f_p$$

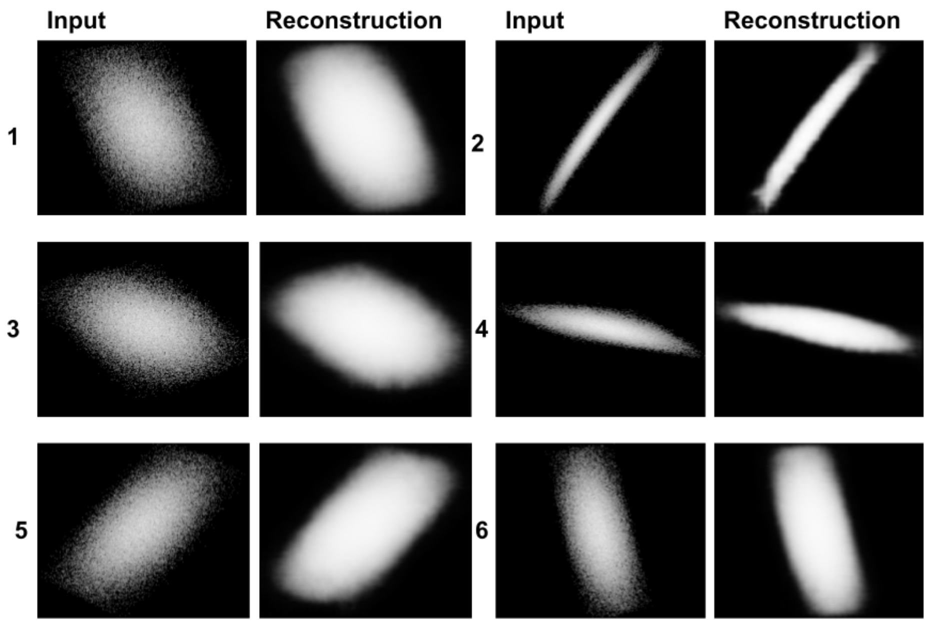


Figure 4.2: Some images and their reconstructions from the 2D VAE

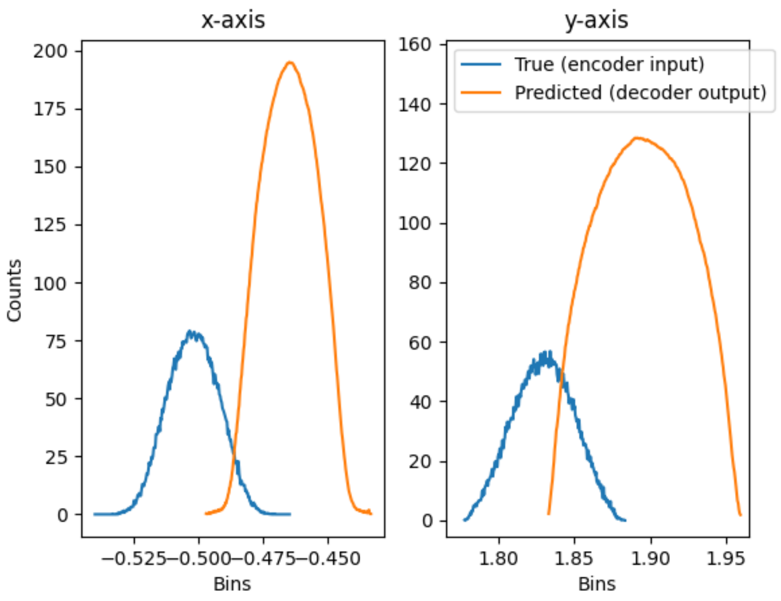


Figure 4.3: Intensity histograms for both axes of sample number 1 in figure 4.2

Thus, using $f_p^* = f_p - \epsilon/(1+\epsilon) f_p$ as the predicted FWHM would get rid of the stationary error. Applying this got the average absolute FWHM error fractions down to 0.29 and 0.45 for the lateral and vertical respectively, to be compared with the values in table 4.1. Histograms of the modified lateral and vertical FWHM errors are displayed in figure 4.4. It is evident that a few big outliers are affecting the mean, and that a majority of the samples appear below the mean, they are not normal distributed. Upon further investigation of the outliers, the cause of the errors was determined to be a faulty reconstruction of the width. The problem is that for the smallest dots an error of 0.01 mm will be big and thus throw off the FWHM. For reliable reconstructions the average width errors should be smaller than the smallest dots which are 0.005 mm in the lateral- and 0.001 mm in the vertical direction. We need to have high demands on the reconstructions to be sure that the generated data is close to reality and to get accurate answers to the question *where are we?* (see 1.2).

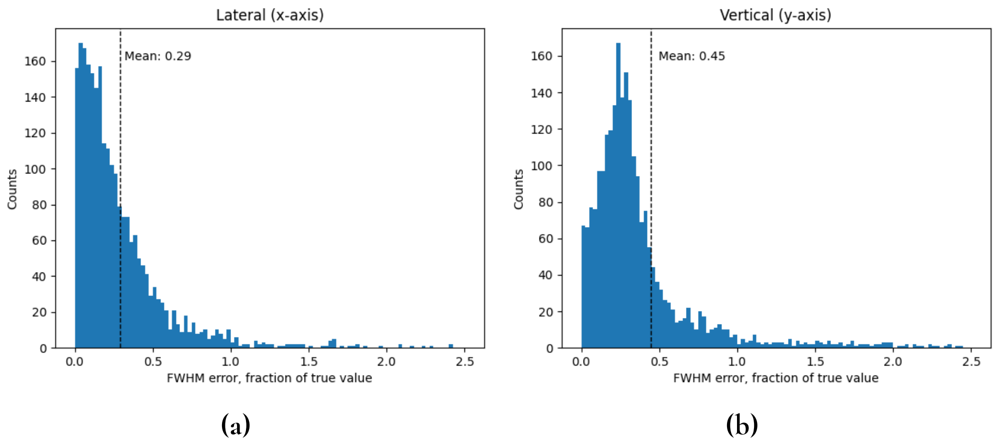


Figure 4.4: Histograms for FWHM reconstruction errors from the VAE. The errors are calculated as the absolute value of the difference between the true and reconstructed value, divided by the true. The values have been modified to remove the constant error.

4.2 Reinforcement Learning

For the DDPG, the discount factor was set to 0.99, and the target network update rate τ was set to 0.005. The learning rates were set to 0.001 and 0.002 for actor and critic respectively, which was all the default values given in the implementation [Singh, nd]. For policy gradient methods like DDPG, it is custom that the critic learning rate is set higher than that of the actor. Having an actor that changes faster than the critic means that the estimated future reward will not fully represent the action taken, this could eventually lead to overfitting [Lillicrap et al., 2016]. Initially the Ornstein-Uhlenbeck process was used for the exploration noise.

The average reward through the first 20 episodes of training is plotted in figure 4.5, if the model is trained properly the average reward should increase but it is evident that this was not the case. Plots of the actions taken in the last episode are displayed in figure 4.6, they saturate a lot with some big oscillations, the model had been overtrained. To solve the overtraining issue, L2 regularization was introduced to the actor network. The L2 regularizer forces the weights to decay towards zero by penalising the loss function with the L2 norm of all the weights [Ohlsson and Edén, 2020]. The regularizer did not solve the problems so then gradient clipping was introduced in the optimizer. This means that the gradients in the Adam optimizer (see section 2.2.2) was forced to stay in a defined range. The noise process was also changed to the uncorrelated normal distributed. The overfitting was, however, still present and the DDPG algorithm was abandoned due to lack of time.

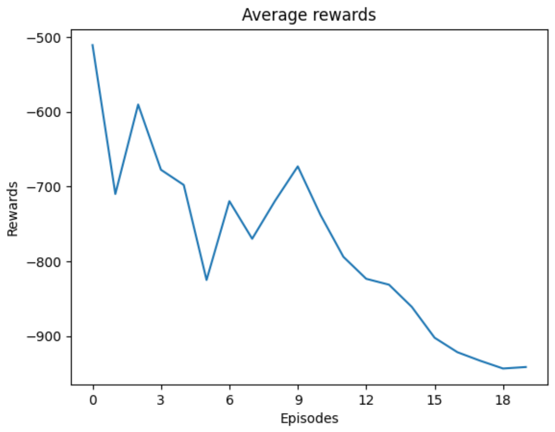
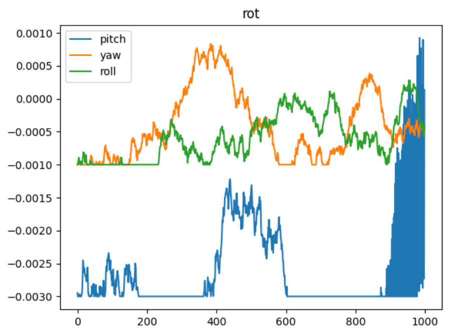
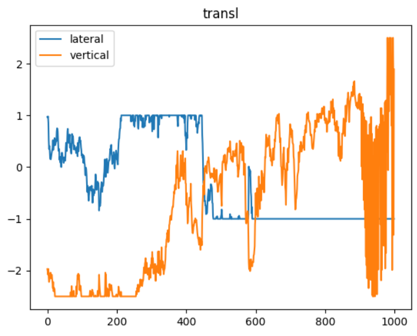


Figure 4.5: The average reward throughout training the DDPG model

No design choices were made for the hyperparameters of the PPO model, it was left to the Stable-Baselines implementation. A plot of the average reward, taken every 600th step in training, can be seen in figure 4.7, along with the last 600 actions taken in figure 4.8. Over the course of the 12000 iterations the algorithm did not achieve a beam which could be considered well aligned. However, the average rewards did increase which indicates that the model is learning. Unfortunately due to lack of time no further training was possible.



(a) Rotational parameters



(b) Transitional parameters

Figure 4.6: Actions taken for the alignment by the DDPG model in the 20th episode of training

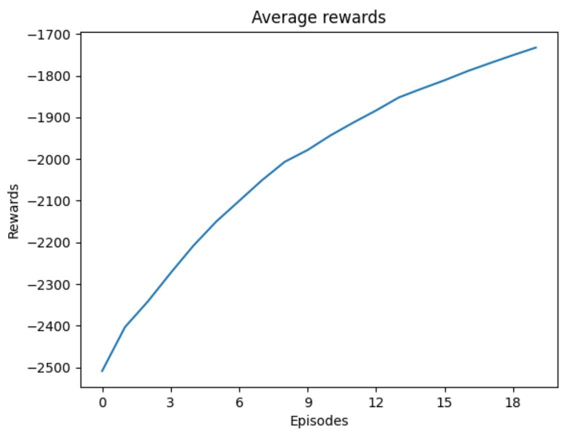
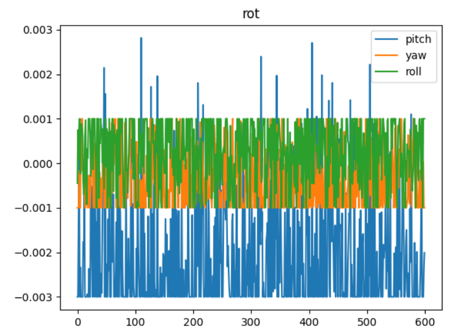
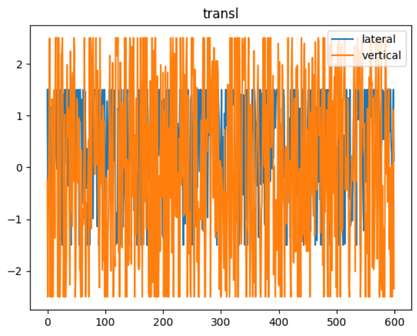


Figure 4.7: The average reward throughout training the PPO model



(a) Rotational parameters



(b) Transitional parameters

Figure 4.8: Actions taken for the alignment by the PPO model in the 20th episode of training

4.3 Bayesian Optimization

Two different setups of HyperMapper were tested. Setup 1 was run for 150 iterations, with all $\alpha_i = 1$ in the objective function (see equation 3.3). In setup 2 the iterations were increased to 250 and with weighting $\alpha_1 = 8.3$, $\alpha_2 = 20$, $\alpha_3 = 0.2$. The average resulting errors are presented in table 4.3. The different setups had little to no effect on the resulting parameter errors. The objective function error was, however, lower in the second setup which had higher weighted FWHM and lower target distance. This tells us that the target distance is the parameter that is the most off when getting close to the best focus position.

Table 4.3: Average M4 parameter errors from HyperMapper with two different setups, five runs in each setting. The objective function error is given as a fraction of the best possible value. The errors from setup 2 are also presented as percentages of their maximum value.

Setting	Pitch	Yaw	Roll	Lat. t.	Vert. t.	Objective fcn
Unit	mRad	mRad	mRad	mm	mm	fraction
Setup 1	0.241	0.093	1.064	0.296	0.058	8.6
Setup 2	0.204	0.034	1.150	0.266	0.068	0.37
S.2 as %	6.8	3.4	115	17.7	2.7	37

Table 4.4: Average values for the alignment parameters from the best points found by the Bayesian optimization runs. For comparison, the average values from the last 5 runs of the DDPG model are included. Green values are below the well-aligned limit in table 3.2, red are above

Model	FWHMx	FWHMy	Gap	Target dist.	Well aligned
Setup 1	0.0150	0.0049	12.09	0.0929	No
Setup 2	0.0096	0.0036	15.80	0.1078	No
DDPG	0.0662	0.1294	389.5	3.947	No

The optimizer was good at aligning the pitch, yaw and vertical translation parameters. The errors were all lower than 10% of the maximum limit (see table 3.1). The roll and lateral translation errors were at 20% and 100% of the maximum limit respectively, which is not where we want them. This difference in performance can be explained by recalling the objective function, plotted in figure 3.11. The yaw and vertical translation both have clear minima at the target spot with steep slopes, which makes it easy for the algorithm to find the correct value. On the other hand, looking at the roll, it does not have a distinct minima in the objective function which in return results in big errors (bigger than the actual alignment limit at 1.0 mRad). Through further analyse of the data and objective function in the scatter plots 4.9 we can see where the issue lies, there is a very big spread towards the bottom where some settings with offsets gets very small values from the objective function. This imposes difficulties on the optimizer and requires more iterations.

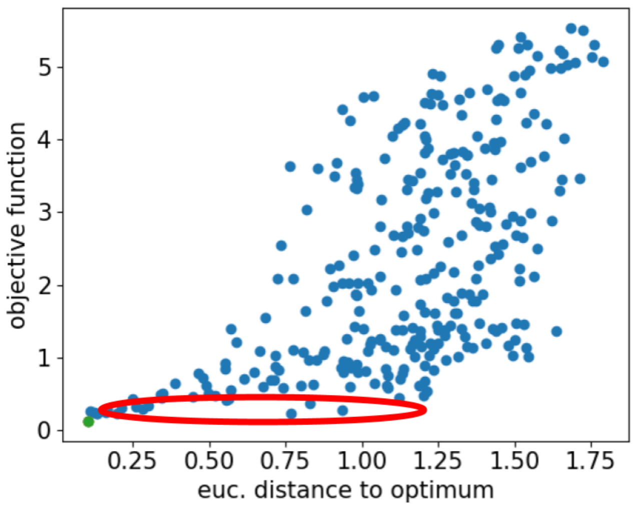


Figure 4.9: Scatter plot of data points from the test data set, with the euclidean distance of the offsets from the origin on the x-axis and the output from the objective function on the y-axis. Marked in the figure is the area where the BOA runs roughly end up

Chapter 5

Discussion

With the overall goal of an automated alignment process at MAX IV, three different ML approaches were designed, implemented, tested and evaluated. The methods were variational autoencoder, reinforcement learning (more specifically policy gradient methods) and Bayesian optimization. Only simulated data was used, and data sets had to be structured. In this chapter follows the discussion about the performance of the models with respect to the aim of the project.

5.1 Variational Autoencoder

The VAE managed to reconstruct the images visually quite well. It managed to capture most shapes and it was clear to see which image was the input. There is, however, room for improvement in the average FWHM errors which are 29% and 45% off the true value. From plotting the histograms in 4.4 we can see that the averages are affected by big outliers which mostly comes from reconstructing the very small focal spots with errors in the scaling parameters. Even if the scaling parameters seem to have small errors they are too big compared to the small focal spots. Achieving better reconstructions of the scaling would improve the resulting FWHMs. The results are not perfect but promising for future work. The architecture could be expanded, especially the network for the scaling inputs which could potentially decrease the errors by a lot. The model could also be expanded to take 3D images and take in the whole reconstruction of the beam, which would require a lot more data. The next immediate step is however to link the latent space to the real parameters through another network. The latent space has no relation to the actual parameters and physics, but consist only of parameters found in the unsupervised training of the VAE. If a connection to the real parameters could be made, we would have a robust model for predicting the set parameters from the output images - an answer to the question *where are we?*.

5.2 Reinforcement Learning

Using RL algorithms in the alignment process is an interesting approach which looked promising in the beginning of this project. Especially in application to the actual beamline where outside factors are affecting the alignment which is something a RL model could handle. One potential advantage of RL is that the agent can learn how to cope with a changing environment. However, the first step was to get it working on the simulated data, which is not a changing environment. Not once, through thousands of ray tracings, did any of the algorithms achieve a well aligned beam. Due to the long time taken for each action[1], they were hard to train and evaluate. The overfitting problem of the DDPG was not resolved and the PPO was not fully evaluated due to lack of time. The PPO model did show positive signs that the average rewards was increasing, it also did a good job of exploring the whole action space compared to the DDPG. It would be very interesting to see how it would get on with more training. The idea of using RL for automation of the alignment process should not be ruled out completely, it is just going to require more work.

5.3 Bayesian Optimizaion

The Bayesian optimization gave mixed, but promising results. The algorithm worked well for the pitch, yaw and vertical translation, but got big errors for the roll and lateral translation. The reason behind the big errors is mostly due to the parameters having little effect on the objective function. The worst parameter in this aspect is the roll which basically could be set to anything within the range and still not affect the output value by much. Still, disregarding the weak parameters, the others still have some room of improvement with errors of magnitude around 5% of the maximal offset. From table 3.3 we see that the average errors should be around 1% to give us an accurate answer to the question of *where are we going?* (see section 1.2).

The solution possibly lies in designing a new objective function of which the roll and lateral translation have a greater influence. From table 4.4 we see that the gap parameter, which was not present in the objective function, is the only one that is way off for both setups. In other words, the beam was to be considered well aligned in terms of the parameters used in the actual optimization. However, the gap parameter was omitted for a reason, it gives the objective function a very non-smooth behaviour and will either dominate the function, or if suppressed enough, not affect it at all. The results prove that the optimization algorithm works and if a better objective function were to be found it would improve them. A big advantage of the BOA is that it requires no initial training, thus the jump from simulation to the real world would go easier than with the deep learning methods which require tons of data. It is also easy to understand and could potentially be integrated into more sophisticated algorithms. A new algorithm could be specifically designed for aligning beamlines and give the model some sort of “memory" instead of having to re-learn the optimization every time. Further ideas for improvement could be to confine the parameters by tightening the limits throughout the optimization or to design separate objective functions for the parameters and optimize them one at a time.

[1]20 episodes would take more than 24 hours even when running on several cores

Chapter 6
Conclusion

Concluding comments from the discussion of the models performance will be presented in this chapter along with suggestions for future work.

Using ML for alignment is a very complex task and we are quite far from a complete automatic process. In this project, the problem was quite heavily simplified compared to reality, but still it was difficult to find a working solution. It is important that the models get the beam data they need in order to make the predictions as accurate as our requirements. Identifying what parts of data is useful and what is not is a crucial step. The complexity and difference between beamlines impose big difficulties since new models probably have to be made for each beamline. The lack of real data is another big obstacle which has to be solved if the algorithms are to be implemented on the real beamline. The problem might seem a bit overwhelming, but there has definitely been some positive signs.

6.1 Future work

All models presented in this project could be futher developed and improved. There is room for improvement with the current VAE model which could be fine tuned. The network could also be expanded to take 3D images which would result in a new model. More testing is needed for the RL models and clever ways of defining a better objective function for the BOA should be explored.

Furthermore the transition from simulation to reality needs to be further looked upon. The question of how an actual automatic alignment process would work needs to be assessed, with more questions following: How and what data would be needed? How many evaluations of the quality of alignment is reasonable? How fast would the alignment model need to work? What are the differences in data between different beamlines? These questions all need answers in order for the automatic alignment to work on the actual beamlines. For a

ML based automatic alignment process to be achieved, a database would need to be set up and alignment data from beamlines should be collected as soon as possible. In section 3.1.1, I propose a way of organizing the data, which worked very well for training and evaluating the VAE.

References

[Altarelli and Salam, 2004] Altarelli, M. and Salam, A. (2004). The quest for brilliance: Light sources from the third to the fourth generation. *Europhysics News*, 35(2):47–50.

[Bennström and Winzell, 2021] Bennström, A. and Winzell, F. (2021). Automated 3d bone segmentation using deep learning in scoliosis. *Master's Theses in Mathematical Sciences*.

[Bishop, 2006] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer New York, NY.

[Brock et al., 2016] Brock, A., Lim, T., Ritchie, J. M., and Weston, N. (2016). Generative and discriminative voxel modeling with convolutional neural networks. *CoRR*, abs/1608.04236.

[Chollet, nd] Chollet, F. (n.d.). Retrieved February 21, 2022 from `https://keras.io/examples/generative/vae/`.

[Cremer, 2012] Cremer, J. (2012). Introduction to neutron and x-ray optics. *Advances in Imaging and Electron Physics*, 172.

[Ekholm and Tokushima, 2022] Ekholm, V. and Tokushima, T. (2022). Data from pinhole scan at veritas beamline. MAX IV.

[Frazier, 2018] Frazier, P. (2018). Bayesian optimization. *Informs*, pages 255–78.

[Gad, nd] Gad, A. (n.d.). How to build a variational autoencoder in keras. Retrieved February 21, 2022 from `https://blog.paperspace.com/how-to-build-variational-autoencoder-keras/`.

[Kirkpatric and Baez, 1948] Kirkpatric, P. and Baez, A. (1948). Formation of optical images by x-rays. *Journal of the Optical Society of America*, 38(9):766–74. doi:10.1364/JOSA.38.000766.

[Klementiev and Chernikov, 2014] Klementiev, K. and Chernikov, R. (2014). Powerful scriptable ray tracing package xrt. *Advances in Computational Methods for X-Ray Optics III*, 9209:60–75. doi:10.1117/12.2061400, Online documentation at xrt.readthedocs.io.

[Lillicrap et al., 2016] Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *International Conference on Learning Representations*. doi:10.48550/ARXIV.1509.02971.

[Maffettone et al., 2021] Maffettone, P., Lynch, J., Caswell, T., Cook, C., Campbell, S., and Olds, D. (2021). Gaming the beamlines—employing reinforcement learning to maximize scientific outcomes at large-scale user facilities. *Machine Learning: Science and Technology*, 2(2).

[MAX IV, nd] MAX IV (n.d.). Retrieved May 25, 2022 from `https://www.maxiv.lu.se/accelerators-beamlines/beamlines/`.

[Nardi et al., 2019] Nardi, L., Koeplinger, D., and Olukotun, K. (2019). Practical design space exploration. *IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 347–58.

[Nash et al., 2020] Nash, B., Goldring, N., Edelen, J., Federer, C., Moeller, P., and Webb, S. (2020). Reduced model representation of x-ray transport suitable for beamline control. *Advances in Computational Methods for X-Ray Optics V*. doi:10.1117/12.2568187.

[Odaibo, 2019] Odaibo, S. (2019). Tutorial: Deriving the standard variational autoencoder (vae) loss function. Retrieved May 24, 2022 from `https://arxiv.org/pdf/1907.08956.pdf`.

[Ohlsson and Edén, 2020] Ohlsson, M. and Edén, P. (2020). *Lecture Notes on Introduction to Artificial Neural Networks and Deep Learning*. Lund University.

[Peatman, 1997] Peatman, W. (1997). *Gratings, Mirrors and Slits: Beamline Design for Soft X-Ray Synchrotron Radiation Sources*. Taylor Francis Ltd.

[Raffin et al., 2021] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*.

[Samuli et al., 2017] Samuli, U., Såthe, C., Grizolli, W., Agåker, M., Head, A., Andersson, M., and Huang, S. e. a. (2017). The species beamline at the max iv laboratory: a facility for soft x-ray rixs and apxps. *Journal of synchrotron radiation*, 24(1):344–53.

[Shulman et al., 2015] Shulman, J., Levine, S., Moritz, P., Jordan, M., and Abbeel, P. (2015). Trust region policy optimization. *International conference on machine learning*.

[Shulman et al., 2017] Shulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *OpenAI*.

[Singh, nd] Singh, H. (n.d.). Retrieved April 19, 2022 from `https://keras.io/examples/rl/ddpg_pendulum/`.

[Sutton and Barto, 2018] Sutton, R. and Barto, A. (2018). *Reinforcement learning: an introduction*. Westchester Publishing Services.

[Tavares et al., 2014] Tavares, P., Leemann, S., Sjöström, M., and Andersson, (2014). The max iv storage ring project. *Journal of synchrotron radiation*, 21(5):862–77.

[Xu et al., 2020] Xu, J., Li, Z., Du, B., Zhang, M., and Liu, J. (2020). Reluplex made more practical: Leaky relu. *2020 IEEE Symposium on Computers and Communications*, pages 1–7.

Appendices

Appendix A
Data XML file layout

```xml
<?xml version='1.0' encoding='utf-8'?>
<data numrays="100000" energy="400.0" source_bandwidth="0.001">
  <setup>
    <source />
  <M4 surface="Au" Pitch_deg="2.0" Yaw="0.0" Roll="0.0"
        Meridional_error="2.327105669325773e-06" Sagittal_error="1.1490084242296004e-05"
        Roughness="3e-10" Dist_to_target="700.0" />
  <Target dqs="9" xz_limits="+/- 2" bins="256" />
  </setup>
<sample_0>
    <specifications>
      <pitch unit="rad">0.0006226396087157561</pitch>
    <yaw unit="rad">0.0007001720908137209</yaw>
    <roll unit="rad">-0.0008560570505258339</roll>
    <x_transl unit="mm">-0.2615230398626873</x_transl>
    <z_transl unit="mm">0.7078175100762971</z_transl>
    </specifications>
  <images>
      <image file="06070915_00000_00.png" centerx="1.1077" dx="0.0656" centerz="0.7035" dz="0.0232" />
    <image file="06070915_00000_01.png" centerx="0.5505" dx="0.0652" centerz="0.7070" dz="0.0237" />
    <image file="06070915_00000_02.png" centerx="-0.0067" dx="0.0632" centerz="0.7106" dz="0.0298" />
    <image file="06070915_00000_03.png" centerx="-0.5639" dx="0.0643" centerz="0.7142" dz="0.0424" />
    <image file="06070915_00000_04.png" centerx="-1.1211" dx="0.0611" centerz="0.7178" dz="0.0531" />
    <image file="06070915_00000_05.png" centerx="-1.6783" dx="0.0614" centerz="0.7214" dz="0.0656" />
    <image file="06070915_00000_06.png" centerx="-2.2355" dx="0.0607" centerz="0.7250" dz="0.0771" />
    <image file="06070915_00000_07.png" centerx="-2.7927" dx="0.0626" centerz="0.7285" dz="0.0897" />
    <image file="06070915_00000_08.png" centerx="-3.3499" dx="0.0640" centerz="0.7321" dz="0.1070" />
    </images>
  </sample_0>
<sample_1>
...
</data>
```

Appendix B
Misaligned samples

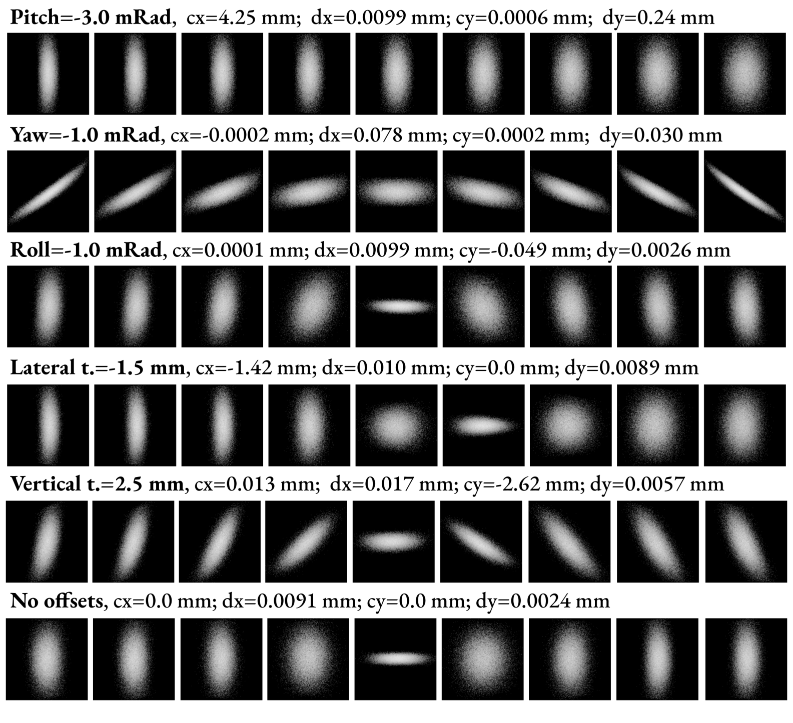


Figure B.1: Misaligned samples from XRT with maximum offset in one parameter. Images from nine screens at equal distances from 14 mm in front of the target to 14 mm behind. Presented along with scaling data for the center image (at target). cx = center of image on x-axis, dx = spread of image on x-axis, cy = center on y-axis, dy = spread on y-axis.

Appendix C
VAE architecture summary

ENCODER

```
Model: "encoder_model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 encoder_input (InputLayer)     [(None, 256, 256, 1  0           []
                                )]

 encoder_conv_1 (Conv2D)        (None, 254, 254, 8)  80          ['encoder_input[0][0]']

 encoder_norm_1 (BatchNormaliza  (None, 254, 254, 8)  32         ['encoder_conv_1[0][0]']
 tion)

 encoder_leayrelu_1 (LeakyReLU)  (None, 254, 254, 8)  0          ['encoder_norm_1[0][0]']

 encoder_conv_2 (Conv2D)        (None, 252, 252, 8)  584         ['encoder_leayrelu_1[0][0]']

 encoder_norm_2 (BatchNormaliza  (None, 252, 252, 8)  32         ['encoder_conv_2[0][0]']
 tion)

 encoder_leayrelu_2 (LeakyReLU)  (None, 252, 252, 8)  0          ['encoder_norm_2[0][0]']

 encoder_conv_3 (Conv2D)        (None, 250, 250, 8)  584         ['encoder_leayrelu_2[0][0]']

 max_pooling2d (MaxPooling2D)   (None, 125, 125, 8)  0           ['encoder_conv_3[0][0]']

 encoder_norm_3 (BatchNormaliza  (None, 125, 125, 8)  32         ['max_pooling2d[0][0]']
 tion)

 encoder_leayrelu_3 (LeakyReLU)  (None, 125, 125, 8)  0          ['encoder_norm_3[0][0]']

 encoder_conv_4 (Conv2D)        (None, 123, 123, 16  1168        ['encoder_leayrelu_3[0][0]']
                                )

 encoder_norm_4 (BatchNormaliza  (None, 123, 123, 16  64         ['encoder_conv_4[0][0]']
 tion)                          )

 encoder_leayrelu_4 (LeakyReLU)  (None, 123, 123, 16  0          ['encoder_norm_4[0][0]']
                                )

 encoder_conv_5 (Conv2D)        (None, 61, 61, 16)   2320        ['encoder_leayrelu_4[0][0]']

 max_pooling2d_1 (MaxPooling2D)  (None, 30, 30, 16)  0           ['encoder_conv_5[0][0]']

 encoder_norm_5 (BatchNormaliza  (None, 30, 30, 16)  64          ['max_pooling2d_1[0][0]']
 tion)

 encoder_leayrelu_5 (LeakyReLU)  (None, 30, 30, 16)  0           ['encoder_norm_5[0][0]']

 encoder_conv_6 (Conv2D)        (None, 14, 14, 32)   4640        ['encoder_leayrelu_5[0][0]']

 max_pooling2d_2 (MaxPooling2D)  (None, 7, 7, 32)    0           ['encoder_conv_6[0][0]']

 encoder_norm_6 (BatchNormaliza  (None, 7, 7, 32)    128         ['max_pooling2d_2[0][0]']
 tion)

 encoder_leayrelu_6 (LeakyReLU)  (None, 7, 7, 32)    0           ['encoder_norm_6[0][0]']
 flatten (Flatten)              (None, 1568)          0           ['encoder_leayrelu_6[0][0]']

 extra_input (InputLayer)       [(None, 4)]           0           []

 dense_1 (Dense)                (None, 1568)          2460192     ['flatten[0][0]']

 dense (Dense)                  (None, 4)             20          ['extra_input[0][0]']

 concatenate (Concatenate)      (None, 1572)          0           ['dense_1[0][0]',
                                                                   'dense[0][0]']

 encoder_mu (Dense)             (None, 10)            15730       ['concatenate[0][0]']

 encoder_log_variance (Dense)   (None, 10)            15730       ['concatenate[0][0]']

 sampling (Sampling)            (None, 10)            0           ['encoder_mu[0][0]',
                                                                   'encoder_log_variance[0][0]']

==================================================================================================
Total params: 2,501,400
Trainable params: 2,501,224
Non-trainable params: 176
__________________________________________________________________________________________________
DECODER
Model: "decoder_model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape          Param #     Connected to
==================================================================================================
 decoder_input (InputLayer)     [(None, 10)]          0           []

 decoder_dense_1 (Dense)        (None, 1568)          17248       ['decoder_input[0][0]']

 reshape (Reshape)              (None, 7, 7, 32)      0           ['decoder_dense_1[0][0]']

 decoder_conv_tran_1 (Conv2DTra  (None, 15, 15, 32)   9248        ['reshape[0][0]']
 nspose)

 decoder_norm_1 (BatchNormaliza  (None, 15, 15, 32)   128         ['decoder_conv_tran_1[0][0]']
 tion)

 decoder_leayrelu_1 (LeakyReLU)  (None, 15, 15, 32)   0           ['decoder_norm_1[0][0]']

 decoder_conv_tran_2 (Conv2DTra  (None, 31, 31, 16)   4624        ['decoder_leayrelu_1[0][0]']
 nspose)

 decoder_norm_2 (BatchNormaliza  (None, 31, 31, 16)   64          ['decoder_conv_tran_2[0][0]']
 tion)

 decoder_leayrelu_2 (LeakyReLU)  (None, 31, 31, 16)   0           ['decoder_norm_2[0][0]']

 decoder_conv_tran_3 (Conv2DTra  (None, 63, 63, 16)   2320        ['decoder_leayrelu_2[0][0]']
 nspose)

 decoder_norm_3 (BatchNormaliza  (None, 63, 63, 16)   64          ['decoder_conv_tran_3[0][0]']
 tion)

 decoder_leayrelu_3 (LeakyReLU)  (None, 63, 63, 16)   0           ['decoder_norm_3[0][0]']

 decoder_conv_tran_4 (Conv2DTra  (None, 126, 126, 8)   1160       ['decoder_leayrelu_3[0][0]']
 nspose)

 decoder_norm_4 (BatchNormaliza  (None, 126, 126, 8)   32         ['decoder_conv_tran_4[0][0]']
 tion)

 decoder_leayrelu_4 (LeakyReLU)  (None, 126, 126, 8)   0          ['decoder_norm_4[0][0]']

 decoder_conv_tran_5 (Conv2DTra  (None, 253, 253, 8)   584        ['decoder_leayrelu_4[0][0]']
 nspose)

 decoder_norm_5 (BatchNormaliza  (None, 253, 253, 8)   32         ['decoder_conv_tran_5[0][0]']
 tion)

 decoder_leayrelu_5 (LeakyReLU)  (None, 253, 253, 8)   0          ['decoder_norm_5[0][0]']

 decoder_conv_tran_6 (Conv2DTra  (None, 256, 256, 1)   129        ['decoder_leayrelu_5[0][0]']
 nspose)

 decoder_dense_ex_1 (Dense)     (None, 4)             44          ['decoder_input[0][0]']

 decoder_output (LeakyReLU)     (None, 256, 256, 1)   0           ['decoder_conv_tran_6[0][0]']

 decoder_dense_ex_2 (Dense)     (None, 4)             20          ['decoder_dense_ex_1[0][0]']

==================================================================================================
Total params: 35,697
Trainable params: 35,537
Non-trainable params: 160
__________________________________________________________________________________________________
```

Appendix D
Elliptic geometry

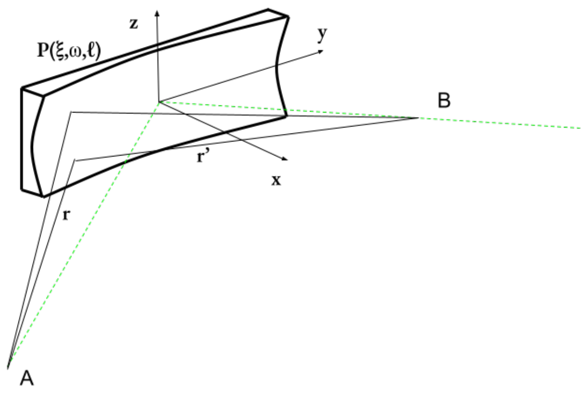


Figure D.1: The coordinate system for the optical path function

We define three points, point A,$P_{\xi,\omega,l}$ and B, where rays travel from A, hits the mirror on P and reflects to B (see figure D.1). Fermats principle states that if the point A were to be imaged at B, then all paths from A via the mirror surface $P(\xi, \omega, l)$ to B must be equally long. This is the challenge for the mirror.

The surface of the mirror is described generally by the equation

$$\xi = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} a_{ij} \omega^i l^j$$

and $a_{00} = a_{10} = 0; \quad j = \text{even}$
An arbitrary path is given by $F = \bar{AP} + \bar{PB}$ and all rays that lead to a focus must fulfill the following:

$$\frac{\delta F}{\delta \omega} = 0; \quad \frac{\delta F}{\delta l} = 0$$

$$\text{An ellipse is given by:} \quad \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$
$$\text{where} \quad a = \frac{1}{2}(r + r')$$
$$b = \sqrt{a^2 + d^2}$$
$$d = \frac{1}{2}\sqrt{r^2 + r'^2 - 2rr' \cos 2\theta}.$$

The coefficients a_{ij} will be obtained numerically, and all definitions recast in terms of $P(\xi, \omega, l)$ in the Cartesian system shown in figure D.1. The optical paths can be defined as:

$$\bar{AP} = \sqrt{(\xi - r\cos\alpha)^2 + (\omega - r\sin\alpha)^2 + (l - z)^2}$$
$$\bar{BP} = \sqrt{(\xi - r'\cos\alpha)^2 + (\omega - r'\sin\alpha)^2 + (l - z')^2}.$$

For the central ray $\left(\frac{\delta F}{\delta l}\right)_{\xi=\omega=l=0} = 0$ leading to $\frac{z}{r} = \frac{z'}{r'}$. F can be defined by a polynomial series employing the coefficients of the surface [Peatman, 1997].

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2022-06-07

EXAMENSARBETE X-ray Beamline Alignment Using Machine Learning at MAX IV
STUDENT Emil Winzell
HANDLEDARE
EXAMINATOR Marcus Klang (LTH)

Applying machine learning for alignment of X-ray beams at MAX IV

POPULAR SCIENTIFIC SUMMARY **Emil Winzell**

At MAX IV, X-ray emissions are used for analysis of materials, to unravel atomic structures and much more. Each new experiment requires exact properties of the light which has be perfectly aligned by several apertures and optics. This thesis explores the possibility of using machine learning in an automatic alignment process, trough an empirical investigation.

X-ray radiation behave very differently from visible light. The rays will rather attenuate than reflect from most materials and X-ray mirrors can only move the beam by letting it slightly graze of the surface by small angles. For good alignments, the mirrors are configured on scales down to micro radians and nano meters. The process is affected by many outside factors such as temperature and vibrations, as a result it takes a lot of time and experience for the scientists to perfectly align the beam. The aim of this thesis was to do an empirical investigation of ML models, designed to predict and optimize the optical parameters, given properties of the beam.
Simulated data from the MAX IV developed program *XRT* was used. The beamline was simplified to a fine-tuning scenario where only the position of last mirror was tweaked, which had a total of five degrees of freedom: pitch, yaw, roll, lateral and vertical translations. From the simulations, the beams propagation through space could be reconstructed through plots of the light intensity (see figure). A variational autoencoder was implemented in order to predict the optical parameters from the output 2D images and to investigate the possibility of generating new data synthetically instead of using the time consuming simulation.
The mirror was given a random offset and two different types of models were implemented to try to counteract the change and align the beam

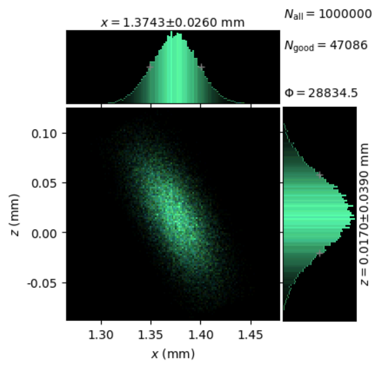


Figure 1: Output data from a slice of the beam in *XRT*

again. The first model was reinforcement learning where an agent learns a policy from taking actions (tweaking the mirror) and receiving rewards for how good its actions are. The second was a pure Bayesian optimization, which used a statistical model to find the optimum settings and thus not using any actual learning. The reinforcement learning models never really worked and were outperformed by the Bayesian optimization which managed to get quite close to the optimal solution. However, the problem was heavily simplified and even if the results were promising there is a long way to go for achieving an automatic alignment process.