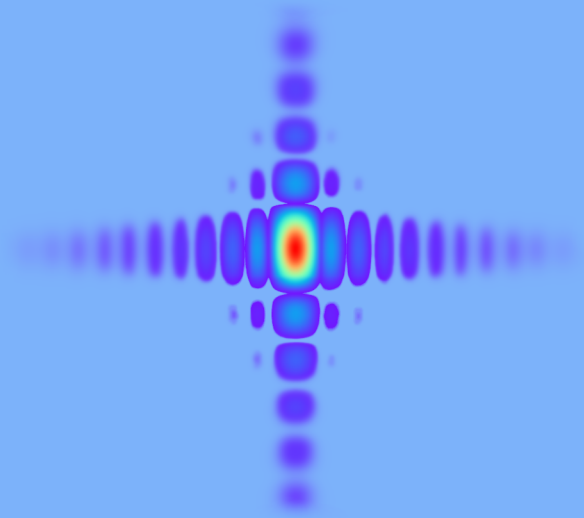
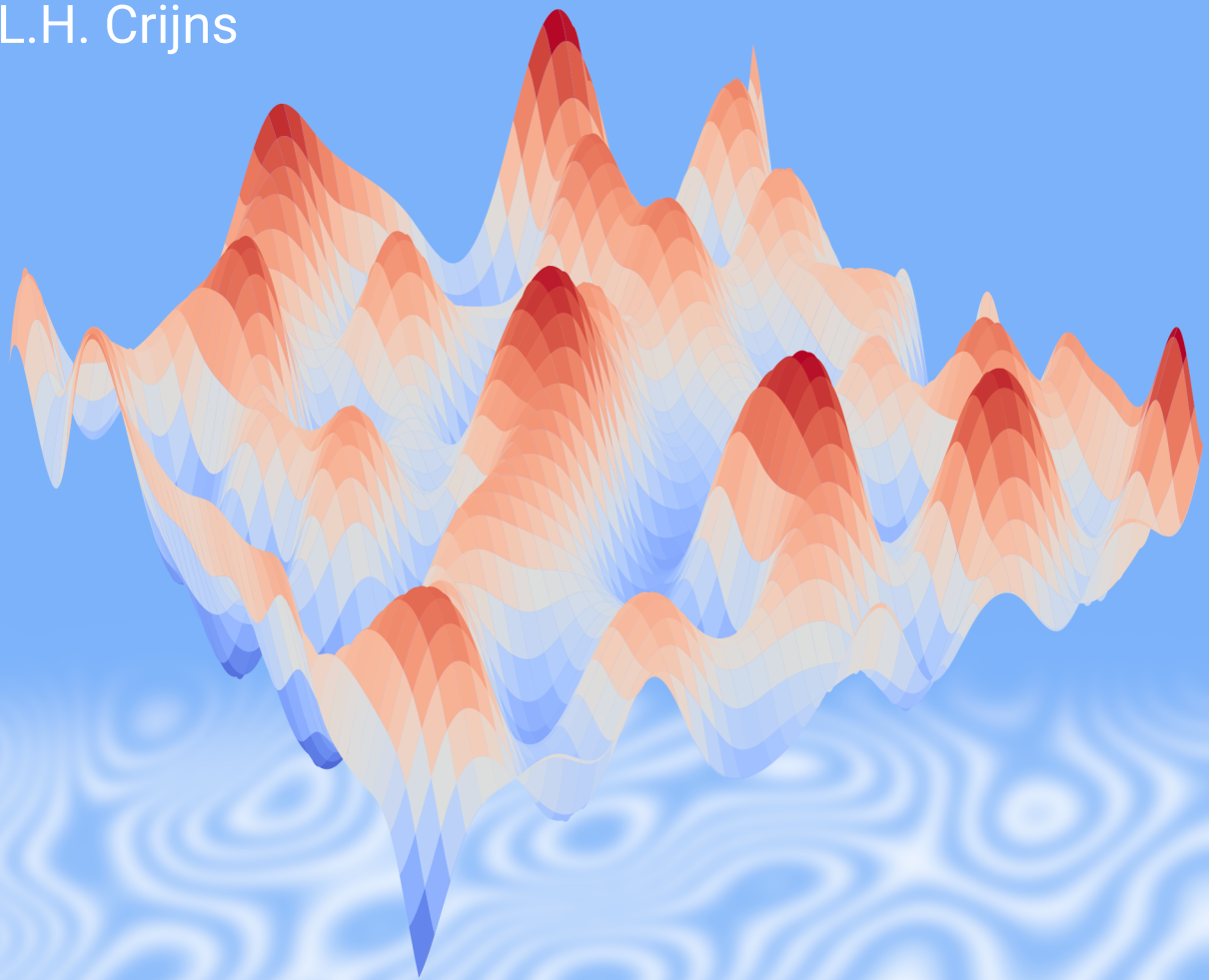
PINN inspired Freeform Design



Using Fraunhofer Diffraction to find Freeforms described by B-spline Surfaces

L.H. Crijns



Delft University of Technology

TU Delft

PINN inspired Freeform Design

Using Fraunhofer Diffraction to find Freeforms described by B-spline Surfaces

by

L.H. Crijns

to obtain the degree of Bachelors of Science

at the Delft University of Technology,

Student number: 4912543

Project duration: March 1, 2021 – July 5, 2021

Thesis committee:

Dr. A. Adam,	TU Delft, Optics Group
Dr. M. Möller,	TU Delft, Numerical Analysis Group
Ir. A. Heemels	TU Delft, Optics Group
Dr. J. Dubbeldam,	TU Delft, Mathematical Physics Group
Dr. Q. Tao	TU Delft, Imaging Physics

TU Delft

Abstract

This project aims to recreate intensity patterns using Fraunhofer diffraction as a means of simulation. These intensity patterns are created by phase shifting specific parts of an incoming field of light. These phase shifts are determined by a B-spline surface, which is in turn controlled by so-called control points. Only a handful of control points can describe a whole surface. The position of these control points is then determined using machine learning and specifically a technique inspired by 'physics-informed neural networks', which were introduced last year by Raissi et al. [1]. With this method, simple experiments which sought to recreate intensity patterns known to be in the solution space were carried out. These experiments showed some success, but suffered from the fact that they used too sensitive parameters in the input of the machine learning model, reducing the sophisticated method to a Monte Carlo search, or they used no input at all, which degraded the machine learning model to simple parameter optimization. Nevertheless, these experiments showed that this method has the potential to be used in more flexible optical setups, where multiple configurations can yield the same intensity pattern or where changing the parameters defining the setup do not induce enormous changes in the resulting intensity pattern. In addition, the proposed method relies upon Fraunhofer diffraction, which, when discretized for numerical computation, introduces aliasing issues when the incoming field changes too rapidly. This phenomenon was especially apparent when using point sources that create spherical wave fronts. A possible solution for this issue is to consider ray tracing techniques in future research.

Contents

1 Introduction

Light is all around us coming from the sun, artificial light and smartphone screens. To see images of objects using light, it must be focused on some kind of sensor. In the case of human vision, the light enters the eye through the cornea and is then focused using a lens, which ensures a proper image forms on the retina. Many have seen such systems in high school physics when they constructed the image of an object using the focal points of a convex lens. If the lens has a more complicated structure, focal points can no longer be used to construct an image. Instead, techniques such as ray tracing and diffraction integrals can be used to find the electromagnetic field directly behind the lens. The problem of finding the intensity distribution after a lens can be readily solved using these techniques. However, one may wonder what kind of lens can be used to obtain a desired intensity profile. This 'inverse' problem is much harder to solve and has no known exact solution. In general, the shape of lenses necessary to construct a certain intensity distribution or image are not the classically shaped convex or concave lenses, but a more 'freely' shaped lens, a so-called freeform lens.

Freeforms are already in use in the automotive and aerospace industry, where they are used in heads-up displays. In the future, freeforms may be used in circuitry printing, more accurate sensors or augmented reality headsets [1]. What makes freeforms an attractive alternative to classical lenses, is the potential for them to be smaller and do more, meaning that a single freeform may be able to replace multiple classical lenses in cascade.

Recently, the idea of using machine learning to help solve the inverse problem emerged. A bachelor thesis by Imhof [2] showed by proof-of-concept that it is possible to find 1D B-spline based freeforms using an unsupervised machine learning approach inspired by the work of Raissi et al. [1]. An important limitation he found is that it only works well for intensity profiles known to be in the solution space, which is quite small.

In this thesis the aim will be to improve upon the work done by Imhof, by extending the problem to 2D freeforms. Specifically, a B-spline surface described by control points will represent the phase distribution induced by a 2D freeform. The control points of this surface will be determined with the aid of machine learning. Just as Imhof did, the machine learning technique used here will be a neural network that is trained in an unsupervised fashion, similar to the work done by Raissi et al. [1] in their Physics informed neural network. In contrast to Imhof's work, in this thesis a network will be trained for a specific intensity profile and optical setup. The inputs and outputs of the network will determine the parameters of the optical setup. With this change, the hope is that more flexibility and a larger solution space can be achieved. Important to note is that the Fraunhofer diffraction integral will be used for simulating the propagation of light, as this is essentially a Fourier transform, allowing GPU computation to speed up the simulation immensely.

To start off, the Theory chapter will introduce the necessary background knowledge used in this thesis. After this the Experiment chapter will introduce the setup used here and discuss what experiments will be carried out. The results of these experiments and some light discussion of these results will be presented in the Results chapter. Next, the Discussion chapter follows, in which main points of discussion will be elaborated

[1] TNO and Asphericon have articles outlinning possible use cases and future developments: `https://www.tno.nl/en/tno-insights/articles/optics-2-0-the-future-of-freeform-optics/` and `https://www.asphericon.com/en/blog/detail/freeform-optics-fields-of-application-and-future-opportunities-for-use`, respectively.

upon and recommendations regarding future research will be presented. And finally, the Conclusion chapter will present the main findings of this thesis and shortly summarise important recommendations.

Nomenclature

Constants

μ_0	Magnetic permeability in vacuum: $4\pi \times 10^{-7}\,\mathrm{N/A^2}$
μ	Magnetic permeability in a certain medium
ϵ_0	Permittivity in vacuum: $8.854\,187\,812\,8 \times 10^{-12}\,\mathrm{Fm^{-1}}$
ϵ	Permittivity in a certain medium

Operators

$\nabla \times (\cdot)$	The curl operator working on $\mathbb{R}^3$
$\nabla \cdot (\cdot)$	The divergence operator working on $\mathbb{R}^3$
$\nabla(\cdot)$	The gradient operator working on $\mathbb{R}^n$, arbitrary number of dimensions
$\nabla^2(\cdot)$	The laplacian operator working on $\mathbb{R}^3$
$\mathrm{Re}\{\cdot\}$	Real part operator: $\mathrm{Re}\{a+bi\} = a$

Optics

$\mathbf{E}$	Electric field vector
$\mathbf{H}$	Magnetic field vector
n	Index of refraction
c	Speed of light in vacuum
$\mathscr{U}$	Scalar field of light. In this thesis it will represent the length of the electric field vector $\mathbf{E}$
f	Frequency of the light in $\mathrm{s^{-1}}$
ω	Angular frequency of a time harmonic solution; $\omega = 2\pi f$
λ	Wavelength of the light
k	Wave number, $k = \frac{2\pi}{\lambda}$
$\mathbf{k}$	Wave vector: $\|\mathbf{k}\| = k$
U	Complex amplitude
ϕ	Phase factor

B-Splines

Ξ	Knot vector with entries ξ_i
$N_{i,p}(\xi)$	B-spline basis function of order p
$\mathbf{B}_i$	B-spline curve control points
$\{\mathbf{B}_{i,j}\}$	B-spline surface control net

Neural Networks

$\mathbf{x}$	Input vector
$\sigma(\cdot)$	Activation function
$\mathbf{w}$	Weight vector

θ	Bias
$\mathscr{P}$	Perceptron
W	Weight matrix
$\boldsymbol{\theta}$	Bias vector
$\mathscr{N}$	Multilayer perceptron
$\mathbf{o}_i$	Intermediate outputs
net_i	Pre-activation intermediate outputs
$\hat{\mathbf{y}}_p$	Prediction, output
$\mathbf{y}_p$	Desired output
E	Error
η	Learning rate

2

Theory

2.1. Introduction

In this chapter the three basic topics this thesis relies on will be introduced. Firstly, optics is introduced. This section will start from Maxwell's equations and build up to the Fraunhofer diffraction integral. Secondly, B-splines will be introduced and expanded to B-spline surfaces. Finally, neural networks will be discussed, specifically how MLP's, a type of neural networks, work.

2.2. Optics

Optics is the study of light and how it interacts with matter. Phenomena within optics can be described with ray optics and physical optics/wave optics. In this thesis the focus is on physical optics. Specifically, Fraunhofer diffraction, which will be derived, as it will be used throughout this thesis. To start with this, first, Maxwell's equations are presented. From these equations the wave equation will be derived. Secondly, this is used to construct a superposition of plane waves using the angular spectrum. With plane waves a superposition integral will be constructed. Finally, this integral will be used to derive the Fraunhofer diffraction integral.

2.2.1. Maxwell's equations and the wave equation

Physical optics is based around classical electrodynamics. To obtain the wave like nature of light, consider Maxwell's equations 2.1 - 2.4 [3, p. 36]. These equations relate the electric and magnetic field to each other locally. In these equations $\mathbf{E} = \mathbf{E}(x, y, z)$ is the electric field and $\mathbf{H} = \mathbf{H}(x, y, z)$ is the magnetic field. The $\nabla\times$-operator is the curl operator and the $\nabla\cdot$-operator is the divergence operator, both in 3D Cartesian coordinates. μ denotes the magnetic permeability in a certain medium, in vacuum it would be μ_0. Likewise, ϵ denotes the permittivity of the medium, and ϵ_0 denotes the permittivity in vacuum. Here, we consider Maxwell's equations in media, so μ and ϵ are used.

$$\nabla \times \mathbf{E} = -\mu \frac{\partial \mathbf{H}}{\partial t} \tag{2.1}$$

$$\nabla \times \mathbf{H} = \epsilon \frac{\partial \mathbf{E}}{\partial t} \tag{2.2}$$

$$\nabla \cdot \epsilon \mathbf{E} = 0 \tag{2.3}$$

$$\nabla \cdot \mu \mathbf{H} = 0 \tag{2.4}$$

If we now apply the curl operator $\nabla\times$ to both sides of equation 2.1, we will obtain the result in equation 2.5. If we assume that the medium is linear, isotropic and homogeneous, we will obtain equation 2.6. In this equation $n = \left(\frac{\epsilon}{\epsilon_0}\right)^{1/2}$ and $c = \frac{1}{\sqrt{\mu_0 \epsilon_0}}$ [3, p. 36].

$$\nabla \times \nabla \times \mathbf{E} = \nabla(\nabla \cdot \mathbf{E}) - \nabla^2 \mathbf{E} \tag{2.5}$$

$$\nabla^2 \mathbf{E} - \frac{n^2}{c^2} \frac{\partial^2 \mathbf{E}}{\partial t^2} = 0 \tag{2.6}$$

We have now derived the wave equation for the electric field, but analogously the equation also holds for the magnetic field [3, p. 37]. In this thesis only linearly polarised light is considered, that is, the electric field vector is confined to a single plane along the direction of propagation. Therefore, if the plane is known, only the length of the vector is needed to fully describe the electric field vector. Hence, the wave equation can be rewritten in terms of a scalar quantity $\mathscr{U}$ which denotes the length of the electric field vector. Since the magnetic field is related to the electric field and can be found with Maxwell's equations, we only consider the electric field part from now on. The scalar wave equation is given in equation 2.7. This description of using a single scalar variable to model light paves the way for scalar diffraction optics in the next sections.

$$\nabla^2 \mathscr{U} - \frac{n^2}{c^2} \frac{\partial^2 \mathscr{U}}{\partial t^2} = 0 \tag{2.7}$$

If we assume that the light we use has a fixed frequency f, with corresponding angular velocity $\omega = 2\pi f$, we can use a time harmonic solution as an ansatz, displayed in equation 2.8:

$$\mathscr{U}(\mathbf{r}, t) = \mathrm{Re}\left\{ U(\mathbf{r}) e^{-i\omega t} \right\} \tag{2.8}$$

In this equation, $U \in \mathbb{C}$ is called the complex amplitude and $\omega = 2\pi f$. If we substitute this into the wave equation in 2.7 we will obtain the Helmholz equation in equation 2.9 [3, p. 39]:

$$\nabla^2 U + \frac{n^2 \omega^2}{c^2} U = 0 \Rightarrow \nabla^2 U + k^2 U = 0 \tag{2.9}$$

where $k = \frac{2\pi}{\lambda}$ and where λ is the wavelength. Note that we omitted taking the real part in this equation. This equation describes the dynamics of the **complex** amplitude U. The Helmholz equation can be used to derive time-independent solutions of the wave equation, assuming a harmonic solution in time.

2.2.2. Angular Spectrum of Plane waves

With the earlier derived Helmholtz equation (Eq. 2.9), we can derive a plane wave solution in Cartesian coordinates. If we use cylindrical or spherical coordinates, other types of waves can be found, but by taking superpositions of them, one can obtain similar solutions. In Cartesian coordinates separation of variables will lead to plane waves in equation 2.10:

$$U(x, y, z) = A e^{i(k_x x + k_y y + k_z z)} \tag{2.10}$$

Here, $A \in \mathbb{R}$ represents the amplitude. k_x, k_y, k_z are the components of the wave vector $\mathbf{k}$ for which we have: $\|\mathbf{k}\| = \frac{2\pi}{\lambda}$. This solution is a plane wave that travels in the direction of the $\mathbf{k}$-vector and has its wavefront in the plane perpendicular to this vector.

Light can be seen as being composed of many plane waves, each with its own amplitude and wave vector. To account for this, we construct the solution by taking a superposition of waves and integrating over the respective wave vector components in equation 2.11 [3, p. 57-58]:

$$U(x, y, z) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \tilde{U}_0(k_x, k_y) e^{ik_x x + ik_y y} e^{\pm iz\sqrt{\|\mathbf{k}\|^2 - k_x^2 - k_y^2}} \, \mathrm{d}k_x \, \mathrm{d}k_y \tag{2.11}$$

Here, $\tilde{U}_0(k_x, k_y)$ is a weighting factor that describes how much a certain plane wave contributes to the solution. Also, as $\|\mathbf{k}\|, k_x, k_y$ is enough to fully describe the direction of a plane wave, we can find the following from this:

$$k_z = \pm\sqrt{\|\mathbf{k}\|^2 - k_x^2 - k_y^2} \tag{2.12}$$

Note, that the positive sign is used to indicate that a plane wave is propagating in the $+z$-direction and a negative sign is used to indicate that the plane wave is propagating in the $-z$-direction. In addition, $\sqrt{\|\mathbf{k}\|^2 - k_x^2 - k_y^2}$

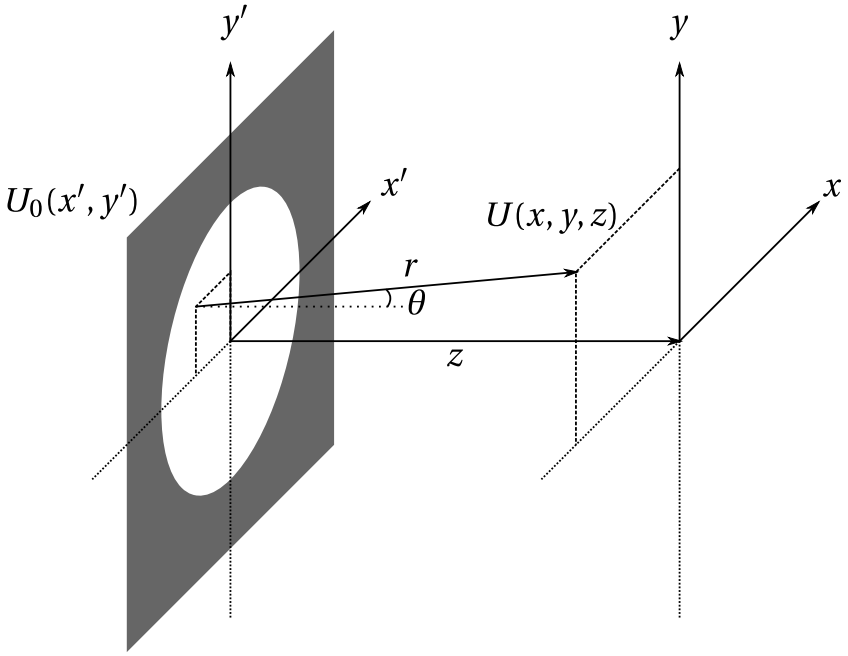


Figure 2.1: Schematic of the coordinate systems used in the Rayleigh-Sommerfeld diffraction integral.

can yield an imaginary number if $\|\mathbf{k}\|^2 < k_x^2 + k_y^2$. If this is the case, the resulting wave will be an evanescent wave that decays exponentially in the $+z$- or $-z$-direction depending on the sign of k_z [3, p. 58].

We can introduce a boundary condition at $z = 0$ to determine $\tilde{U}_0(x, y)$. Let the boundary condition be defined as follows: $U(x, y, 0) = U_0(x, y)$. This yields the following in equation 2.13.

$$U(x, y, 0) = U_0(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \tilde{U}_0(k_x, k_y) e^{ik_x x + ik_y y} \, \mathrm{d}k_x \, \mathrm{d}k_y \tag{2.13}$$

Not surprisingly, we recognize the 2D Fourier transform. So, we have found $\tilde{U}_0(k_x, k_y) = \mathscr{F}\{U_0(x, y)\}$. The $\tilde{U}_0(k_x, k_y)$ is called the angular spectrum of $U_0(x, y)$.

2.2.3. Rayleigh-Sommerfeld diffraction integral

When we introduced the boundary condition in equation 2.11, we found a away of determining the field at any point in space. However, $\tilde{U}_0$ is the Fourier transform of U_0. So, in fact we have a quadruple integral expression for U. To use this for simulations is computationally expensive. In fact, an equivalent formulation is the Rayleight-Sommerfeld diffraction integral. This integral will yield identical fields as the angular spectrum approach [3, p. 61]. This diffraction integral can be derived from the Helmholz equation, see equation 2.9. For the derivation, the reader is referred to Goodman [3, §3.3-§3.5]. The integral is defined as follows:

$$U(x, y, z) = \frac{1}{i\lambda} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} U_0(x', y') \frac{z}{r} \frac{e^{i\|\mathbf{k}\| r}}{r} \, \mathrm{d}x' \, \mathrm{d}y' \tag{2.14}$$

where we define $r \equiv \sqrt{(x - x')^2 + (y - y')^2 + z^2}$ and λ is the wavelength. The primed coordinates x' and y' specify the plane at $z = 0$. This plane is also called the pupil plane.

With the definition of r, one can readily recognize that the integral is essentially a superposition of spherical waves: $\frac{e^{i\|\mathbf{k}\| r}}{r}$ with amplitude $\tilde{U}_0(k_x, k_y)\frac{z}{r}$. Note that $\frac{z}{r} = \cos\theta$, with θ the angle between the z-axis and the line connecting to (x, y, z). To further illustrate this, consider figure 2.1 in which the primed and non-primed coordinate systems are drawn.

2.2.4. Fraunhofer diffraction integral

To make things easier, we mostly consider the field far away from any sources. This is called the far field. Specifically, we assume that z is very large. If we express r differently, by taking out the z^2, we obtain:

$r = z\sqrt{1+((x-x')^2+(y-y')^2)/z}$. Using the fact that z is large, we can do a Taylor expansion of $\sqrt{1+s}$ for s small. This gives [4, p. 55]:

$$r \approxeq z + \frac{(x-x')^2+(y-y')^2}{2z} = z + \frac{x^2+y^2}{2z} + \underbrace{\frac{x'^2+y'^2}{2z}} - \frac{xx'+yy'}{z} \approxeq + \frac{x^2+y^2}{2z} - \frac{xx'+yy'}{z}$$

At the brace we assume that x', y' are both small in comparison to $2z$, so we can set that term to zero. We now substitute this expression in the exponent of equation 2.14 and we set $r \approxeq z$ in the rest of the integral. This yields the Fraunhofer diffraction integral after some rearranging in equation 2.15 [3, p. 74]:

$$U(x,y,z) = \frac{e^{ikz}e^{ik\frac{x^2+y^2}{2z}}}{i\lambda z}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} U_0(x',y')e^{-ik\frac{xx'+yy'}{z}}\,\mathrm{d}x'\,\mathrm{d}y' = \frac{e^{ikz}e^{ik\frac{x^2+y^2}{2z}}}{i\lambda z}\mathscr{F}\{U_0\}\Big(\frac{x}{\lambda z},\frac{y}{\lambda z}\Big) \tag{2.15}$$

This diffraction integral can be easily computed, as it is effectively a Fourier transform. The use of coordinate systems (x', y') and (x, y, z) is the same as in figure 2.1. It is important to emphasize that the Fraunhofer diffraction integral is valid in the far field. More quantitatively, see the criterion in equation 2.16, where D is the characteristic size for U_0. In other words, the approximate size over which U_0 is nonzero [3, p. 74].

$$z > \left(\frac{D}{\lambda}\right)^2 \tag{2.16}$$

2.2.5. Discretisation of Fraunhofer Diffraction

To be able to numerically simulate Fraunhofer diffraction, the continuous Fourier transform needs to be discretised. To this end, the Fourier transform is approximated by a Riemann sum, which in turn will lead to the discrete Fourier transform [4, p. 18-20]. So for a function $g(x, y)$, we can discretise it as $g[n, m] = g(n\Delta x, m\Delta y)$, for some sampling interval lengths Δx and Δy. In total:

$$\mathscr{F}\{g\}(\xi,\eta) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} g(x,y)e^{-ix\xi}e^{-iy\eta}\,\mathrm{d}x\,\mathrm{d}y \tag{2.17}$$

$$\approx \int_{-L_y/2}^{+L_y/2}\int_{-L_x/2}^{+L_x/2} g(x,y)e^{-ix\xi}e^{-iy\eta}\,\mathrm{d}x\,\mathrm{d}y \tag{2.18}$$

$$\approx \sum_{n=-N/2}^{N/2-1}\sum_{m=-M/2}^{M/2-1} g\big(n\Delta x, m\Delta y\big)e^{-in\Delta x\xi}e^{-im\Delta y\eta}\Delta x\Delta y \tag{2.19}$$

where the Fourier transform is taken on $N \times M$ sampled points in the x- and y-direction, respectively. In addition, we assume that the function is approximately zero outside the rectangle $[-L_x/2, L_x/2] \times [-L_y/2, L_y/2]$. This gives that $\Delta x = \frac{L_x}{N}$ and $\Delta y = \frac{L_y}{M}$. By convention, the continuous frequency space is divided in evenly spaced values: $\xi = \frac{2\pi p}{N\Delta x}$ and $\eta = \frac{2\pi q}{M\Delta y}$, where p and q are the indices along the ξ- and η-direction, respectively. This yields the discrete Fourier transform [4, p. 20]:

$$\mathscr{F}_D\{g\}(p,q) = \sum_{n=-N/2}^{N/2-1}\sum_{m=-M/2}^{M/2-1} g[n,m]e^{-i2\pi\frac{pn}{N}}e^{-i2\pi\frac{qm}{M}}\frac{L_xL_y}{NM} \tag{2.20}$$

With the discrete Fourier transform at hand, the Fraunhofer diffraction integral of equation 2.15 can be approximated in the following way:

$$U(x,y,z) = \frac{e^{ikz}e^{ik\frac{x^2+y^2}{2z}}}{i\lambda z}\mathscr{F}\{U_0\}\Big(\frac{x}{\lambda z},\frac{y}{\lambda z}\Big) \tag{2.21}$$

$$\approx \frac{e^{ikz}e^{ik\frac{x^2+y^2}{2z}}}{i\lambda z}\mathscr{F}_D\left\{U_0\left(n\frac{L_x}{N},m\frac{L_y}{M}\right)\right\}\left(k=\left\lfloor\frac{x}{\lambda z}\frac{L_x}{2\pi}\right\rfloor, l=\left\lfloor\frac{y}{\lambda z}\frac{L_y}{2\pi}\right\rfloor\right) \tag{2.22}$$

where $k = \left\lfloor\frac{x}{\lambda z}\frac{L_x}{2\pi}\right\rfloor$ and $l = \left\lfloor\frac{y}{\lambda z}\frac{L_y}{2\pi}\right\rfloor$ are indices into the discrete Fourier transform. This approximation has some interesting properties that are essential to understand when doing numerical simulations.

Firstly, consider the largest x we can have, that is $k = N/2-1$: $\frac{x}{\lambda z} = \frac{2\pi}{L_x}(N/2-1)$. This gives that the largest x is $x_{\max} = \lambda z \frac{2\pi}{L_x}(N/2-1)$. From this we can observe that a larger N will result in a larger $x_{\max}$. In other words, increasing the number of sampled points will increase the physical area being observed in the far field.

Secondly, suppose some feature has size $d = \frac{\Delta x}{\lambda z}$ in the continuous Fourier transform based Fraunhofer diffraction along some axis; then it encompasses index range $\Delta k = \frac{\Delta x}{\lambda z}\frac{L_x}{2\pi}$. This relation is dependent on L_x, the size of the field that is to be Fraunhofer diffracted. Consequently, enlarging the size of the field to be Fraunhofer diffracted results in larger index ranges for a certain feature and thus more detail.

With these two properties the physical size and the amount of detail in the simulation can be tuned.

2.2.6. Altering the Phase of Incoming Light

In the previous section a certain initial distribution $U_0(x,y)$ was propagated in the positive z-direction. This initial distribution can be anything from an incoming plane wave or a complicated setup involving multiple (in-)coherent sources. To modify the incoming light and adjust it to our liking, we can modify the phase of light in $z = 0$ by using a lens. This lens can be a classic spherical shape as used in paraxial light propagation or a shape without any rotational symmetry, a so-called freeform lens. The lens can be described as having a certain thickness $d(x',y')$ at some location (x',y'). In media light has a lower phase velocity than in vacuum. That is, the propagation velocity of the phase of certain frequency component of the wave is slower than in vacuum. Suppose that in the lens medium light has phase velocity: c_m. The time it takes the phase of a certain frequency component of light to propagate through the medium is then: $\Delta t = \frac{d}{c_m}$. This results in the following phase shift ϕ:

$$\phi\big(d(x',y')\big) = 2\pi\Delta t f = 2\pi\frac{d(x',y')f}{c_m} = \frac{2\pi d}{\lambda_m} = k_m d = k_0 n_m d\big(x',y'\big) \tag{2.23}$$

where we define f to be the frequency of the wave, λ_m the wavelength of the wave in the lens medium, k_m the wave vector of the wave in the lens medium and k_0 the wave vector of the wave in vacuum. For a certain incoming field $U_{\text{before}}(x',y')$ we get after the lens, the field [3, p. 97]:

$$U_{\text{after}}\big(x',y'\big) = U_{\text{before}}\big(x',y'\big)e^{i\phi(d(x',y'))} = U_{\text{before}}\big(x',y'\big)e^{ik_0 n_m d(x',y')} \tag{2.24}$$

Important to note that this only works in the limit of a thin lens, as otherwise translation of rays would become evident due to Snell's law of refraction [3, p. 97]. The $\phi(d(x',y'))$ is called the phase distribution of the lens. Additionally, if the lens is smaller than the nonzero region of the incoming field U_{before}, an aperture function $P(x',y')$ is needed to constrain what light passes through. This aperture function is 0 outside the lens and 1 inside the lens [3, p. 102]. To illustrate this, consider a circular lens on which an infinite plane wave is incident. Then only the light that passes through the circular lens is of interest, the rest of the light is blocked.

If we consider a thin lens with spherical surfaces, so lenses in the paraxial approximation, the phase distribution of the lens becomes dependent on the focal length f of the lens. According to Goodman [3, p. 99] the phase distribution becomes (excluding constant factors):

$$\phi(x',y') = -\frac{k}{2f}\big(x'^2 + y'^2\big) \tag{2.25}$$

Note that it is assumed that the lens is positive, i.e. it has positive focal length f. If we apply this phase distribution to incoming light $U_0(x',y')$ and then propagate this in the positive z-direction until the focal length f of the lens with Fresnel diffraction (based on Rayleigh-Sommerfeld diffraction), we obtain the following (neglecting the aperture function) [3, p. 103]:

$$U(x,y,f) = \frac{e^{i\frac{k}{2f}(x^2+y^2)}}{i\lambda f}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} U_0(x',y')e^{-i\frac{2\pi}{\lambda f}(xx'+yy')}\,\mathrm{d}x'\,\mathrm{d}y' = \frac{e^{i\frac{k}{2f}(x^2+y^2)}}{i\lambda f}\mathscr{F}\{U_0\}\left(\frac{x}{\lambda f},\frac{y}{\lambda f}\right) \tag{2.26}$$

From this equation it becomes apparent that one can use a lens to obtain a Fourier transform of the field without letting the field propagate to the far field, as would be required in Fraunhofer diffraction, see equation 2.16 for the propagation distance criterion of Fraunhofer diffraction.

Without lenses it is also possible to add a certain phase distribution to an incoming field. This can be done with so-called spatial light modulator devices, or SLMs for short. These devices have a certain resolution and in each pixel they can modify the incoming phase of the field with a predefined phase shift. This allows one to replicate the effect of a lens by setting the phase distribution.

2.3. B-Splines Surface Description

Surfaces in 3D can be described in a number of ways. For instance, with implicit formulas, like the unit sphere: $x^2 + y^2 + z^2 = 1$. However, in this thesis the aim is to describe phase distributions as a surface. Therefore, a multi-valued function such as an implicit formula is of no use. Hence, surfaces described by functions of the form $f : \mathbb{R}^2 \to \mathbb{R}$ will be used. To build surfaces of this form, B-spline surfaces will be used. This method has been chosen, because it allows describing smooth surfaces using a discrete number of so-called control points. B-splines are in essence piecewise continuous polynomials, with higher order continuity between pieces when desired.

In what follows, we first describe knot vectors and basis functions. With these preliminaries, B-spline curves can be described using control points. Finally, this can be extended to B-spline surfaced where a control net of points is used, i.e. a 2D mesh of control points.

2.3.1. Knot vector

To describe splines, a parameter space is needed. This space can be seen as the 'domain' of the polynomials. The parameter space is an interval in $\mathbb{R}$. Since splines are piecewise polynomial, the parameter space needs to be partitioned in subdomains for each specific part. To do this, a knot vector is used. Specifically, a knot vector is a non-decreasing set of scalar coordinates that partition the parameter space [5, p. 19]. It is denoted as $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, where $\xi_i \in \mathbb{R}$ is called the i^{th} knot. Here, i is the knot index and it ranges from 1 to $n + p + 1$, with p denoting the polynomial order and n denoting the number of polynomial pieces and basis functions.

Knot vectors are called uniform if the knots are equally spaced. In contrast, non-uniform knot vectors have unequally spaced knots [5, p. 20]. A knot vector is called open if the first and last knot value is repeated $p + 1$ times. Open knot vectors result in interpolation in the endpoints of the spline [5, p. 20].

2.3.2. Basis Functions

With the defined knot vector the basis function of B-splines can be defined using the Cox-de Boor recursion formula [5, p. 21] in equations 2.27 and 2.28. The recursion start with $p = 0$, a step function in parameter space:

$$N_{i,0}(\xi) := \begin{cases} 1 & \text{if } \xi_i \le \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.27}$$

For higher values of p, the recursion is given by:

$$N_{i,p}(\xi) := \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi i + 1} N_{i+1,p-1}(\xi) \tag{2.28}$$

Important to note is that with open knot vectors knots at the start and end are repeated. This can result in a division by zero as $\xi_{n-1} = \xi_n$ may occur. Therefore we define: $\frac{0}{0} \doteq 0$.

In addition, repeating knots allow for controlling the continuity of basis functions and their derivatives across knots in the following way: Basis function of order p are C^{p-m_i} continuous across a knot ξ_i with m_i denoting the multiplicity of the value of the knot in the knot vector. As a consequence, a multiplicity of $p + 1$ will result in a discontinuity [5, p. 23].

To further illustrate these basis functions, consider the plot in figure 2.2. In this figure multiple basis functions have been plotted with order 0, 1 and 2. Important to observe is that each basis function is nonnegative and that each basis function of order p has $p - 1$ continuous derivatives across the knots. For instance, $N_{1,2}$ has a continuous derivative in $\xi = 0, 1, 2, 3$. In addition, each basis function of order p is supported by $p + 1$ knot

spans [5, p. 22]. To illustrate this, consider $N_{2,2}$. This function is nonzero between $\xi = 1$ and $\xi = 4$, which contains 3 units of knot spans.

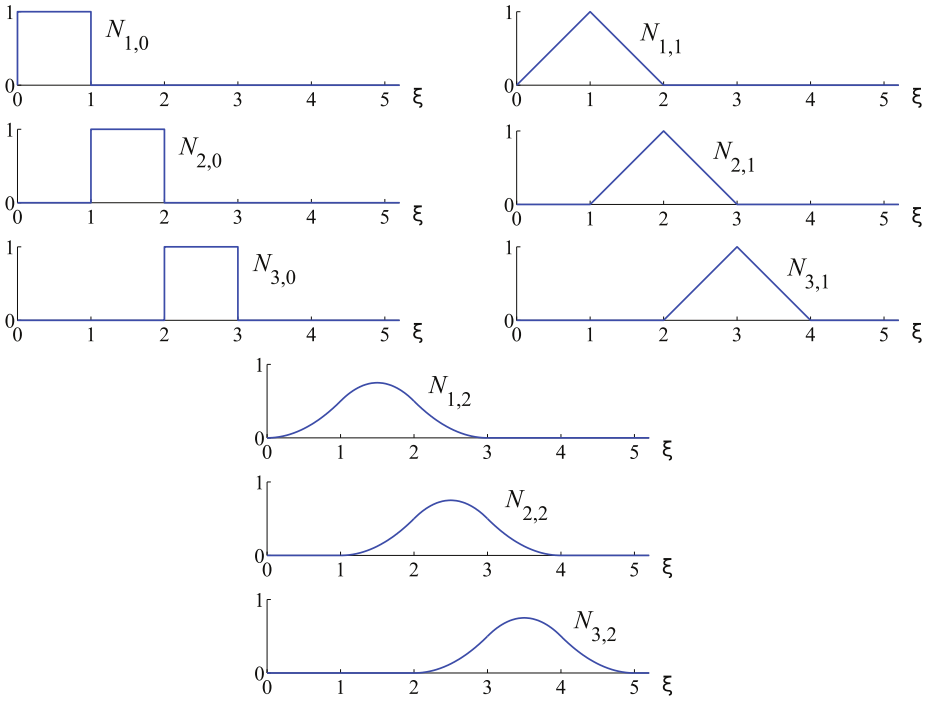


Figure 2.2: Plot of basis function in parameter space of orders 0, 1 and 2 using a uniform knot vector $\Xi = \{0,1,2,3,4,\ldots\}$. This figure has been extracted from [5, p. 21, figure 2.3].

2.3.3. B-Spline Curves

With the basis functions at hand, B-spline curves can be defined as explained in the following: B-spline curves are constructed by taking a weighted linear combination of B-spline basis functions. The weights in this linear combination are called the control points. More formally, given n basis functions $N_{i,p}$ for $i = 1,\ldots,n$ and the corresponding control points $\mathbf{B}_i \in \mathbb{R}^d$ again for $i = 1,\ldots,n$, the B-spline curve in d-dimensions is given by [5, p. 29]:

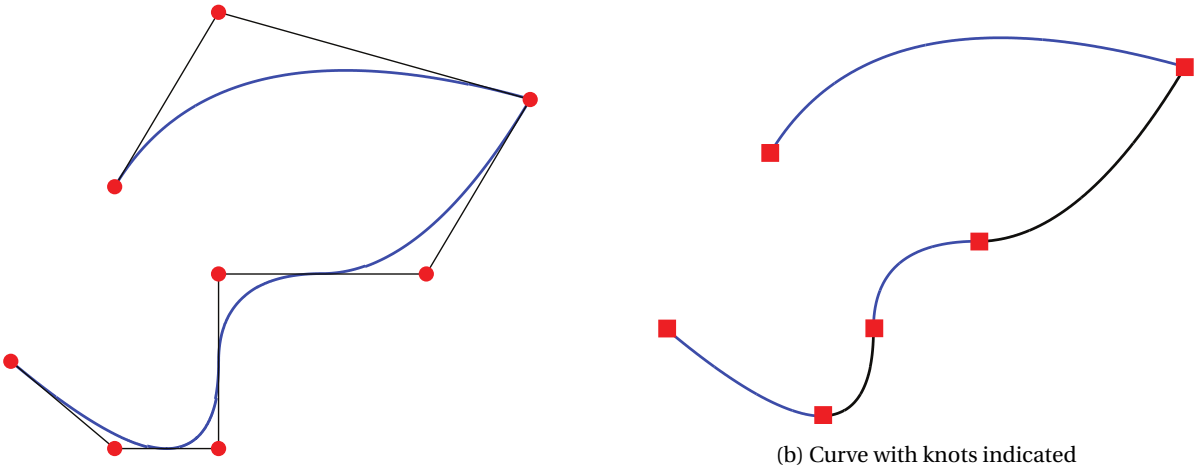
$$\mathbf{C}(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi)\mathbf{B}_i \tag{2.29}$$

Note that in the basis functions, a knot vector Ξ is used. As stated in section 2.3.1, the knot vector has length $n+p+1$, where n is the number of polynomial pieces, which is equal to the number of control points $\mathbf{B}_i$, and p is the order of the polynomials. Additionally, as previously stated, open knot vectors result in interpolation at the endpoints. Interpolation at intermediate points is also possible, by repeating knot values $p+1$ times [5, p. 29].

To illustrate what a B-spline curve may look like and what the repetition of knot values does, consider figure 2.3. In this figure an open knot vector is employed, resulting in interpolation at the endpoints of the curve. In addition, knot value $\xi = 4$ has been repeated 2 times, resulting in interpolation at the corresponding control point, as the B-spline curve is of order 2.

2.3.4. B-Spline Surfaces

To construct B-spline surfaces a tensor product of B-spline curves is taken. This time, the control points are arranged in a control net $\{\mathbf{B}_{i,j}\}$ with $i = 1,2,\ldots,n$ and $j = 1,2,\ldots,m$. If polynomial orders p and q are used, the knot vectors are defined as: $\Xi = \{\xi_1,\xi_2,\ldots,\xi_{n+p+1}\}$ and $\mathscr{H} = \{\eta_1,\eta_2,\ldots,\eta_{m+q+1}\}$. Then, the B-spline surface is defined as [5, p. 32]:



(a) Curve with control points

(b) Curve with knots indicated

Figure 2.3: This is a plot of a B-spline curve in 2D with knot vector $\Xi = \{0,0,0,1,2,3,4,4,5,5,5\}$. In (a) the curve is shown with its control points as red circles and in (b) the knots are shown as red squares. These figures have been extracted from [5, p. 29, figure 2.10].

$$\mathbf{S}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) \mathbf{B}_{i,j} \tag{2.30}$$

where $N_{i,p}(\xi)$ and $M_{j,q}(\eta)$ are the B-spline basis function of order p and q that correspond to knot vector Ξ and knot vector $\mathscr{H}$, respectively.

2.4. Neural Networks

Neural networks are a family of powerful techniques inspired by biological neurons that can be used to solve a wide range of tasks. Neural networks are a central technique in the field of machine learning. Due to their nature, neural networks are well-equipped in use as universal function approximators [6, p. 98]. This property makes neural networks especially apt for predictions on the behaviour of physical systems, as neural networks can be optimized to approximate the underlying dynamics of those systems.

In this section neural networks, and specifically multilayer perceptrons will be introduced by starting at their constituents: perceptrons. From this, multilayer perceptrons, a kind of neural networks, will be assembled. Next, to optimize these kinds of networks, a process called training is needed. In this section the optimization problem is posed. After this, gradient descent is described, a technique that can be used for optimizing. Finally, physics-informed neural networks will be discussed.

2.4.1. Perceptrons

Biological neurons take inputs through dendrites, process them and then output through axons which connect to other neurons. In artificial neurons, this principle is similar. The focus is specifically on so-called perceptrons, which are a kind of artifical neurons [6, p. 81]. A schematic of a perceptron is shown in figure 2.4. In this figure, the x_i's depict inputs to the perceptron, which are then weighted using the respective weights w_i's. To this, a bias θ is added.

The perceptron then performs its 'activation' σ, a kind of processing of the inputs, which yields the final output. More rigorously, a perceptron $\mathscr{P} : \mathbb{R}^n \to \mathbb{R}$ is defined as $(x_1, x_2, \ldots) \mapsto \sigma(\sum x_i w_i - \theta) = \mathbf{w}^T\mathbf{x} - \theta$, where $\theta \in \mathbb{R}$, $\mathbf{w} = (w_1, w_2, \ldots) \in \mathbb{R}^n$ and $\mathbf{x} = (x_1, x_2, \ldots) \in \mathbb{R}^n$ [6, p. 6].

Important to note is that the activation function σ is what introduces the non-linear behaviour of perceptrons. These functions usually map real numbers into the range $(0,1)$ or $(-1,1)$ [6, p. 6].

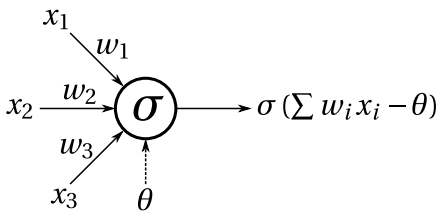


Figure 2.4: Schematic depiction of an artificial neuron.

2.4.2. Multilayer Perceptron

Just as in biology, arranging multiple neurons into a network structure creates a powerful instrument that can be used to do complex computations. In this thesis, the focus is on feed-forward neural networks. That is, the network will have clearly defined inputs and outputs and internally the connections between neurons will only go one way. Usually, a feed-forward neural network is organized in layers, i.e. there is no connection between neurons in the same layer. Such a network is called fully connected, if each neuron in a specific layer is connected to all neurons in the next layer [6, p. 9]. Important to note here is, that types of neurons, other than the one described above may be employed. If however, perceptrons are used, and the perceptrons are arranged in layers, the resulting network is called a muiltilayer perceptron or MLP for short.

To illustrate a simple MLP, a small fully connected feed-forward neural network with perceptrons as neurons is shown in figure 2.5. In this figure, two inputs are together denoted as the vector $\mathbf{x}$ and three outputs are together denoted as the vector $\mathbf{y}$. The inputs are fed into the first 'hidden' layer. Each connection has a weight. These weights are in matrix $W_1 = [\mathbf{w}_1^2\, \mathbf{w}_1^2\, \mathbf{w}_2^3]$, where the indices of the weight vector $\mathbf{w}_j^i$ denote to which perceptron it belongs, specifically, the subscript indices denote the layer and the superscript indices denote the perceptron in that layer. So $\mathbf{w}_1^2$ corresponds to the first perceptron in the hidden layer, and is defined as $\mathbf{w}_1^2 = (w_1, w_2) \in \mathbb{R}^2$, where w_1 is the weight that acts on the first input and w_2 is the weight that acts on the second input, just as defined in the previous subsection on perceptrons. Each perceptron has an activation function σ and a bias, which is denoted here with a vector $\theta_1 = (\theta_1^1, \theta_1^2, \theta_1^3)$ of size 3, one component for each perceptron. After the perceptrons activated their inputs, the output of this layer is again fed to the next layer with weights $W_2 = [\mathbf{w}_2^1\, \mathbf{w}_2^2\, \mathbf{w}_2^3]$ and biases $\theta_2 = (\theta_2^1, \theta_2^2, \theta_2^3)$ for the perceptrons. This layer yields the final output $\hat{\mathbf{y}}$ of size 3. Hence, this network has function signature $\mathcal{N} : \mathbb{R}^2 \to \mathbb{R}^3$ and definition: $\mathbf{x} \mapsto \sigma(W_2^{\mathrm{T}} \sigma(W_1^{\mathrm{T}} \mathbf{x} - \theta_1) - \theta_2)$, where σ is elementwise applied to vectors [6, p. 97].

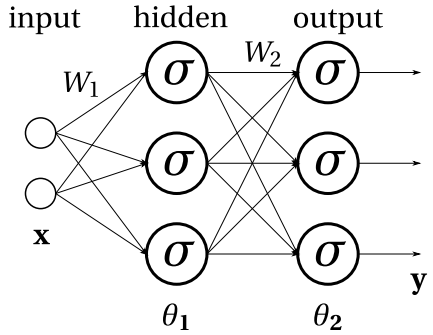


Figure 2.5: Schematic of a fully connected feed-forward neural network with input $\mathbf{x}$ and output $\mathbf{y}$. It has signature: $\mathcal{N} : \mathbb{R}^2 \to \mathbb{R}^3$.

In the example network in the figure each perceptron has the same activation function σ, but this is not necessarily the case. Any combination of activation functions may be used. However, usually the activation function used by perceptrons in a specific layer is the same. As discussed in the previous section, the activation function is what introduces the non-linearity in perceptrons. Even more so, it is vital in MLPs to have non-linear activation functions when a universal function approximator is needed [6, p. 98].

In general, an MLP can be defined with intermediate outputs $\mathbf{o}_i$ and pre-activation intermediate outputs net_i, with i the index of the layer. So, $\mathbf{o}_0 = \mathbf{x}$ for the input layer (zero-based indexing) and $\hat{\mathbf{y}} = \mathbf{o}_M$ for the last

layer at index M. The intermediate outputs are then defined recursively by [6, p. 97]:

$$\mathbf{o}_i = \sigma_i(\text{net}_i) = \sigma_i(W_i^{\mathrm{T}}\mathbf{o}_{i-1} + \theta_i) \quad \forall i = 1,2,3,\ldots,M \tag{2.31}$$

where W_i and θ_i are defined in the same way as in the example and σ_i is the activation function, which can change per layer.

2.4.3. Training

As described in the introduction of this section, neural networks can be optimized to approximate the dynamics of physical systems. In machine learning the process of optimization done by a specific algorithm to a network is usually called 'learning' or 'training' of the network. For training to be possible a network needs to have adjustable parameters. In the case of MLPs, each perceptron is described by a bias θ, a weight vector $\mathbf{w}$ and an activation function σ. The activation function stays usually fixed, while the bias and weight vector can be freely changed by an optimization algorithm. As with all sorts of optimization some sort of objective function is needed. In the case of MLPs, the mean squared error (MSE) is usually employed, where the error is taken between the network output $\mathcal{N}(\mathbf{x}_p) = \hat{\mathbf{y}}_p$ and the desired output $\mathbf{y}_p$ for $(\mathbf{x}_p, \mathbf{y}_p) \in \mathcal{S}$. This set $\mathcal{S}$ represents the training pairs used for training. The error then becomes [6, p. 100]:

$$E = \frac{1}{N}\sum_{p\in\mathcal{S}} E_p = \frac{1}{2N}\sum_{p\in\mathcal{S}} \left\|\hat{\mathbf{y}}_p - \mathbf{y}_p\right\|^2 \tag{2.32}$$

where N is the size of the set $\mathcal{S}$ and $E_p = \frac{1}{2}\left\|\hat{\mathbf{y}}_p - \mathbf{y}_p\right\|^2$. The resulting objective of this optimization problem is then:

$$\min_{\forall i: W^i, \theta^i} E \tag{2.33}$$

Note that in the previous objective function explicit training pairs, which are known in advance, are used. Formally the network is trained to mimic the behaviour of some function $f : \mathcal{X} \to \mathcal{Y}$ of which samples $\mathcal{S} = \{(\mathbf{x}_p, \mathbf{y}_p) : \mathbf{y}_p = f(\mathbf{x}_p), \mathbf{x}_p \in \mathcal{X}\}$ are known. This learning task is known as supervised learning [6, p. 25]. In contrast, unsupervised learning is a learning task without 'targets', meaning that only the input values are available [6, p. 26]. For such learning tasks a different expression for the error is employed, one that is only dependent on the input values and the output values of the network. In the section about physics-informed neural networks, this will be made more concrete with an example.

2.4.4. Gradient Descent

An important technique used for optimizing the objective function is gradient descent. This technique will also be employed in this thesis when training networks. This technique uses the gradient of the error with respect to a parameter of the network to determine how to update that parameter. Formally, it updates a parameter q with Δq, with Δq calculated from [6, p. 100]:

$$\Delta q \doteq -\eta \frac{\partial E}{\partial q} \tag{2.34}$$

The η in this equation is called the learning rate and determines how much a parameter is changed in each update. After a parameter is updated, the error is reevaluated and a new update occurs. This process continues iteratively until the error reaches a minimum, i.e. after each update the error does not change substantially. What 'substantially' means is dependent on what stopping criterion is chosen. An example stopping criterion can be to halt when no improvement in the error for seven updates is detected.

To illustrate the intuition behind these updates, consider figure 2.6. In this figure some error E is plotted as a function of the parameter q. The optimization starts at q_1. The **negative** gradient at this point is depicted by the green arrow. This arrow is followed for a factor η to obtain a new point q_2. For q_3 the same is done, following the negative gradient of q_2 a factor η. In essence, gradient descent follows the gradient 'downwards', i.e. descending, in hopes of finding a minimum. Important to note is that this will be a **local** minimum, so not the global minimum. This is also directly a major shortcoming of gradient descent. The start value of q determines what optimum will be reached.

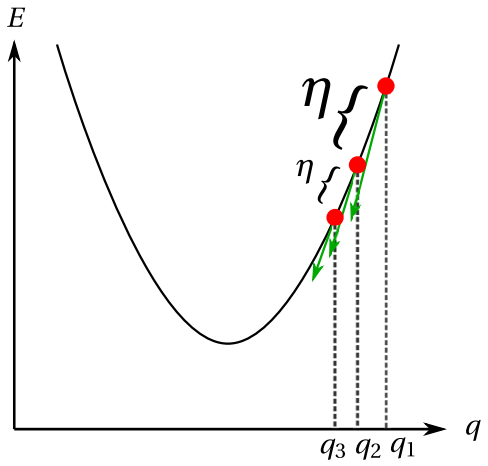


Figure 2.6: Gradient descent in 1D, shown here for two updates of q using equation 2.34, where it starts off at q_1. η is the learning rate.

To apply this to the MLP of the previous section, define the matrix $Q = [W_1\ W_2\ \dots\ W_M\ \theta_1\ \theta_2\ \dots\ \theta_M]$ of all adjustable parameters. The update of parameters becomes then like in equation 2.34, but for multiple parameters:

$$\Delta Q = -\eta \frac{\partial E}{\partial Q} = -\eta \frac{1}{N} \sum_{p \in \mathscr{S}} \frac{\partial E_p}{\partial Q} \tag{2.35}$$

If $\mathscr{S}$ is used to sum over, the update of Q is influenced by all training samples. This way of updating is called batched gradient descent. Another option is to use E_p directly as the error and update Q per training sample. This is called stochastic gradient descent, because it is more influenced by the characteristics of a single training sample [6, p. 104]. Both options can be used to train a neural network. The first one is slower, but yields more accurate updates at each step. The second one is much faster, but can yield inaccurate updates due to being dependent on a single training sample.

To illustrate how taking these kind of derivatives looks like, consider the following equations in which $\frac{\partial E_p}{\partial W_{M-1}^{(u,v)}}$ is calculated. $W_{M-1}^{(u,v)}$ occurs in the expression of $(\text{net}_{M-1})^v = \sum_m W_{M-1}^{(m,v)} (\mathbf{o}_{M-2})^m$, so the chain rule has to be used to get to this expression [6, p. 100]. In total:

$$\frac{\partial E_p}{\partial W_{M-1}^{(u,v)}} = \nabla_{\hat{\mathbf{y}}_p} E_p \cdot \frac{\partial \hat{\mathbf{y}}_p}{\partial W_{M-1}^{(u,v)}} = \sum_k \frac{\partial E_p}{\partial (\mathbf{o}_M)^k} \frac{\partial (\mathbf{o}_M)^k}{\partial W_{M-1}^{(u,v)}} \tag{2.36}$$

$$= \sum_k \Big((\mathbf{o}_M)^k - (\mathbf{y})^k \Big) \frac{\partial (\mathbf{o}_M)^k}{\partial (\text{net}_M)^k} \frac{\partial (\text{net}_M)^k}{\partial W_{M-1}^{(u,v)}} \tag{2.37}$$

$$= \sum_k \Big((\mathbf{o}_M)^k - (\mathbf{y})^k \Big) \dot{\sigma}\Big((\text{net}_M)^k \Big) \frac{\partial}{\partial W_{M-1}^{(u,v)}} \Big(\big(W_M^T \mathbf{o}_{M-1} + \theta_M \big)^k \Big) \tag{2.38}$$

$$= \sum_k \Big((\mathbf{o}_M)^k - (\mathbf{y})^k \Big) \dot{\sigma}\Big((\text{net}_M)^k \Big) \frac{\partial}{\partial W_{M-1}^{(u,v)}} \left[\sum_l (W_M)^{(l,k)} (\mathbf{o}_{M-1})^l + (\theta_M)^k \right] \tag{2.39}$$

$$= \sum_k \Big((\mathbf{o}_M)^k - (\mathbf{y})^k \Big) \dot{\sigma}\Big((\text{net}_M)^k \Big) \left[\sum_l (W_M)^{(l,k)} \dot{\sigma}\Big((\text{net}_{M-1})^l \Big) \frac{\partial (\text{net}_{M-1})^l}{\partial W_{M-1}^{(u,v)}} \right] \tag{2.40}$$

$$= \sum_k \underbrace{\Big((\mathbf{o}_M)^k - (\mathbf{y})^k \Big) \dot{\sigma}\Big((\text{net}_M)^k \Big)}_{\text{I}} \underbrace{(W_M)^{(v,k)} \dot{\sigma}\big((\text{net}_{M-1})^v \big)}_{\text{II}} (\mathbf{o}_{M-2})^u \tag{2.41}$$

In these equations k is used to index the vector $\mathbf{o}_M$ and l is used to index the result of $W_M^T \mathbf{o}_{M-1}$. Note that as $W_{M-1}^{(u,v)}$ only occurs in $(\text{net}_{M-1})^v$, the other derivatives will yield zero.

This expression is rather lengthy and is expensive to calculate. However, a technique called backpropagation can speed this up. Backpropagation works by caching the results of a so-called forward pass. A forward pass is essentially calculating the output of the network. During this forward pass intermediate results such as net_i and $\mathbf{o}_i$ are saved for later use. Then after a forward-pass has occurred, the intermediate results can be used to calculate the gradient of the error with respect to all the parameters. What really defines backpropagation is the caching of so-called 'layer errors', such as the one marked by underbrace (I). Each step deeper 'back' into the network adds an additional factor to this error, for instance the factor marked by underbrace (II). By caching these layer errors, the gradient with respect to a certain parameter can be calculated by using these cached factors [6, p. 101-102]. This immensely speed ups the calculation of these gradients and in turn the updating of parameters.

Important to note is that backpropagation requires that activation functions σ are differentiable. Also, taking derivatives can be expensive for certain functions. That is why the tangent hyperbolic and the logistic function are widely used, as these functions have derivatives that can be expressed in terms of their function value, making it enough to store $\mathbf{o}_i$ and omit net_i [6, p. 101]:

$$\sigma_1(x) = \tanh x \Rightarrow \dot{\sigma}_1(x) = 1 - \sigma_1(x) \tag{2.42}$$

$$\sigma_2(x) = \frac{1}{1+e^{-x}} \Rightarrow \dot{\sigma}_2(x) = \sigma_2(x)(1-\sigma_2(x)) \tag{2.43}$$

2.4.5. Physics Informed Neural Networks

Physics informed neural networks are a new type of neural networks introduced by Raissi et al. [1]. This new technique can be used to solve PDEs of physical phenomena whilst respecting the laws of physics encoded in these PDEs. This technique uses MLPs at its heart, as MLPs are efficient universal function approximators.

To illustrate this technique, consider the following abstract PDE with boundary conditions as an example:

$$\mathscr{L}(u)(x) = f(u(x), x) \quad x \in [0, L] \tag{2.44}$$

$$u(0) = A \tag{2.45}$$

$$u(L) = B \tag{2.46}$$

In this example $\mathscr{L}$ is some operator that takes derivatives of u. Raissi then uses an MLP $\mathscr{N} : [0, L] \to \mathbb{R}$ in the place of u to obtain:

$$\mathscr{L}(\mathscr{N})(x) - f(\mathscr{N}(x), x) = 0 \quad x \in [0, L] \tag{2.47}$$

$$\mathscr{N}(0) = A \tag{2.48}$$

$$\mathscr{N}(L) = B \tag{2.49}$$

The aim is then to train the MLP $\mathscr{N}$ such that it mimics the solution u of this PDE. To do this, $g := \mathscr{L}(\mathscr{N})(x) - f(\mathscr{N}(x), x)$ is first defined. Then the following MSE is used for the optimization:

$$\text{MSE} = \text{MSE}_u + \text{MSE}_g \tag{2.50}$$

where,

$$\text{MSE}_u = \frac{1}{2}\left(|\mathscr{N}(0)| - A|^2 + |\mathscr{N}(L) - B|^2\right) \tag{2.51}$$

$$\text{MSE}_g = \frac{1}{N}\sum_{i=1}^{N} |g(x_i)|^2 \tag{2.52}$$

Effectively, there are two MSEs: one for the boundary conditions MSE_u that tries to let the solution satisfy the boundary condition and one for the interior MSE_g that tries to let the solution satisfy the PDE itself. The latter uses sampled training points $x_i \in (0, 1)$ with $i = 1, \ldots, N$ to obtain an estimate of the error. When the total MSE is minimized by algorithms such as gradient descent with backpropagation, the MLP is effectively trained to become the solution u, as is the aim of Raissi et al. [1].

Important to note is that using PINNs to find the solution of a PDE is a perfect example of an unsupervised learning task, as only the inputs (sample points) are known and the error is determined only using the inputs and outputs of the network.

In figure 2.7 a schematic overview of the architecture of PINNs is shown. In this figure the MLP and PDE part are indicated alongside with the loss calculation. The paper of Raissi et al. [1] shows accurate results of applying this technique to a multitude of different PDEs, making it an interesting alternative to established alternatives such as the finite element method.

One may wonder how the gradients required for gradient descent are taken when other operations such as in the differential operator $\mathscr{L}$ and function f are used. Modern machine learning frameworks, such as PyTorch use an internal graph that records every operation applied to a variable [7, Documentation]. Backpropagation is then used to traverse the graph and calculate the gradients. This is exactly the same as the example posed in the subsection on gradient descent, but it is now extended to generic operations.

Inspired by this, one can extend physics informed neural networks to other types of problems. In the case of this thesis, it will be extended to Fraunhofer diffraction simulations. In figure 2.7, this would mean that the PDE part will be replaced with a simulation and that the input and output of the MLP are parameters of this simulation. The loss will then be based on what intensity profile is desired.

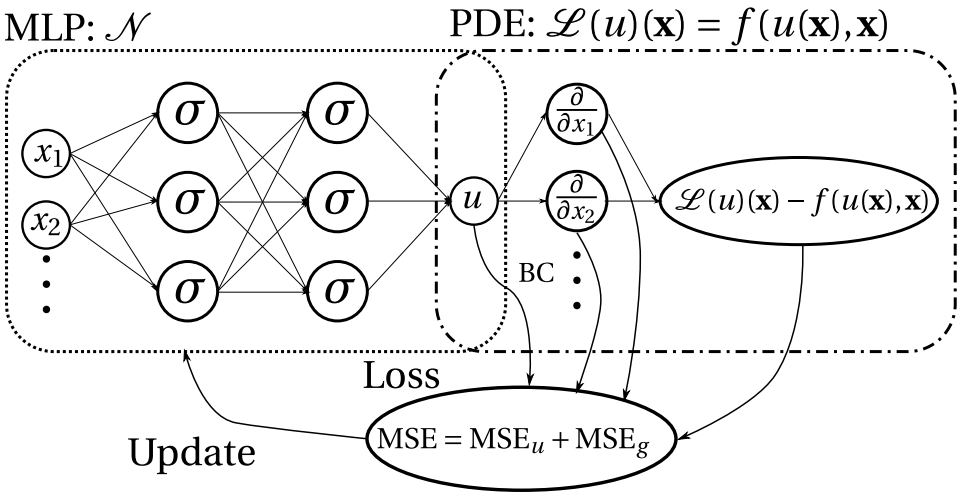


Figure 2.7: Schematic overview of a PINN, with the MLP and PDE part of the network. The same notation is used as in the subsection on PINNs. The three dots represent that there can be arbitrary inputs x_i or arbitrary derivatives. In addition, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$.

3

Experiment

3.1. Introduction

In this chapter the details of the experimental method and the experiments that were carried out will be discussed. To recapitulate, the general aim of the experiments is to find the phase distribution of a freeform lens in terms of B-spline surfaces such that a certain incoming field configuration results in a desired intensity pattern in the far field.

In the setup section the experimental method and required infrastructure will be described. After this description, the experiments section describes what experiments are done and what to expect in the results chapter.

3.2. Setup

The setup used for experiments is mainly composed of two components: An optical simulation and a Neural Network. The optical simulation uses Fraunhofer diffraction to simulate the light propagation. The Neural Network will be an MLP. The parameters used in the optical setup, such as the control net of the phase distribution and other parameters determining the initial field, will be provided by the MLP.

3.2.1. Optical Simulation

The optical simulation consists of two parts: The first part constructs the scalar field $U_0(x', y')$ and applies the phase distribution. The second part propagates the incoming field to the far field using Fraunhofer diffraction.

In the first part, where construction of $U_0(x', y')$ occurs, the field is created by using a superposition of plane waves and point sources placed at a specific distance to an SLM, the configuration of which is depending on the specific experiment. The SLM is the physical device that is capable of adjusting the phase of incoming light. The specific SLM (Pluto 2 VIS-096), on which this simulation is based, has a resolution of 1920×1080 (full HD) with pixel size $8.0\,\mu$m and maximum phase shift of 2.5π to 4π depending on the wavelength, as specified by manufacturer Holoeye Photonics AG [8]. In essence, the SLM alters the incoming field $U_0(x', y')$ in the following way:

$$U_0'(x', y') = U_0(x', y')\, e^{i\phi(x', y')} \tag{3.1}$$

where $\phi(x', y')$ is the phase adjustment carried out by the SLM. Since the SLM has a finite resolution, the phase is actually a discrete function that is sampled from some continuous function:

$$\phi[n, m] = \phi\big(n \cdot (8.0\,\mu\text{m}), m \cdot (8.0\,\mu\text{m})\big) \tag{3.2}$$

where the n, m are the zero-based indices, meaning $n, m \in \mathbb{N} \cup \{0\}$, in the x- and y-direction, respectively. And the square brackets signify that it is a discrete function. Important to note is that the SLM itself is rectangular and does not extend to infinity. Therefore, the rectangular shape of the SLM acts as an aperture function on the incoming field, meaning that everything outside its rectangle will vanish.

As described earlier, the continuous phase distribution will be based on a B-spline surface that is created using a 2D control net of control points. To illustrate this, suppose that $\{B_{i,j}\}$ with $i = 1,2,\ldots,n$ and $j = 1,2,\ldots,m$ is such a control net. Note that $\{B_{i,j}\} \in \mathbb{R}$ is not in bold face, as the phase distribution surface only has one value per x- and y-coordinate. In all experiments we use the same polynomial order p along both indices. The knot vectors Ξ and $\mathscr{H}$ along both indices will partition the unit interval $[0,1] \subseteq \mathbb{R}$ uniformly, meaning that each part in the partition has equal length. The resulting B-spline surface $S(\xi,\eta)$ will represent the phase distribution the SLM will have. To do so, the B-spline surface is used in the following way:

$$\phi[n,m] = S\Big(\frac{n}{1920}, \frac{m}{1080}\Big) \tag{3.3}$$

where n, m are again zero-based indices.

After the SLM has phase adjusted the incoming field, the second part commences. In this part the resulting field is propagated to the far field using the Fraunhofer diffraction integral, as seen in equation 2.15 in section 2.2.4. For Fraunhofer diffraction to work properly, the criterion of equation 2.16 must be satisfied. In simulations this criterion is easily satisfied by specifying a large enough propagation distance and using a small aperture. In practice this is difficult, as large propagation distances become infeasible in a laboratory setting. Therefore, as mentioned in section 2.2.6, one can use a classical convex lens to obtain a Fourier transform of the field without letting the field propagate to the far field, effectively executing Fraunhofer diffraction. In this thesis simulations will be carried out with a propagation distance of 20 cm for Fraunhofer diffraction. Physically, this is too short of a distance to satisfy the criterion, so this will correspond to using a lens of focal length $f = 20\,\text{cm}$. In the results chapter 4 'far field' will refer to the result obtained in the focal plane of the lens used to execute Fraunhofer diffraction.

The aforementioned parts in the optical setup are also depicted in figure 3.1, where U_0 is the initially constructed field, $\phi(x',y')$ is the phase distribution used by the SLM. After the SLM a positive lens with focal length f is placed, which results in Fraunhofer diffraction with a propagation distance f.

Since the Fraunhofer diffraction is executed numerically, it will be discretised. As described in section 2.2.5, this is done by using the discrete Fourier transform instead of the continuous Fourier transform. Specifically, the Fast Fourier Transform, or FFT for short, of the PyTorch library is used, as this library implemented the algorithm such that it preserves gradients, allowing backpropagation to update the weights and biases in the MLP that led to the parameters given to the optical simulation. In this thesis the FFT will be used with an aspect ratio of 1 : 1 and resolutions that are powers of 2. Since the SLM already has a resolution of 1920 on its longest side, a resolution of 2048 must already be used in the FFT. To increase the resolution of features in the far field, the size of the incoming field that is to be Fraunhofer diffracted must be increased. This can be done in two ways: One way is by adding padding around the SLM, and thus increasing the size of the incoming field, but at the cost of adding more resolution to the FFT. The other way is by downsampling the SLM but keeping the resolution the same. In other words, each pixel in the near field will cover a larger area. For instance, when downsampling with a factor of 2, a pixel in the near field will correspond with four pixels on the SLM. The specific approach used to increase the resolution of features is dependent on the specific experiment and this will be elaborated upon in section 3.2.4.

3.2.2. Neural Network

Each experiment will have a neural network specific to its optical setup. The inputs and outputs of the neural network will be parameters of the optical simulation. For instance, a network can have 5×5 control points as output for the B-spline surface phase distribution and as input the angle with which a plane wave is coming in. The neural network will be an MLP with the layer configuration dependent on the complexity needed for the experiment. The activation function will default to the tangent hyperbolic function. This function is applied to intermediate layer values, but not to the output. The Adam optimizer will be used during training to optimize the weigths.

To detect convergence of the loss and stop training the network, an early stopping criterion is used. This criterion works by the use of two parameters: A delta parameter and a patience parameter. The delta parameter is used to classify whether an update is an improvement. Specifically, an update is an improvement if the loss has gone down by at least delta. If the delta parameter is zero, then every arbitrary decrease in loss is considered an improvement. The patience parameter determines how many stagnating updates may occur before the training is stopped. To illustrate this, suppose delta is 0 and the patience parameter is 10. Then, if during training the loss does not decrease for consecutive 10 training steps, the training is halted.

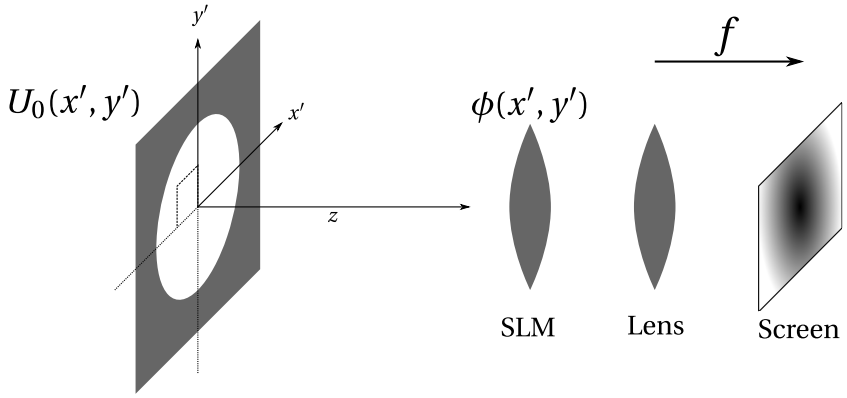


Figure 3.1: General setup used in experiments. The left coordinate system describes the incoming field $U_0(x', y')$, this field is phase altered by the SLM at $z = 0$ with function $\phi(x', y')$, then with the aid of of a positive lens with focal distance f, the field is Fourier transformed. Hence, the field is 'Fraunhofer diffracted' to the far field at a distance f.

The complete setup of both the optical simulation and the neural network is depicted in figure 3.2. The MLP receives an input, it computes an output, then both inputs and outputs are fed into the optical simulation and serve as parameters. The optical simulation will produce an intensity distribution based on the parameters it was fed. This intensity distribution is compared to the desired intensity distribution, which gives a loss value. This loss value is then used to adjust the weights and biases in the MLP. Important to note is that the optical simulation will only do operations on its parameters that are gradient preserving (this means that the simulation only contains operations that are differentiable and therefore the gradient with respect to a certain trainable parameter can be found after simulation), as otherwise no training would have been possible. This is also exactly what physics informed neural networks are based on.

3.2.3. Loss functions

The loss as a result of comparing the predicted intensity profile and the desired intensity profile can be calculated in a multitude of ways. In this thesis a variation of the mean squared error between the squared moduli of the predicted field $\left|\hat{U}_p\right|^2$ and desired field $\left|U_p\right|^2$ will mainly be used:

$$E = \frac{1}{N^2} \sum_{n=0}^{N} \sum_{m=0}^{N} \left| \frac{\left|\hat{U}_p[n,m]\right|^2}{\max_{\{i,j\}} \left|\hat{U}_p[i,j]\right|^2} - \frac{\left|U_p[n,m]\right|^2}{\max_{\{i,j\}} \left|U_p[i,j]\right|^2} \right| \tag{3.4}$$

where $N \times N$ is the resolution of the Fraunhofer diffraction. Other loss functions can be used as well. Specifically, in the previous equation $\left|\hat{U}_p\right|$ and $\left|U_p\right|$ can be used instead of the intensities. The motivation for such a loss function is that it will prioritize smaller differences more 'equally', as larger differences are not blown up by squaring the field. This gives the following loss function:

$$E = \frac{1}{N^2} \sum_{n=0}^{N} \sum_{m=0}^{N} \left| \frac{\left|\hat{U}_p[n,m]\right|}{\max_{\{i,j\}} \left|\hat{U}_p[i,j]\right|} - \frac{\left|U_p[n,m]\right|}{\max_{\{i,j\}} \left|U_p[i,j]\right|} \right| \tag{3.5}$$

Another possible loss function can be based on the fact that the perception of light in terms of brightness is logarithmic in nature for human vision, as described by Kenneth and Fellers [9]. With this in mind a loss function with a logarithm of the difference between two intensity distributions or the absolute value of the field can be constructed. This idea yields the following two loss functions, with the intensity of the field first and the modulus of the field second:

$$E = \frac{1}{N^2} \sum_{n=0}^{N} \sum_{m=0}^{N} \log\left(\left| \frac{\left|\hat{U}_p[n,m]\right|^2}{\max_{\{i,j\}} \left|\hat{U}_p[i,j]\right|^2} - \frac{\left|U_p[n,m]\right|^2}{\max_{\{i,j\}} \left|U_p[i,j]\right|^2} \right| + \epsilon \right) \tag{3.6}$$

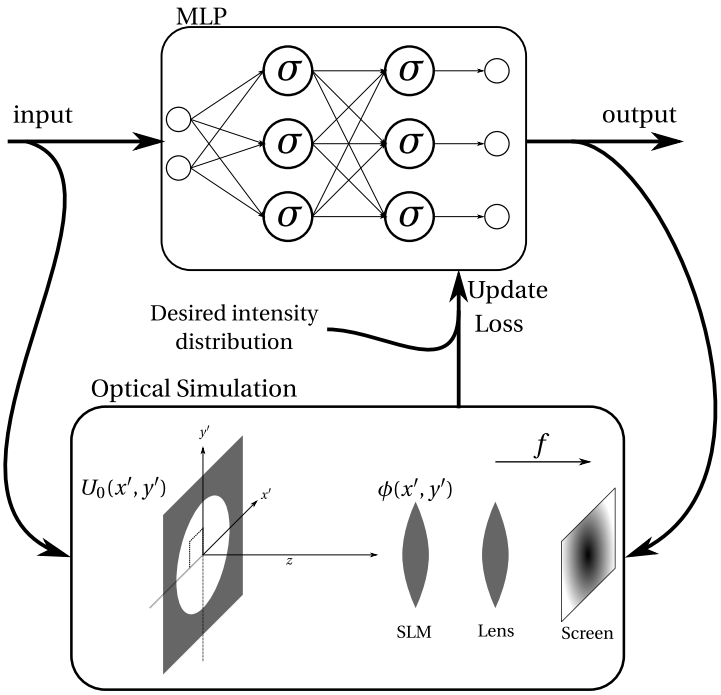


Figure 3.2: The adopted setup of the MLP and the optical simulation. In this figure the parameters used by the optical simulation are determined by the inputs and outputs of the neural network.

$$E = \frac{1}{N^2} \sum_{n=0}^{N} \sum_{m=0}^{N} \log\left(\left| \frac{\left|\hat{U}_p[n,m]\right|}{\max_{\{i,j\}} \left|\hat{U}_p[i,j]\right|} - \frac{\left|U_p[n,m]\right|}{\max_{\{i,j\}} \left|U_p[i,j]\right|} \right| + \epsilon \right) \tag{3.7}$$

where $\epsilon = 10^{-15}$ is added to avoid reaching $-\infty$ when using floating point numbers that have limited precision. Obtaining $-\infty$ will result in 'Not a Number' values for the weights and biases when training continues with such a value for the loss.

To compare these four loss functions, we will run them on the same optical setup with the same initialization of the weights and biases in the MLP. Since comparing them requires some sort of objective metric, we will use the first loss function for that. This is not ideal, but this loss function seems to make the least assumptions on the input, whereas the exponential based loss functions do. In the experiments section it will be described what experiment will be used to compare them. To make distinguishing them easier, consider the following table of abbreviations:

Table 3.1: Naming convention for the different loss functions.

Abbreviation	**Equation number**
Intensity MSE	3.4
MSE	3.5
Logarithmic intensity MSE	3.6
Logarithmic MSE	3.7

3.2.4. Progressively increasing resolution

Since working at large resolution requires more computational power, it is preferable to work on resolutions as low as possible for the least amount of iterations. One way that may achieve this, is to start of with a lower resolution and once convergence has been reached there, start increasing the resolution of the simulation. After each increase in resolution training commences again and is done until convergence is reached. The convergence is defined here as triggering the early stopping criterion as described in section 3.2.2.

As described in section 3.2.1, there are two ways to achieve more detail in the resulting Fraunhofer diffraction.

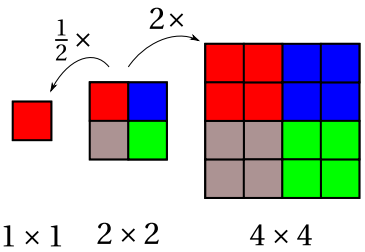


Figure 3.3: Sampling of the SLM for lower and higher resolution. The specific choice of what pixel is preserved or duplicated is in accordance with the 'nearest' sampling specification of the [10] library.

One way is to add more padding around the SLM and the other way is to downsample the SLM. Both methods result in a larger margin to SLM size ratio. Since the SLM works at a fixed resolution of 1920 × 1080, we must either sample the phase distribution down or up. When downsampling it is important to make sure that no averaging occurs, as to stay as close as possible to the original phase distribution. When upsampling, no interpolation can occur, as the SLM cannot work at higher resolutions. Hence, 'nearest' interpolation is used. That is, when sampling up or down, replicate the closest available value. To illustrate this, consider figure 3.3. In this figure, the center square that is divided in four parts depicts the SLM at original resolution. When downsampling, only the red upper left pixel is preserved and when upsampling, the pixels are repeated. This method ensures that the result of simulations will be applicable to a physical SLM.

In practice the following procedure will be used for increasing the resolution in the far field of the problem: The resolution will be doubled in either direction, so $(N \cdot 2) \times (N \cdot 2)$. Then, the physical size of the near field will be doubled. So the near field that is sampled and then diffracted will have new size: $(L_x \cdot 2) \times (L_y \cdot 2)$. To understand this procedure, consider section 2.2.5. There it is stated that a feature of size $d = \frac{\Delta x}{\lambda z}$ in the continuous Fourier transform has index range $\Delta k = \frac{\Delta x}{\lambda z} \frac{L_x}{2\pi}$. So doubling the resolution L_x, and homologously doubling L_y, will result in a larger index range in either direction and thus more detail. However, this will result in a smaller area that is seen in the far field, as the largest x-value is determined by $x_{\max} = \lambda z \frac{2\pi}{L_x}(N/2-1)$, which gives that $x_{\max}$ will halve in size. To counteract this, N (simulation resolution) is doubled in either direction. This procedure makes sure that the spatial scale of the far field stays approximately the same.

3.2.5. Infrastructure

To join the optical simulation and the neural network together, infrastructure that does the plumbing between the two is necessary. In this thesis a modular approach is taken, which consists of a 'Scenario' that is essentially the optical simulation and a 'Model' that is the neural network and instructs the Scenario what parameters and resolution to use. Both of these components are configured in a configuration file. This configuration file uses Javascript Object Notation, or JSON for short, as its syntax. Each experiment has a specific configuration file. In this file the model and scenario to use is specified, along with model and scenario specific configuration options. These options include the specification of the desired intensity distribution, here referred to as a profile. This intensity profile can be anything from an image to a solution known to be in the solution space. Additionally, in the configuration file it is specified what 'topology' is used for the experiment. This topology describes how parts of the input and output of the neural network are related to parameters in the optical simulation that is carried out in a scenario. For an example configuration file, see appendix section A.2.

To visualize this infrastructure, consider figure 3.4. In this figure each modular component is shown in a flowchart. The result of running an experiment will be a set of optimal parameters that can be used in the optical simulation to achieve a result as close as possible to the desired intensity profile.

3.3. Experiments

As described in the introduction, the general aim will be to improve upon the work of Imhof [2] by extending the problem to 2D and using a neural network that is specifically trained for a certain optical setup. To check the feasibility and correctness of the proposed setup we will verify that a solution that is certainly in the solution space can be found. To obtain such a solution in the solution space, we will use the optical setup of a specific experiment and feed it randomized parameters. The neural network is then tasked with finding these

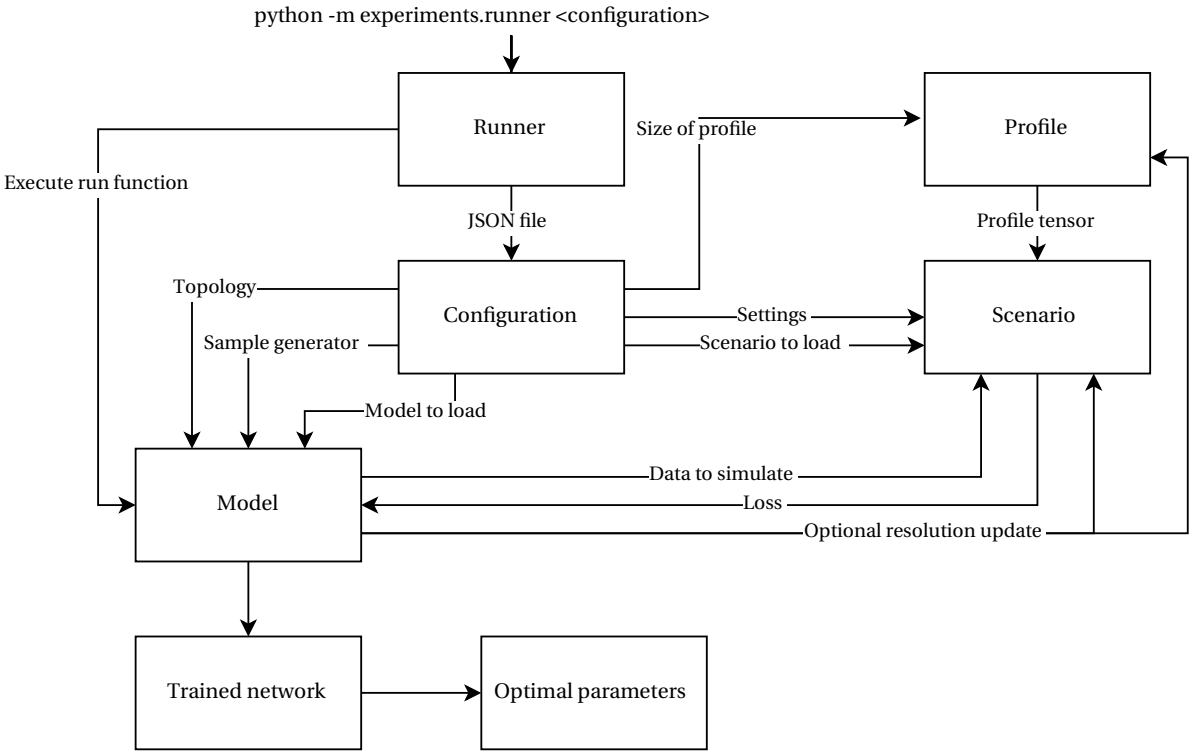


Figure 3.4: Modular infrastructure used to carry out all experiments described in the experiment section.

parameters. This will be done for each experiment. After we have done this for a few different setups, we will try to see if the setup is able to recreate advanced images that are not necessarily in the solution space.

3.3.1. Single perpendicular incident plane wave: solution space

In this experiment a single perpendicular incident plane wave to the SLM will be used. The SLM will have a phase distribution that is created by an order 3 B-spline with 5×5 control points. The desired intensity profile will be generated by selecting 25 control points at random and running the simulation to obtain a profile. Since this optical simulation only has control points as its input, the neural network that is trained on this will only have 25 outputs and no inputs. The input will simply be a constant 1 value. In essence, the neural network is then merely an advanced optimization algorithm and is no longer like a 'physics informed' neural network, as no inputs are varied and the output of the network is expected to converge to a constant solution. Nevertheless, if this approach works, it proves that the simulation infrastructure is written in such a way that gradients are indeed preserved and backpropagation can work on it. In addition, this experiment will show if optimizing the network for a specific problem is a feasible strategy.

In section 3.2.3 multiple loss functions have been described. In this experiment these loss functions will be tested to see their performance. As described in that section, it is difficult to compare them, as for the same desired profile they can all yield different solutions. Therefore, the Intensity MSE loss function will be used as a baseline comparison between the different solutions. When comparing them it is important to make sure that the network and random control points are initialized in the same way. This can be achieved by setting a so-called 'random seed'. Since it can occur that a specific loss function is better suited to a certain initialization, the experiment with all four loss functions will be carried out 5 times with a different seed.

In addition, in section 3.2.4 a procedure for increasing the resolution of the simulation without changing the physical size of the far field domain was introduced. During training it can be advantageous to start off with a small resolution and increase the resolution when convergence is reached, as doing training on a smaller resolution is much faster. This experiment will be used to test this.

3.3.2. Single point source: solution space

This experiment will make use of a single point source placed at a distance d from the SLM. The SLM will have a phase distribution that is created by an order 3 B-spline with 5×5 control points. The topology of the neural network will be: distance d as a single input and $5 \times 5 = 25$ control points as output. For some random distance d and 25 random control points a simulation will be ran. The output of this simulation will become the desired intensity profile. The neural network is then tasked with finding the parameters to achieve this. If the network is able to find these parameters, this experiment will show that the network is able to find and converge towards a solution that is known to be in the solution space.

3.3.3. Single variable angle incident plane wave: solution space

In this experiment a plane wave with its wavefront parallel to the y'-axis is tilted by some angle α to the optical axis, the z-axis. This plane wave is incident to the SLM. Just as before, the SLM will have a phase distribution created by an order 3 B-spline with 5×5 control points. The network will then have the following topology: the angle α as an input and the 25 control points as an output. The desired intensity profile will be the resulting profile of an optical simulation with random angle and control points. An important limitation is that the angle α will be constrained to the range $-30°$ till $30°$, as to ensure the setup can be replicated in a physical experiment and to ensure the paraxial approximation is satisfied. The paraxial approximation is what is used in the derivation of the Fraunhofer diffraction and results in the assumption that $r \approxeq z$ in section 2.2.4. Hence, we must not violate this assumption by allowing greater angles. The network will be trained with random input angles in the aforementioned range.

3.3.4. Single perpendicular incident plane wave: recreating the university logo

Here the same optical setup will be used as in the experiment described in section 3.3.1. That is, there is a single perpendicular incident plane wave to the SLM. The phase distribution of the SLM is created with an order 3 B-spline surface. The desired profile will be based on the logo of Delft University of Technology, specifically the truncated logo that consists of three shapes. For reference, the logo is shown in figure 3.5. The number of control points is yet to be determined, depending on the amount of complexity needed to reconstruct the logo. In this experiment the network topology will consist of no inputs (only a 1) and $n \times n$ control points, where n is yet to be determined.

Figure 3.5: The to be replicated university logo.

3.3.5. Two variable angle and amplitude incident plane waves: solution space

In this experiment two incoherent plane waves will be used. Since these plane waves are incoherent, these plane waves will not interfere with each other. Therefore, to simulate this, the Fraunhofer diffraction will be calculated for each plane wave separately after which the resulting intensities are added together. Each plane wave will have a variable amplitude and variable incidence angles. In figure 3.6 a wavefront of the planewave, indicated in gray, is shown. The plane wave can be rotated along the x'- or y'-axis with angles α and β, respectively. So each plane wave has three adjustable parameters, two angles and one amplitude. For this experiment the SLM will have a phase distribution created by an order 3 B-spline surface with 5×5 control points. Hence, the topology of the neural network becomes: 6 inputs and 25 outputs. Just as in the experiment described in section 3.3.3, where only a single plane wave had one variable angle, the angles will be restricted to the range $-30°$ till $30°$.

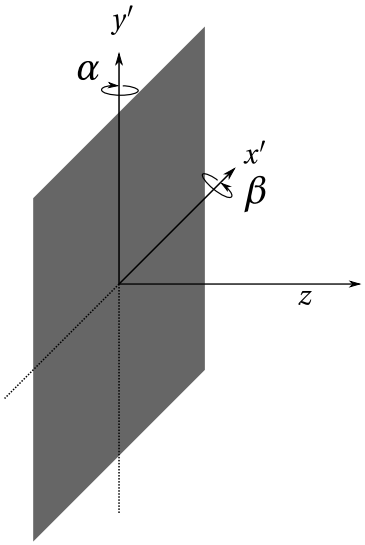


Figure 3.6: Rotating a plane wave along the x'-axis with angle α and along the y'-axis with angle β. The grey area depicts a wave front of the plane wave.

4

Results

4.1. Introduction

In this section the results of the experiments described in section 3.3 will be presented. Experiment specific discussion will be described after the results of that experiment have been presented. Overall discussion of experiments will be described in chapter 5, which is the next chapter.

The program and its configurations are available in a git repository. See section A.3 in the appendix for the link.

4.2. Single perpendicular incident plane wave: solution space

This experiment consists of three parts. The first part verifies whether the simulation infrastructure is working correctly. Secondly, with this in mind the different loss function are benchmarked and a best loss function is picked for use in the rest of the experiments. Lastly, with the chosen loss function, the results of an experiment aiming to find a phase distribution known to be to in the solution space, are presented.

4.2.1. Qualitative verification of simulation infrastructure

To start off, we first assess qualitatively whether the simulation gives the expected result. The simplest case is to to set the control points in the B-spline surface all to zero, i.e. not altering the phase of the incoming field of the planewave. Then the SLM will act as a rectangular aperture. For a perfectly square aperture the Fraunhofer diffraction simulation result should look like the pattern in figure 4.1. Not that in this figure Voelz boosted the image contrast to better show the side lobes of the pattern. In addition, he works on a different length scale than we do here. Since the SLM is not a square but a rectangle with aspect ratio $16:9$ with its long side along the horizontal axis, we expect that in the horizontal direction the pattern is squished and it is stretched in the vertical direction. And indeed this is what we observe in figure 4.2, where the result of the simulation has been plotted. The plot uses a sequential color map, where the brightness indicates the value for the intensity, that is $|U|^2$ is plotted. This simulation used a resolution of 1024×1024 pixels with the resolution of the SLM scaled to 38×21, giving a near field size of $41\,\text{cm} \times 41\,\text{cm}$ and far field size of $0.03\,\text{cm} \times 0.03\,\text{cm}$ when a propagation distance of $20\,\text{cm}$ is used. In addition, this simulation used a wavelength of $633\,\text{nm}$. The far field size may seem small, but keep in mind that the SLM has dimensions $1.536\,\text{cm} \times 0.864\,\text{cm}$.

From this comparison we can conclude that the simulation is working and correctly implements the Fraunhofer diffraction integral in numerical form. Therefore, we continue with comparing the different loss functions.

4.2.2. Loss function comparison

The results of the loss function comparisons are shown in table 4.1, where the abbreviations of table 3.1 in the experiments chapter have been used. The experiments have been run with the same settings as mentioned in the first paragraph, which is paragraph 4.2.1. The results in the table are the averages of running the experiment with five different seeds. The seed determines the control points used for the B-spline surface phase

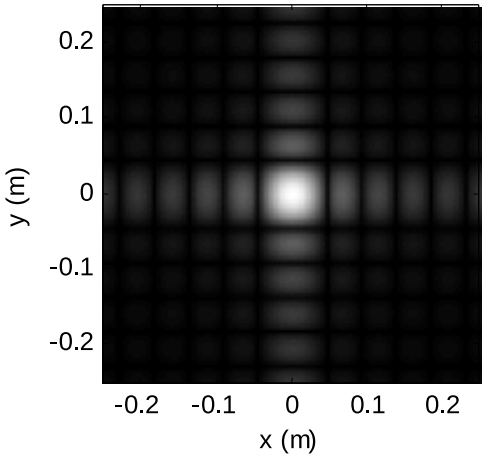
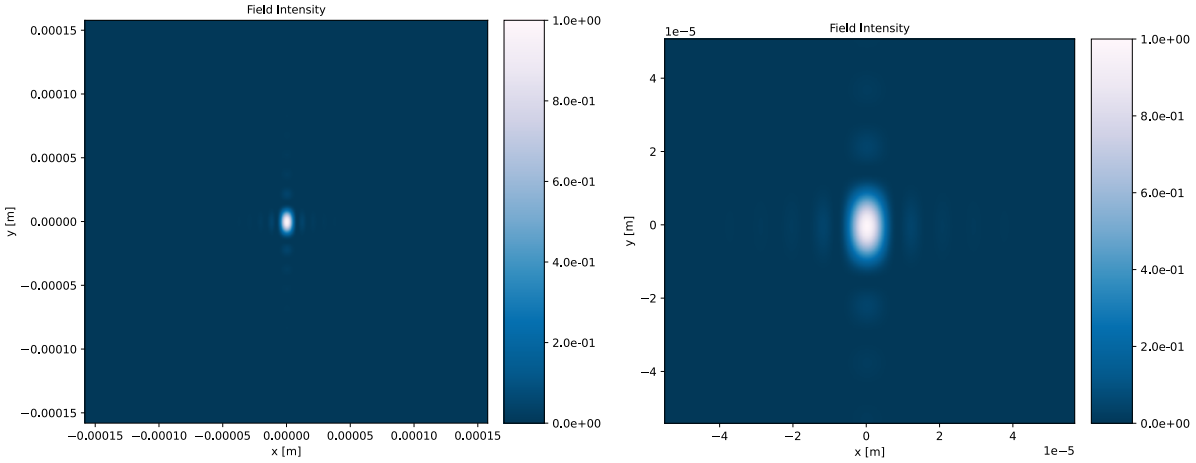


Figure 4.1: Fraunhofer diffraction pattern (contrast is boosted) as a result of a rectangular aperture. This figure has been extracted from Voelz [4, p. 81, figure 5.7a]. It does not come with a color scale, but it does indicate the overall pattern.



(a) Original normalized intensity distribution.

(b) Zoomed in normalized intensity distribution.

Figure 4.2: Intensity distribution created using a B-spline phase distribution of 5×5 zero-valued control points.

distribution that will give the desired intensity profile. The seed also determines how the weights and biases in the MLP are initialized. Using a seed ensures that the results presented here are reproducible and that the comparison between different loss functions is fair. The MLP consists of 1 input and 25 outputs. It has two hidden layers of size 20. The tangent hyperbolic function is used as an activation function, except for the final output. To detect convergence, the early stopping criterion, as described in section 3.2.2, is used with a patience of 10 updates and no delta parameter. The delta parameter is not used, as each of the loss functions will work in widely different orders of magnitude, which makes setting a fair delta parameter infeasible. The loss value shown in the table is calculated with the Intensity MSE loss function. The measured runtime is the runtime of the whole script, not only the optimization time. Therefore, a small amount of the runtime will be constant time, which is independent of the chosen loss function. The experiments have been run on a high-performance computer using the PyTorch library. The PyTorch library was configured to use CUDA on a single NVIDIA GeForce RTX 2080 Ti GPU.

From the results in the table we can conclude that the loss functions without a logarithm perform best in terms of lowest loss value. This is to be expected when the baseline comparison is done using the Intensity MSE loss function. Interestingly, the loss functions based on a logarithm give faster convergence, in the sense of the early stopping criterion, on average than their non-logarithmic counterparts. Also worth noting is

the fact that the plain MSE loss function gives a better loss value than the Intensity MSE, while both final loss values have been calculated with the Intensity MSE loss function. This difference may be explained by the fact that the MSE loss function took much longer on average for the early stopping criterion to trigger, continuing with its optimization.

To further investigate these results we will consider the loss values of the respective loss functions over the epochs. For this, consider figure 4.3, where the runs with seed 1.0 have been plotted. In this figure all four loss functions with their values over the epochs have been plotted. Here, an epoch consists of 5 updates grouped together. The value plotted here is the average of 5 consecutive updates, as to smooth the loss curves. In addition, note the difference in the amount of epochs for each plot. This difference is the result of the early stopping criterion with a patience parameter of 10. Since these loss functions have widely different definitions, comparing their specific values has no use. Therefore, instead, we only consider the general shape.

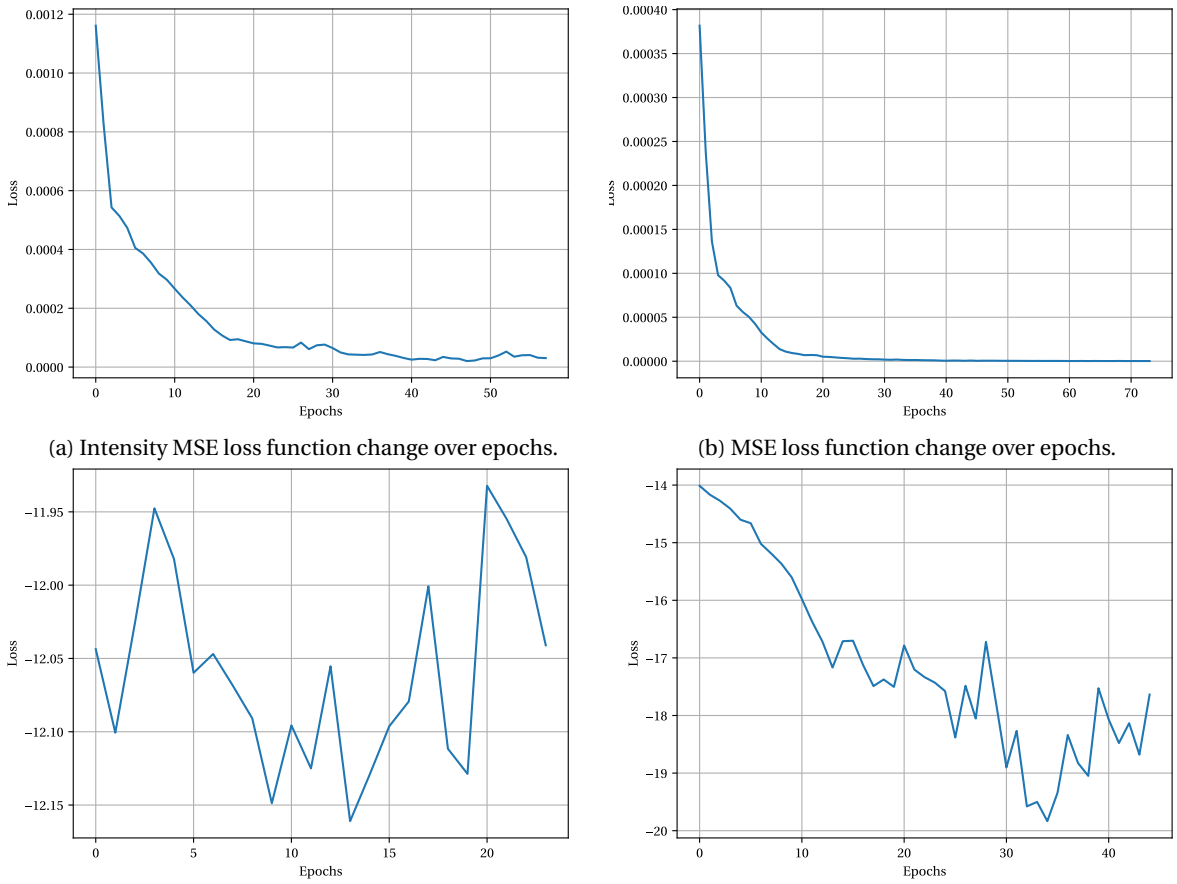
From the general shape, we can conclude that the non-logarithm based loss functions, in figures (a) and (b), show exponential like convergence. Important to note is that while the MSE loss function looks like it converges much faster, it may not be the case, as the figure shows more epochs on the x-axis. To compare their speed of convergence properly, we will compare them by using the amount of epochs it took to decrease the loss value by 90%. For the intensity MSE this took 16 epochs and for the MSE this took 10 epochs. From this comparison it seems that the MSE loss function converges faster. To further substantiate this claim, we will calculate an average speed of convergence for the 5 different seeds using the aforementioned criterion. This yields the following results: 16.2 (8.0) epochs on average for the MSE loss function and 32.6 (15.9) for the intensity MSE loss function, where the value between brackets is the sample standard deviation. For the results per seed, consider table A.2 in the appendix. Hence, the MSE loss function does indeed converge faster. Since this comparison may not be entirely fair when using different types of loss functions, we will also calculate the average convergence of the intensity MSE loss when training with the MSE loss function. This gives a value of 29.6 (9.2) epochs for convergence when using the MSE loss function to optimize, again with the sample standard deviation between brackets. The results used for this are in table A.2 in the appendix. Therefore, with a value of 32.6 (15.9) for the intensity MSE and a value of 29.6 (9.2) for the MSE, when using the same baseline intensity MSE loss function, we cannot conclude which one has faster convergence.

In figure 4.3 and then specifically figures (c) and (d), we see the result of the logarithm based loss functions. From these plots we can observe that their behaviour is much more oscillatory than for their non-logarithm counterparts. In fact, the logarithmic intensity MSE loss function does not seem to converge at all, while the logarithmic MSE loss function does seem to converge, but then starts oscillating. For the logarithmic intensity MSE it is useless to use the 90%-decrease criterion for comparison, as this is never reached. Instead, we will use the intensity MSE loss function to check the convergence when training with either of the logarithm based loss functions. In figure 4.4, the progression of the intensity MSE loss function values as a function of the number of epochs have been plotted. From the first plot in (a), we can conclude that no clear convergence occurs for the logarithmic intensity MSE loss function, as most of the curves stay relatively constant, albeit with oscillatory behaviour. For the logarithmic MSE loss function in figure (b), we also observe the relative constantness of the intensity MSE loss value, except for the red curve (seed 1.0). Nevertheless, we can conclude that on average no definitive convergence occurs.

Hence, from the previous results, we conclude that the logarithm based loss functions do not yield proper convergence. This leaves us with the intensity MSE and MSE loss functions. As previously found, these loss functions have similar convergence speeds. From table 4.1 we find that the MSE loss function takes on average longer, but gives a better loss in the end, while the intensity MSE loss function has it the other way around. Since the intensity MSE is the most useful loss value, i.e. it makes the most physical sense, we will use that loss function for the rest of the experiments. In addition, the better loss value of the MSE loss function can explained by it not triggering the early stopping criterion as early as the intensity MSE loss function. This can be corrected for the intensity MSE loss function by adjusting the patience and delta parameter of the early stopping criterion.

4.2.3. Solution space

As stated in the last paragraph of the previous subsection, the intensity MSE loss function will be used from now on. With the same settings as described in the first paragraph of this experiment, we will use a seed of 1.0 to generate the control points that are used to create the intensity profile. This intensity profile is certainly



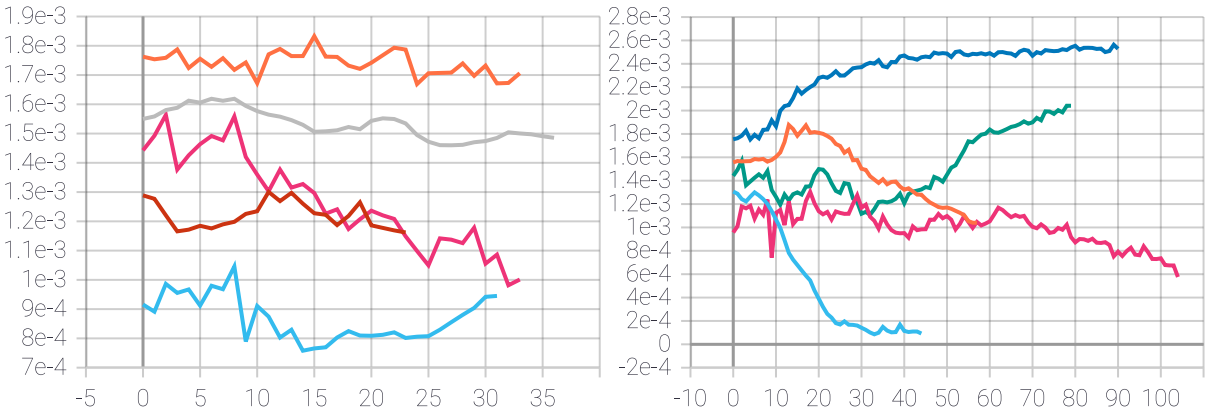
(a) Intensity MSE loss function change over epochs.

(b) MSE loss function change over epochs.

(c) Logarithmic intensity MSE loss function change over epochs.

(d) Logarithmic MSE loss function change over epochs.

Figure 4.3: Comparison of loss function convergence for all four loss functions, initialized with seed 1.0 and using an early stopping criterion with a patience of 10.



(a) Logarithmic intensity MSE loss function.

(b) Logarithmic MSE loss function.

Figure 4.4: These plots show how the intensity MSE loss values evolve over time when training with the logarithmic intensity MSE loss function an the logarithmic MSE loss function. On the horizontal axis the epoch index is shown and on the vertical axis the intensity MSE loss value is shown. Each line in the plot corresponds with the seeds 1.0, 2.0, 3.0, 4.0 and 5.0.

Table 4.1: Average results for loss (in terms of Intensity MSE) and runtime until convergence for different loss functions. The abbreviations used here are the same as in table 3.1. The average has been calculated over 5 runs with random seeds 1.0, 2.0, 3.0, 4.0 and 5.0.

Loss function	**Loss**	**Runtime**
Intensity MSE	$6.53 \cdot 10^{-5}$	31 s
MSE	$1.98 \cdot 10^{-5}$	56 s
Logarithmic intensity MSE	$1.40 \cdot 10^{-3}$	15 s
Logarithmic MSE	$8.69 \cdot 10^{-4}$	28 s

in the solution space. Then, the network is trained to find the control points that create this intensity profile. The result of this is shown in figure 4.5. The first figure uses a sequential color map and shows the intensities in the far field. The second figure uses a non-sequential color map and shows the moduli of the field in the far field. Both figures show the desired profile that has been calculated by using a random 5×5 control net of control points. Then the second plot in both figures is the result after training and the last plot in both figures shows the difference between the two. Important to note is that for comparison the fields have been normalized, i.e. their maximum is put at 1 and their minimum is put at 0. The final loss value for the intensity MSE is $3.46 \cdot 10^{-5}$ and it took 58 epochs of 5 updates for the early stopping criterion to trigger. From both figures in figure 4.5, we can see that the found control points result in an intensity profile pattern that is very much alike to the desired profile. In addition, we see that the greatest difference between the two patterns is around 0.012 in the center (in figure (a)), so around 1.2% of the maximum intensity.

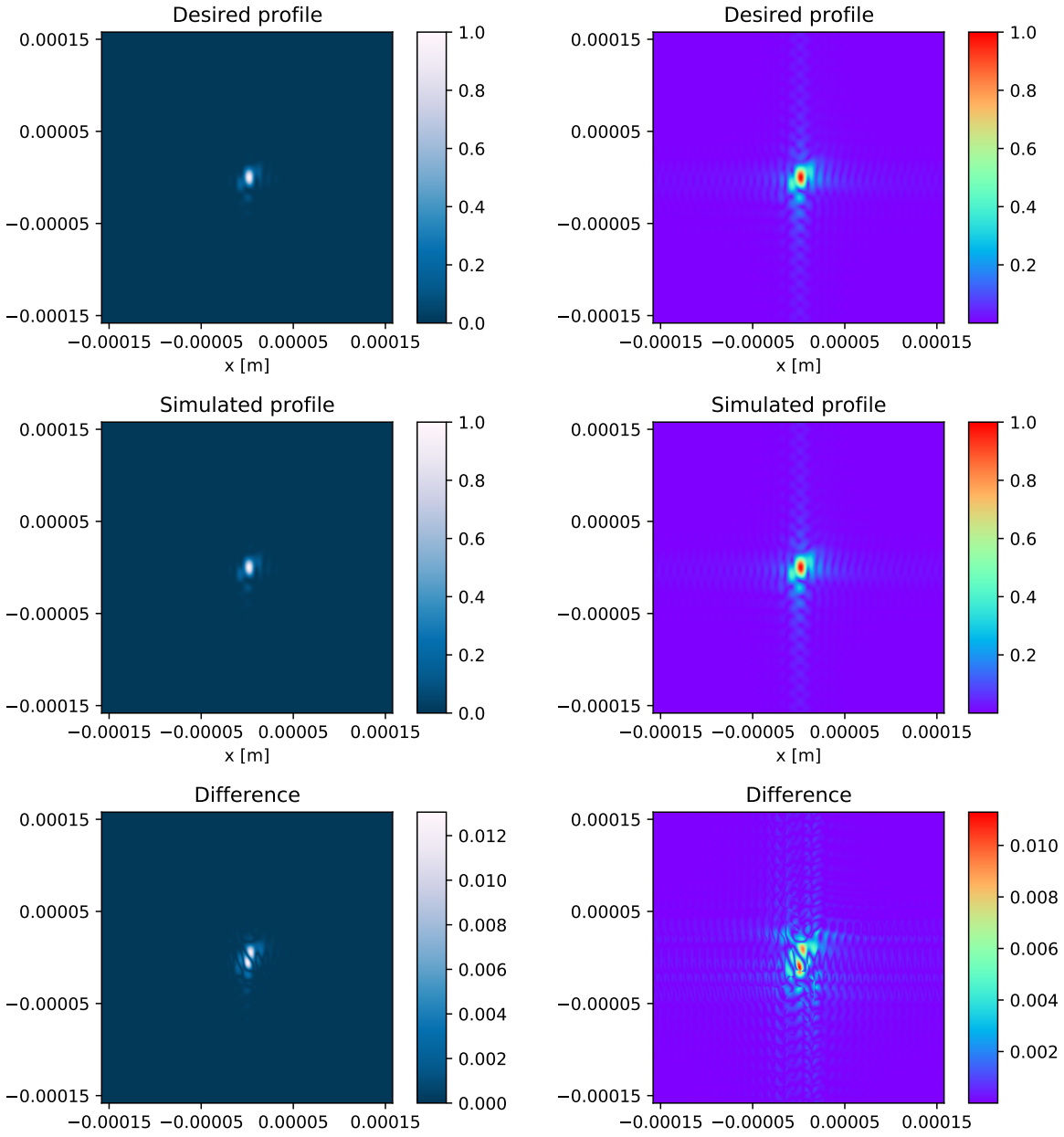
From these results we can conclude that backpropagation is working correctly, even across the simulation, as the training is able to modify the weights and biases such that the network outputs control points that give an intensity profile very close to the desired profile. In addition, we can conclude that the proposed setup is able to replicate solutions in its solution space.

To further investigate the solution that has been found, we will create an image of the B-spline surface phase distribution. This image will be modulo 2π with a cyclic color map, as the control points can lay outside the range $-\pi$ to π. On top of this image the control net will be drawn. This results in figure 4.6, where in subfigure 4.6a the desired phase distribution is drawn and in subfigure 4.6b the found phase distribution is drawn. When observing these plots, it is clear that, due to their equal color maps, that these two phase distributions are not some multiple of 2π apart. Otherwise, plotting both phase distributions modulo 2π would have resulted in the same image. In addition, we observe that where the image is kind of red in (a), we have a kind of blue color in (b) and vice versa. An important exception to this observation is the black color in (a), which signifies a π or $-\pi$ value, whereas in (b) these areas are white and signify a 0 value. This seems to suggests that the found phase distribution is the same as the desired phase distribution, but shifted by a factor π. To verify this, consider figure 4.6c, where the difference between the desired phase distribution and the found phase distribution is plotted and taken modulo 2π. And indeed, the found phase distribution is now almost exactly equal to the desired phase distribution, except for a global phase difference of $\pm\pi$. The extra phase difference does not lead to a different intensity profile in the far field, as a factor $e^{\pm i\pi}$ can be taken out of the integral expression of the Fraunhofer diffraction integral in equation 2.15 in the Theory chapter. From this result we can conclude that the proposed setup is able to replicate a solution known to be in the solution space of the problem.

4.2.4. Progressively increasing resolution

In section 3.2.4 a procedure for increasing the resolution without changing the physical size of the far field was introduced. In practice, we will use the following approach: we will start at a resolution of 256×256 and train until convergence, i.e. when the early stopping criterion with a patience parameter of 10 is triggered; then we will double the resolution to 512×512 and again train until convergence; and finally we will do the same for 1024×1024, which is the same resolution as used in the previous subsection.

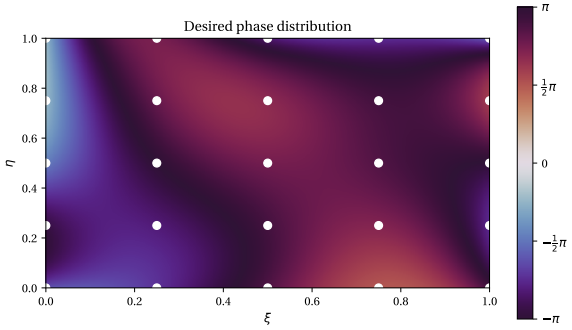
This method resulted in a loss value of $2.86 \cdot 10^{-5}$ in terms of the intensity MSE loss function. This is a better result than training at a single resolution, because we previously obtained a loss value of $3.46 \cdot 10^{-5}$. To make the comparison more fair, we will consider their running times. For the resolution update method training took 45 s with three updates and three times convergence. For the static resolution training took 27 s, which is significantly shorter. However, the static resolution took 58 epochs and the resolution update method took 45



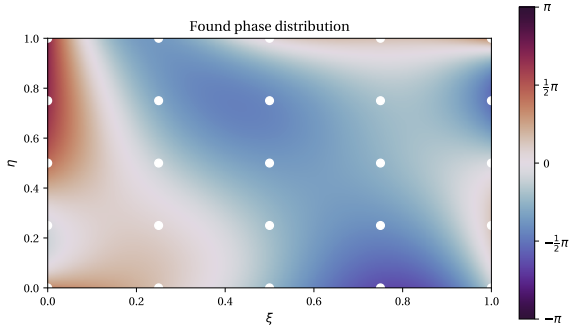
(a) Using a sequential color map for the intensities and their differences.

(b) Using a non-sequential color map for the moduli of the fields and their differences. This gives more contrast.

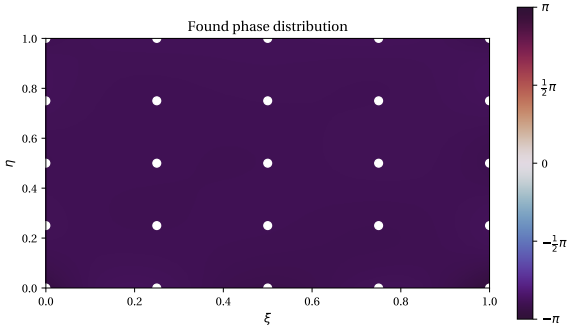
Figure 4.5: Results after training for 58 epochs with initialization done using a seed 1.0 and using the intensity MSE loss function.



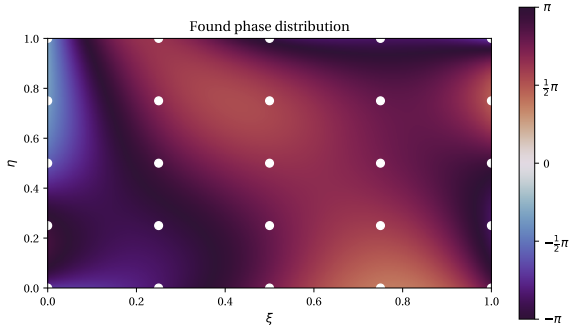
(a) The desired phase distribution.



(b) The found phase distribution.



(c) Difference between the plots in figures 4.6a and 4.6b taken modulo 2π.



(d) Image of the found phase distribution using the resolution update method.

Figure 4.6: Images of the desired and found phase distribution with and without the resolution update method. In these plots the white dots represent the location of control points. The ξ-axis corresponds with the long side of the SLM and η-axis corresponds with the short axis of the SLM.

epochs at 1024×1024 resolution. So the resolution update method takes a shorter time at full resolution, but it still took longer because of the convergence at the other resolutions. This runtime comparison used the GPU for training, where working at larger resolutions for the Fourier transform does not result in much longer execution, as most of the calculations can be done in parallel. Therefore, when using a CPU as compute device, which has less parallelization options, it may prove useful to use this resolution update method, as it can shorten runtime.

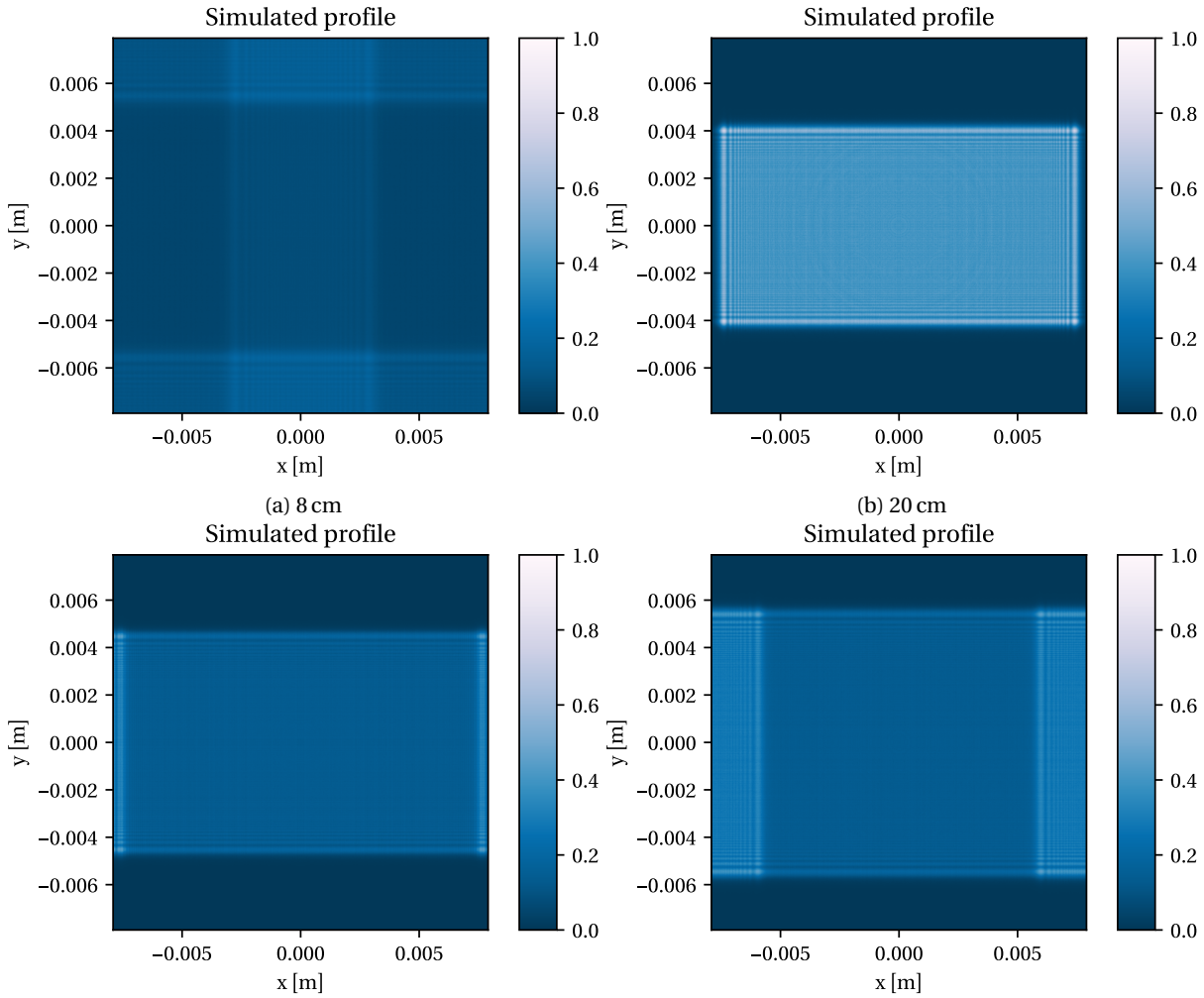
To test this hypothesis, we will run both setups on the CPU. This results in a runtime of 160 s for the static resolution and a runtime of 149 s for the resolution update method. Note that running on a different compute device results in a different internal random state on which the seed is imposed. Therefore, these running times cannot be compared to the GPU running times, as the networks and desired profiles are differently initialized. These results confirm the hypothesis that on a CPU it may be beneficial to use this method of updating resolution. However, more runs need to be carried out to make this claim more substantiated. Since this is a bit outside the scope of this thesis, we will carry on with investigating the found phase distribution.

In figure 4.6d the phase distribution as found by the resolution update method on the GPU is shown. From this figure we immediately observe that it looks very similar to figure 4.6a, where the desired phase distribution is shown. This is in contrast to the first phase distribution we found, see figure 4.6b, which was shifted by approximately π in phase, as illustrated in figure 4.6c.

4.3. Single point source: solution space

Before we start training a network to find a solution, we first verify that the simulation implementation for a point source is working well. To do so, we set all control points to zero and simulate merely the light of a point source going through a rectangular aperture, the SLM in this case. For this setup, we put a point source at 8 cm from the SLM, which is a typical distance. This simulation will start with the SLM unscaled, so the SLM has its original resolution of 1920×1080. Then the simulation will have a resolution of 2048×2048, as to fit the

SLM. This gives the result of figure 4.7a. In this figure we see horizontal and vertical bands occurring. With a point source and the SLM as aperture we would expect the center to have the greatest intensity, instead we have the greatest intensity where the horizontal and vertical bands cross. At first glance it may look like the algorithm does some bad Fourier frequency shifting, as we expect the bottom and top part to be in the center, but this is not the case. This discrepancy can be explained by aliasing. To illustrate this, consider the other plots in figure 4.7. The second plot, in figure 4.7b, has no aliasing occurring and we observe the pattern we expect to observe. Note that the pattern is close to the borders of the domain. If we now consider the third plot, in figure 4.7c, we can see that at the edges the pattern starts to 'fold' over. Then, if we decrease the distance even more to 15 cm, we obtain figure 4.7d, in which it has folded more. If we continue with this, we will eventually get the result we saw in figure 4.7a.



(a) 8 cm (b) 20 cm

(c) 18 cm (d) 15 cm

Figure 4.7: Image in the far field for point sources at different distances to the SLM. This figure shows aliasing occurring as the point source is brought closer. Note that the SLM does not induce any phase shift.

Aliasing occurs when a certain signal, in this case the incoming field of the point source, is sampled with too little samples, resulting in certain high and low frequency components becoming indistinguishable. In the case of the discrete Fourier transform, carried out here as part of Fraunhofer diffraction, this results in higher frequency components outside the domain being 'folded' back into the domain resulting in the curious banding patterns seen in figure 4.7a and the progression we see in figure 4.7. One may wonder why we did not encounter this phenomenon in the previous experiment, but in fact we did, albeit not so prominent. Consider figure A.2 in the appendix, where the desired profile of the previous experiment has been plotted on a logarithmic scale. In this figure we can clearly see the repetitive pattern in all four quadrants. This pattern is characteristic of aliasing. However, note that we are looking at very small values there, so we may neglect

this.

The difference between both experiments is, that with a point source we add additional high frequency components to the near field, namely the spherical shape of the wave front. This spherical wave front is created by multiple high frequency components, which, due to limited resolution, experience aliasing. When a simple plane wave is used, the wave front of it is simply a zeroth order frequency, resulting in no aliasing. Hence, the aliasing effects we did observe in figure A.2 in the appendix, are due to the phase distribution imposed by the SLM.

To mitigate the effects of aliasing, a higher resolution for the simulation must be used. Trying out multiple higher resolutions, resulted in needing a resolution larger than 8192×8192 to obtain an acceptable level of aliasing. Unfortunately, when employing backpropagation, an enormous amount of computer memory is needed to keep track of each operation. At resolutions higher than 8192×8192, the amount of memory needed becomes infeasible to allocate. Therefore, the rest of this experiment will not be carried out, as lower resolutions do not yield the accuracy that is desired. In the discussion chapter recommendations for future research regarding this issue will be presented.

4.4. Single variable angle incident plane wave: solution space

This experiment has been carried out with the following settings: Simulation resolution of 2048×2048 with the SLM scaled down to 192×108, so a near field size of $16.4\,\text{cm} \times 16.4\,\text{cm}$ and a far field size of $0.16\,\text{cm} \times 0.16\,\text{cm}$ due to a propagation distance of 20 cm. This simulation uses a wavelength of 633 nm. In the description of the experiment it was said that angles in the range $-30°$ till $30°$ would be used, but this range is too big for the chosen far field size. If the angle becomes too large, this will result in the profile being 'folded' back into the far field domain. In reality, the profile will just appear outside the far field domain. Therefore, the maximum angle range we can have is: $\tan^{-1}\left(\frac{0.0016\,\text{m}}{0.2\,\text{m}}\right) = 0.45°$. Hence, the range will be restricted to $-0.22°$ till $0.22°$. During training the samples that produce the lowest loss are kept. Once training is done, by means of triggering the early stopping criterion with an update parameter of 10, the best sample is selected. Important to note is that the early stopping criterion stops training after 10 epochs have not improved the loss. Since different angles can result in large loss values, due to the zeroth order being located somewhere else, the number of samples per epoch is increased from 5 to 20 samples.

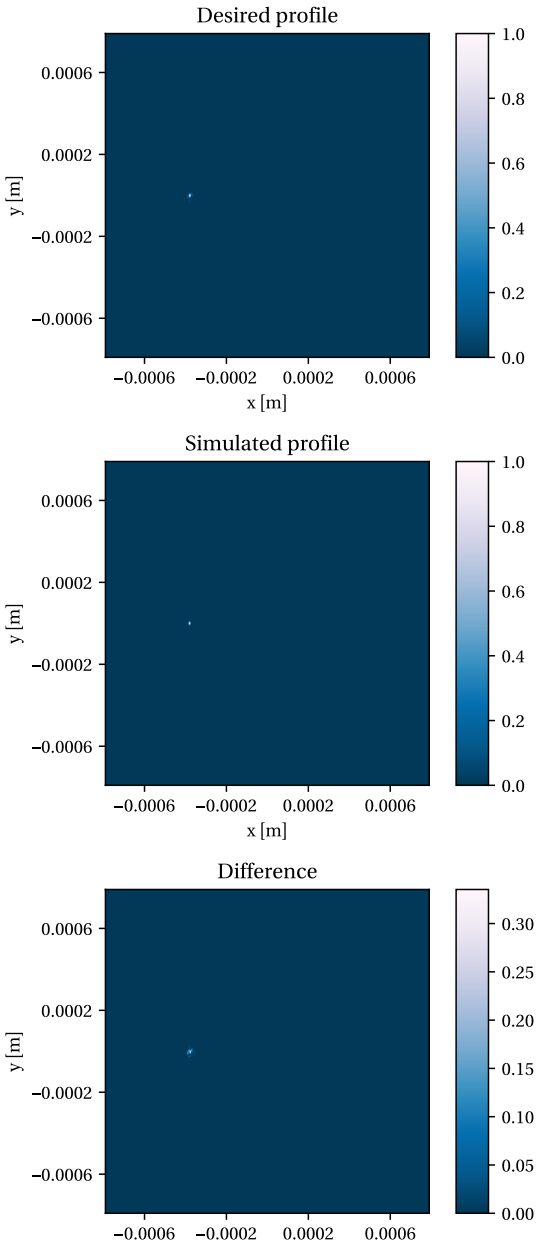
This setup results in the following images, as seen in figure 4.8a. For this experiment an angle of $-0.0019\,\text{rad} = -0.11°$ was chosen at random. Together with 5×5 random control points this resulted in the desired intensity profile shown first in the figure. After training for 19 epochs, with each epoch being 20 samples, a loss value of $3.31 \cdot 10^{-5}$ was obtained. The best angle was found to be -0.019 rad, rounded to two significant figures, which is equal to the initial value.

Since the pattern is quite small in the plot, we further investigate the pattern with the use of figure 4.8b, where the pattern is zoomed in and the contrast boosted. The contrast is boosted by plotting $|U|$ instead of $|U|^2$ and by the use of a non-sequential color map. From this figure we can immediately see that only the central spot corresponds well, but the surrounding areas have no meaningful similarity. This can be explained by the fact that during training the network was optimized for different random angles. If the angle is not the desired angle the whole pattern will be shifted left or right from the desired position. The network is then trying to optimize this by tweaking the control points such that the pattern is smeared in the direction of the desired spot. Hence, the network is on average very little trained in obtaining the right pattern when it is at the right angle.

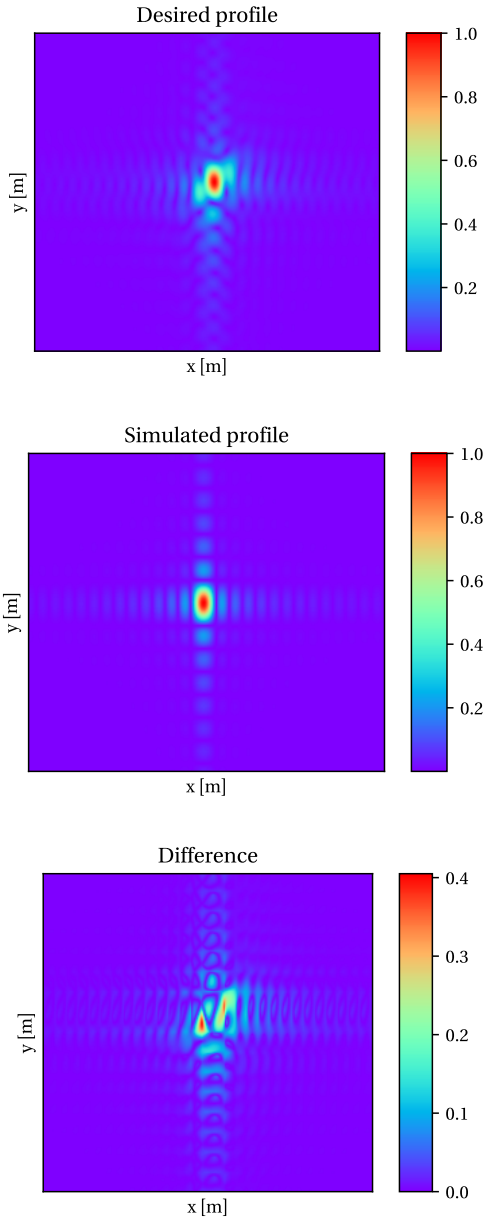
In addition, the approach of training with a random angle each time results in erratic loss behaviour as seen in figure 4.9. This can be explained by the fact that a wrong angle has a loss in the order of 10^{-4} and a correct or close to correct angle has a loss in the order of 10^{-5}. If we take averages of 20 samples per epoch to calculate the loss, then certain average values will be pulled down when a correct or close to correct angle is present. This also explains why the loss function seems relatively constant when omitting big downward jumps.

In conclusion, this experiment illustrates that using an angle as an input to the network does work, but the network does not really learn from it. The angle of a plane wave is simply too sensitive, meaning that changing the angle results in enormous changes in the far field, as the zeroth order shifts with it. Therefore, the network does not really add much, as the network itself cannot control this parameter. In essence, we are doing a random Monte Carlo search on the best angle, as we save the sample that has the lowest loss during training.

In the discussion chapter recommendations regarding this problem will be presented.



(a) In the first two plots the field intensity is plotted using a sequential color map. In the last plot the absolute difference between these two intensities is shown.



(b) In the first two plots the moduli of the field are plotted using a non-sequential color map. The last plot is the difference between the two. The spatial axis labels are omitted in this figure, as this is purely for pattern comparison.

Figure 4.8: Result of finding the optimal angle and control points for a variable angle plane wave experiment. The left figure is the full scale result with a sequential color map. The right figure is a contrast boosted ($|U|$ plotted instead of $|U|^2$ and using a non-sequential color map) zoomed in depiction of the pattern without scale indicated axes.

4.5. Single perpendicular incident plane wave: recreating the university logo

In this experiment recreating the university logo was attempted. The to be replicated logo was shown in figure 3.5 in chapter 3. For this experiment the logo was adjusted to be purely binary black and white, i.e.

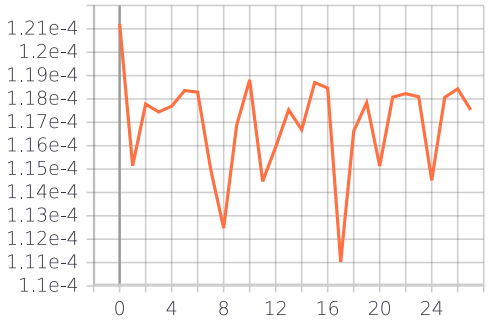


Figure 4.9: Plot of the loss over the epochs for the variable angle incident plane wave. On the horizontal axis the epoch index is shown and on the vertical axis the loss value as calculated by the intensity MSE loss function is shown.

where blue or black is present the value is one and otherwise it is zero. This experiment was carried out at a resolution of 4096×4096 and the SLM at full scale, meaning each pixel in the resolution of the SLM, 1920×1080, corresponds one to one with a pixel in the near field. A perpendicular incident plane wave was used to construct the near field. The resulting intensity profile was created by a propagation distance of 20 cm.

4.5.1. Results of using 5×5 control points

When using 5×5 control points for an order 3 B-spline surface phase distribution, we obtain the plot in figure 4.10. The loss value is $3.38 \cdot 10^{-3}$ in terms of the intensity MSE loss function. This figure shows the resulting intensity profile on a sequential color scale with left the result of the simulation and right a zoomed in version. In these images we can see that the neural network was able to find the control points such that the intensity profile resembles the general outline of the logo, but not the finer details. However, this is to be expected when it can only control 25 control points. For further analysis, we will consider the found phase distribution. This is plotted in figure 4.11, where we use a cyclic color map and the values have been taken modulo 2π. Observing this image we see immediately that we have a lot of fast changing values, indicated by the bands that occur. When the B-spline jumps between two control points with large values outside the $-\pi$ and π range, we get an area with a steep slope between them. When such an area is taken modulo 2π, we get that it oscillates rapidly between $-\pi$ and π. To further substantiate this claim, consider figure 4.12. In this figure the B-spline surface is plotted in 3D and we readily observe that it jumps from -100 rad to 100 rad, which explains the oscillatory we observe in figure 4.11.

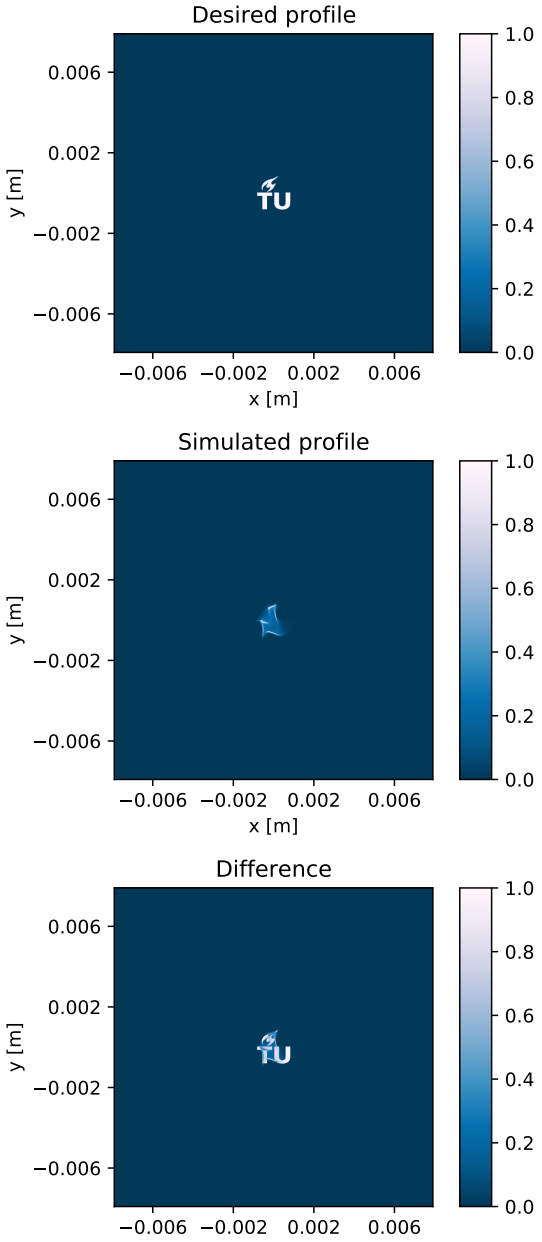
Interesting to note is that this oscillatory behaviour looks analogous to Fresnel lenses. A Fresnel lens has an enormously small focal length, resulting in a very thick lens. To mitigate the issue of having to use large amounts of glass, circular discontinuities are introduced where jumps of multiples of 2π in phase are taken to reduce the thickness of the lens. To illustrate this, consider figure 4.13, where a cross section of a Fresnel lens and its corresponding thick lens are shown schematically.

4.5.2. Results of using 20×20 control points

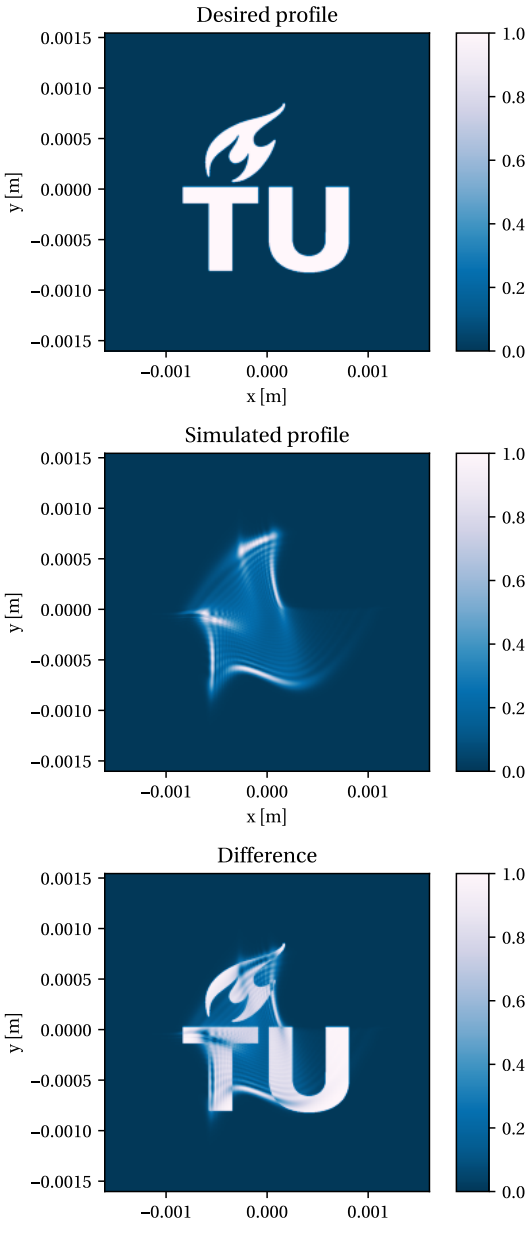
To further improve the results obtained using 5×5 control points, we increase the number of control points to 20×20. Increasing this number much more is not feasible as for each update in the neural network the whole B-spline has be to regenerated, which is computationally expensive. Therefore, this amount of control points was chosen as a maximum. Using this amount results in each update taking approximately 1 second on a GPU, whereas the 5×5 number of control points has a throughput of 16 updates per second on a GPU.

The results of using 20×20 control points are shown in figure 4.14. In this figure, the first subfigure is the full scale result and the second subfigure is a zoomed in version. When looking at the zoomed in version we see that the simulated result actually does read 'TU', but very faintly. There is a streak of scattered light from the bottom of the 'U' up to the top of the flame that degrades the overall image quality.

The loss value is $3.58 \cdot 10^{-3}$ in terms of the intensity MSE loss function. This loss value is higher than the earlier found loss value of $3.38 \cdot 10^{-3}$ for the 5×5 control points case. In the description of this experiment in the experiments chapter, it was said that the number of control points was yet to be determined depending on



(a) Full scale result.



(b) Zoomed in result.

Figure 4.10: Result of recreating the university logo with a 5×5 control points order 3 B-spline surface phase distribution. The intensities are plotted on a sequential color scale and are normalized such that the maximum intensity is 1.

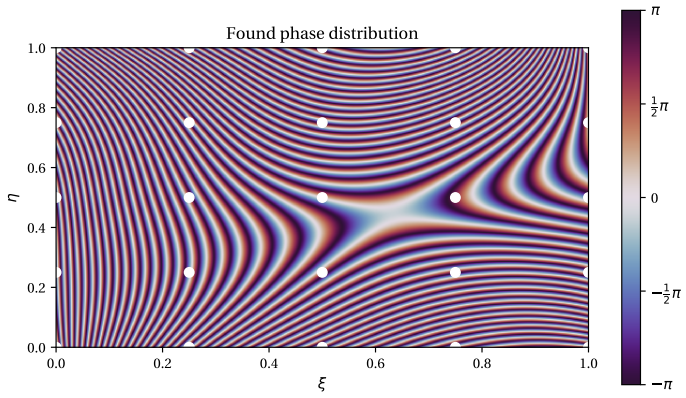


Figure 4.11: Image of the found phase distribution for recreating the university logo with 5×5 control points. In this plot the white dots represent the location of control points.

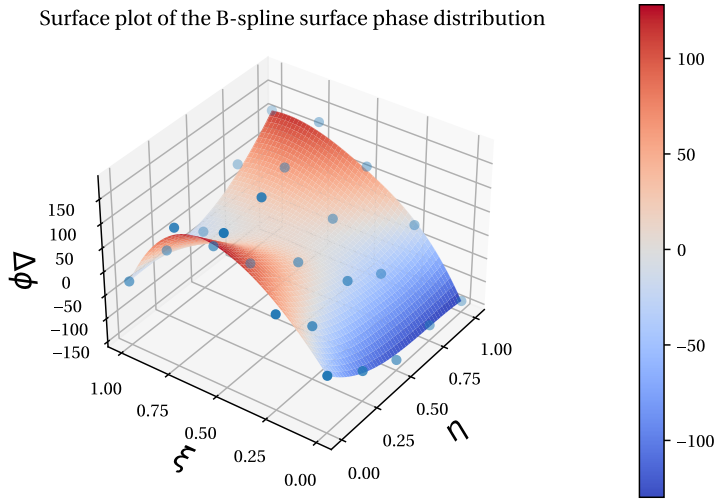


Figure 4.12: The 3D surface of the found B-spline surface for recreating the university logo with 5×5 control points. The blue points indicate the control points used to construct the B-spline surface. The values shown here are the phase in radians.

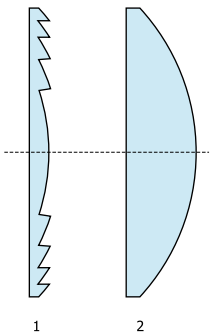


Figure 4.13: Example of a Fresnel lens. In (1) the Fresnel lens is shown and in (2) the corresponding thick lens is shown. This figure was created by Pko [11].

the complexity needed. This statement implicitly assumed that more control points would result in a better image. Visually, we can see that the letters themselves are now readable with more control points, but the loss value is worse. This discrepancy may be explained by this specific run, as each neural network is initialized with random weights and biases.

Next, we will further investigate the phase distribution that was found by the neural network. For this result, consider figure 4.15, where the phase distribution is plotted modulo 2π with a cyclic color map. We can observe that this phase distribution is much rougher than the one for 5×5 control points, which probably explains why the result in figure 4.14b looks so dotted. The increased 'roughness' of this phase distribution may be a general trend when increasing the number of control points, as more fine grained control of the sharp edges in the logo requires precise control over the phase. This is a consequence of the fact that Fraunhofer diffraction relies on the Fourier transform. In figure A.4 in the appendix the result of using the Gerchberg-Saxton algorithm [12] for finding the optimal phase distribution is shown. This method can fine tune each of the 1920×1080 pixels of the SLM directly, instead of using control points. Important to observe in this image is how rough and pixelated this image looks in comparison to the B-spline surfaces. However, it can almost perfectly reconstruct the university logo, as seen in figure A.5 in the appendix.

In figure A.3 in the appendix the B-spline surface is plotted in 3D along with its control points. In this figure we can see that the values range from -40 to 40, which explains the banding behaviour we see in figure 4.14. This is homologous to the banding behaviour we saw in the previous subsection.

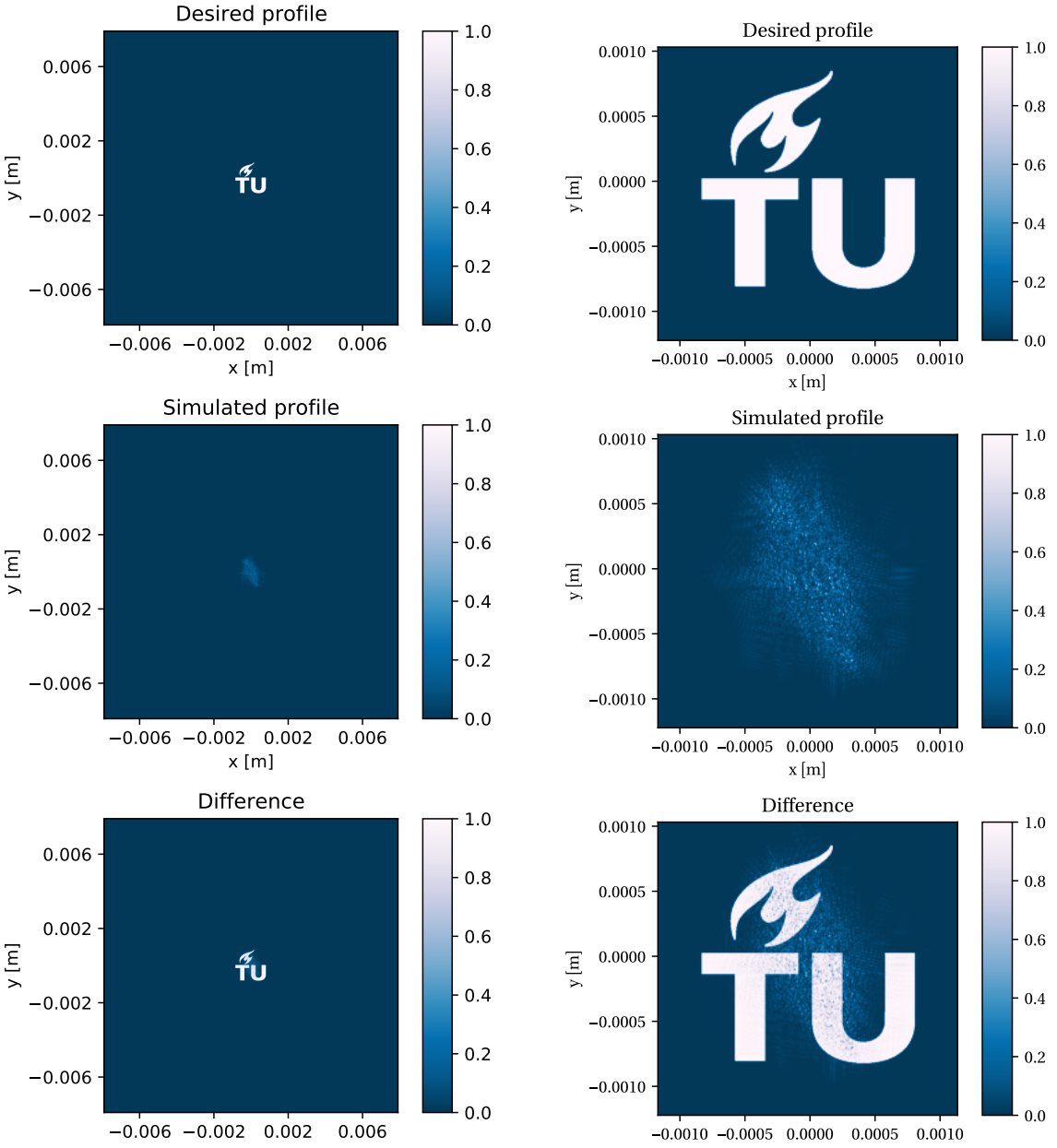
4.5.3. Loss function

Although it was said in section 4.2.2 that the intensity MSE loss function would be used from now on, it proved to be visually better for the university logo to use the MSE loss function for optimizing the network. Therefore, in the previous two results the MSE loss function was used. A possible explanation for this result is the fact that the intensity MSE loss function prioritizes larger differences in intensity over smaller differences in intensity, while the MSE loss function uses the moduli of the field and therefore puts less emphasis on larger differences. This last property is especially useful when larger areas need to be lit uniformly, as is the case with the university logo.

4.6. Two variable angle and amplitude incident plane waves: solution space

This experiment is similar to the variable angle experiment in section 4.4, but with extra degrees of freedom. Specifically, two plane waves can have differing amplitudes and each plane waves can be rotated along two axes, as depicted in the description of this experiment by figure 3.6. In the previous variable angle experiment, we saw that the angle needs to be restricted, as to make sure the zeroth order spot lays inside the far field domain and does not fold over. Since the variable angle experiment displayed the intensity pattern as quite a small dot, we will use the settings of the first experiment. These settings make the far field domain smaller and hence the range of possible angles, but it makes the pattern better visible. The resulting angle range becomes then: $\tan^{-1}\left(\frac{0.000316\,\mathrm{m}}{0.2\,\mathrm{m}}\right) = 0.091°$, which results in the range $-0.045°$ to $0.045°$. Like before, the sample with the lowest loss value is kept and an early stopping criterion with patience parameter 10 is used. So after 10 epochs of no improvement the training will be halted. During training the loss will fluctuate much, as angles are sensitive parameters. Therefore, each epoch will consist of 20 samples.

The results of this experiment are shown in figure 4.16. In the desired profile plot we can see that one plane wave yields the bright spot just off center and next to it is a fainter plane wave. In the simulated profile we observe that the network got the bright spot almost right, but completely missed the fainter spot. This can be explained by the fact that the network is trained on many randomly generated samples. If no input sample has the perfect angle, then the network will fail to optimize. Hence, just as described before, we are essentially carrying out a Monte Carlo search. In the discussion chapter recommendations regarding this issue will be done.



(a) Full scale result.

(b) Zoomed in result.

Figure 4.14: Result of recreating the university logo with a 20×20 control points order 3 B-spline surface phase distribution. The intensities are plotted on a sequential color scale and are normalized such that the maximum intensity is 1.

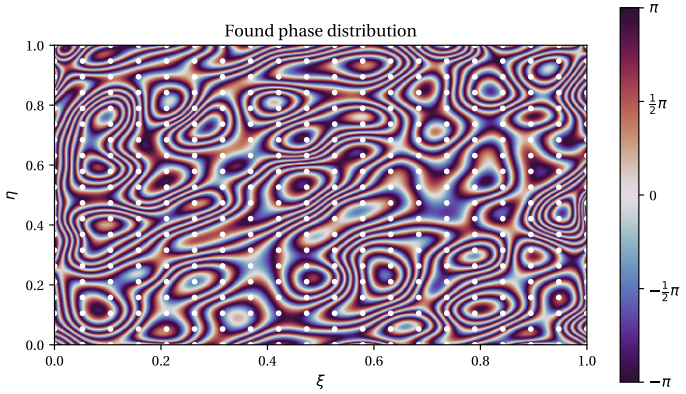


Figure 4.15: Image of the found phase distribution for recreating the university logo with 20×20 control points. In this plot the white dots represent the location of control points.

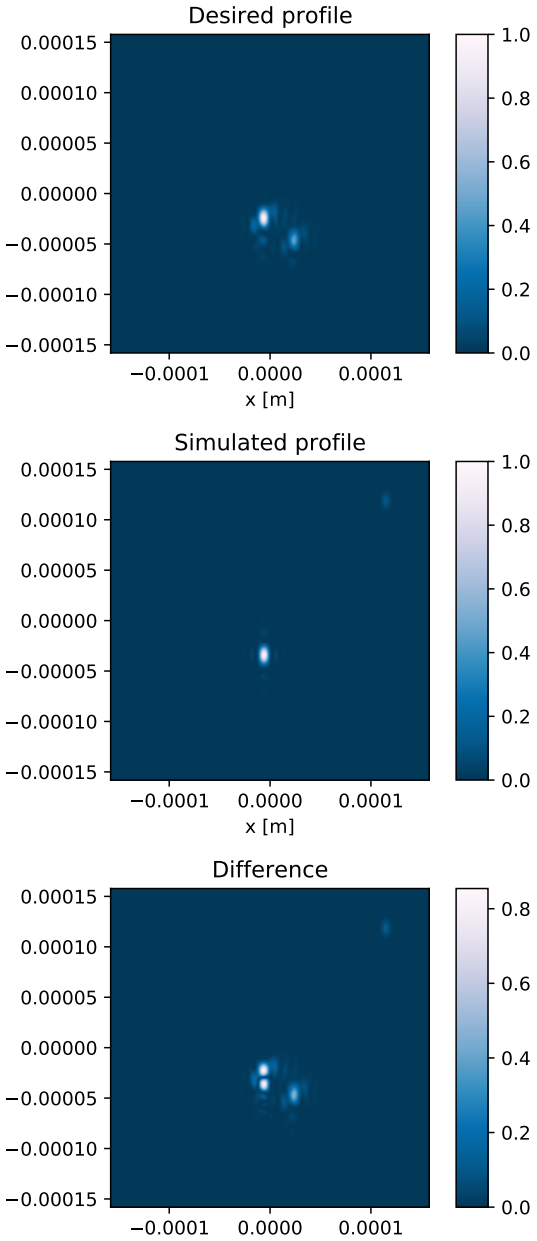


Figure 4.16: Result of finding the optimal angles, amplitudes and 5×5 control points for two plane waves. The intensities are plotted on a sequential color scale and are normalized such that the maximum intensity is equal to 1.

5

Discussion

In this chapter issues that arose in the results chapter will be discussed. During this discussion recommendations for future research will be presented.

5.1. Aliasing issues

In section 4.3 we saw that using a point source yields a spherical wave front, which in turn leads to aliasing issues when working at a limited resolution. This phenomenon is inherent to using a discrete Fourier transform when carrying out Fraunhofer diffraction. A possible solution one may think of is using the Rayleigh-Sommerfeld diffraction integral directly and then use numerical techniques to calculate this integral. However, this still requires discretization of the problem that may result in aliasing effects.

A better way to mitigate this issue might be with the use of ray tracing techniques. In such techniques classical ray optics is used to construct an image. With ray tracing we need to reinterpret the construction of an image, as ray tracing does not simulate interference or diffraction, since it does not capture the phase of rays but merely their angles and positions. For that matter, lenses can be thought of as bending incoming rays with Snell's law of refraction, instead of adjusting the phase of an incoming field. In the case of the point source experiment, a ray originates from the point source, it passes through the SLM which bends it and then it ends at the screen. The image is then formed by where the rays hit the screen. Such images are also called 'caustics'. Important to note is that, unlike Fraunhofer diffraction, ray tracing does not need some minimum propagation distance for the approximation to hold.

Ray tracing is a computationally intensive process, but with the advent of faster GPUs that promise 'realtime' ray tracing, like the recent GPUs produced by NVIDIA [13], it may be possible. There are two main types of ray tracing, forward ray tracing and backward ray tracing [14]. In backward ray tracing, a ray originates from the camera, or in this case the screen, and is then propagated backwards, as described by Lu et al. [14]. If the ray hits a light source, the pixel in the camera from which it originated is colored. This method is more efficient for complex scenes with many light sources and where lots of rays will never reach the camera. However, this is not what happens physically and with the addition of optimization tricks, the result of such a simulation will not be reproducible in the real world. Therefore, 'forward' ray tracing is needed, where the ray originates from light sources. What is more, to render caustics properly, bi-directional ray tracing is needed. In this technique forward and backward ray tracing are combined to render the caustic.

In addition, when these techniques need to be combined with machine learning , it is required that these simulations are gradient preserving for backpropagation to be possible. This has already been done for backward ray tracing by Mirman [15], but not yet for forward ray tracing. Therefore, using ray tracing in conjunction with machine learning will require future work to create a gradient preserving forward ray tracing simulation.

5.2. Focused loss function and background

In section 4.4 the results of the variable angle experiment are presented. In this section we saw that getting the angle wrong results in a loss value in the order of 10^{-4} and getting the angle right results in a loss value of

10^{-5}. This is quite a small difference due to taking the average over $N \times N$ points. In addition, in that specific case we might only care for the result we obtain inside a certain radius around the center of the intensity distribution. The same applies for the experiment in which we tried to recreate the university logo. How the intensity behaves far away from the center is not relevant. Therefore, it might prove useful to create the loss function such that it only considers a certain area of importance and neglects the outer regions. This should increase the difference in loss values for a 'good' and 'bad' result, which in turn allows training to occur more targeted.

Another useful addition might be blurring the university logo's intensity profile, as it is hard to recreate the sharp edges the logo is composed of with the limited resolution we have available when using a B-spline surface. This will hopefully reduce the emphasis on recreating sharp edges, which might focus the attention to better coloring of the logo.

5.3. Physics-informed neural network and Monte Carlo search

The first experiment did not use an input for the network, but simply a 1 as input. Therefore, this architecture does not qualify for the definition of phsyics-informed neural network; it is simply an optimization algorithm. When we did try to make it 'physics-informed' in the experiment with a variable angle, the network did not really learn how to adjust the parameters to obtain the desired profile. Instead, we selected the sample with the lowest loss at the end, and this was largely determined by the angle. Hence, this architecture essentially reduces to a Monte Carlo like search algorithm.

From this we conclude that the experiments chosen did not show the true potential of the specific physics-informed neural network inspired architecture employed here. This is mainly due to the fact that we chose static experiments, where only the control points change, and that we chose experiments with inputs to the network that were too sensitive. For future research it may be interesting to consider more advanced experiments in which an intensity profile can be generated with multiple different configurations.

In addition, if one wants to find an optimal angle, or any parameter that is sensitive to change, a preemptive Monte Carlo search can be carried out. The result of this search can then be used as a static parameter when training the network for a specific optical setup. In the case of the double plane wave experiment, a Monte Carlo search on the angle can be carried out in the following way: First, the control points of the SLM are simply kept at zero, as changing the phase does not alter the location of the intensity spot too much. Then the first plane wave is incident under some random angles. When after a certain number of tries a lowest loss value has been found, the resulting spot is subtracted from the desired intensity profile, as to avoid obtaining the same angles for the other plane wave for the same spot. On this new profile an extra Monte Carlo search is carried out, until the second plane wave has a lowest loss after a certain number of tries. These found angles are then set as static parameters and a neural network can then be trained to find the remaining parameters.

5.4. B-spline surfaces and the Gerchberg-Saxton algorithm

In the experiment on recreating the university logo we saw that phase distributions based on B-spline surfaces were able to recreate the university logo to some extent, but nowhere near as good as was possible with the Gerchberg-Saxton algorithm [12]. This discrepancy can be explained by the fact that control points allow far less fine tuned control over each pixel in the SLM. We can already observe that adding more control points results in a rougher phase distribution, when we compare the results of the 20×20 and 5×5 case. These results seem to indicate that using B-spline surfaces for such tasks is futile. However, B-spline surfaces have their use, as using B-spline surfaces results in smoother phase distributions and hence smoother freeforms when physically produced. The resulting phase distribution of the Gerchberg-Saxton algorithm is harder to produce as it requires enormous discontinuities in the material, which itself can result in unintended aberrations. Nevertheless, an important observation is that when the control points of B-spline surfaces get arbitrarily large, areas of steep slope occur in the phase distribution, which makes producing a freeform based on it harder. Then, perhaps, discontinuities need to be introduced to recreate such steep areas. As pointed out in the results of the experiment, this is analogous to the construction of a Fresnel lens.

6

Conclusion

The setup proposed in this thesis, with a neural network, B-spline surface and Fraunhofer diffraction to recreate a desired intensity profile, is promising, but does have its issues. This thesis is therefore a useful extension on the work dony by Imhof, but it does not necessarily supersede the work of Imhof. When using a simple perpendicular incident plane wave, the setup is able to recreate solutions in the solution space and even recreate the university logo to some extent. However, in these experiments the neural network was used with a 1 as input, reducing the setup to merely an optimization algorithm, which is not actually 'physics-informed'.

In two experiments it was attempted to have plane waves incident under an angle and this angle was then the input of the neural network. In one case the setup was able to find the desired angle, while in the other case it did not. Since the angle is such a sensitive parameter, in terms of shifting the pattern much in the far field, training on random input angles reduces the setup to a simple Monte Carlo search. This also explains why the setup failed to find the solution in the second experiment, as that specific combination of angles must not have occurred randomly during training.

The problem of the setup not being 'physics-informed' or merely being a Monte Carlo search in these specific instances, can be resolved by considering experiments with more variable parameters that are not too sensitive. For instance, an experiment with an array of lights or an experiment in which multiple solutions will produce the same intensity profile.

In one experiment the setup failed completely. In this experiment a point source was used to create a spherical wave. When the point source is close to the SLM, its wave front will curve rapidly when viewed from the SLM. Fraunhofer diffraction is based on the Fourier transform, which results in the discrete Fourier transform being used when simulating it numerically. This method fundamentally suffers from aliasing issues when the to be transformed field, has too many high frequency components. For the spherical wave fronts emanating from the point source, this was the case. Therefore, experiments using point sources in this thesis have been stopped, as the simulation results cannot occur physically.

A possible solution for this, is to use ray tracing instead of Fraunhofer diffraction for simulating the intensity profile. With the advent of new GPUs it has become feasible to do ray tracing in a reasonable amount of time. However, current techniques running on the GPU do ray tracing 'backwards', while 'forward' is needed for a physical simulation. In addition, this 'forward' technique needs to be adjusted as to make sure it preserves gradients across operations.

In this thesis the focus was on using B-spline surfaces for the phase distributions of the SLM. Using B-spline surfaces limits the resolution and sharpness of images in the far field. Algorithms such as Gerchberg-Saxton are able to tweak each individual pixels of the SLM, which results in a larger solution space in the far field and therefore better reconstruction of images. However, freeforms based on phase distributions created by the Gerchberg-Saxton algorithm require many discontinuities, which can be difficult to construct physically. Therefore, B-spline surfaces are a natural way of limiting the amount of discontinuities needed for construction.

Acknowledgements

I would like to thank my supervisors Dr. Aurele Adam and Dr. Matthias Möller for guiding me in this project using discussions during our meetings and their invaluable feedback on my thesis. They assisted me in researching my topic and helped me shape my goal. In addition, I would like to thank Alex Heemels for our discussions and helping me out when I had issues in my program. He was always readily available when I could not figure out what was wrong. All three were of immense support during my research and enabled me to work on my thesis at home during the COVID-19 pandemic. Also, I would like to thank my friends and fellow students Otto Broers and Hidde de Bos for pre-reading my thesis and the discussions we had about my thesis. Lastly, I would also like to thank my parents for supporting me in the last three years. Without the support of all these people, this thesis would not lay before you now.

Bibliography

[1] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378:686–707, 2019.

[2] J. Imhof. Freeform lens predictions by a neural network and b-splines. 2021.

[3] J.W. Goodman. Introduction to Fourier Optics. McGraw-Hill physical and quantum electronics series. W. H. Freeman, 2005.

[4] David George Voelz. Computational Fourier optics: a MATLAB tutorial. SPIE, 2011.

[5] T. J. R. Hughes J. A. Cottrell and Y. Bazilevs. Isogeometric Analysis: Toward integration of cad and fea. Wiley, 2009.

[6] Ke-Lin Du and M. N. S. Swamy. Neural Networks and Statistical Learning. Springer, 2019.

[7] PyTorch. Autograd mechanics, 2019.

[8] HOLOEYE Photonics AG. Pluto-2, phase only spatial light modulators, 2020. manual version 4.5.

[9] Kenneth R. Spring and Thomas J. Fellers. Human vision and color perception.

[10] PyTorch. Upsamples, 2019.

[11] Pko. Fresnel lens.svg from wikimedia commons, 2006.

[12] R.W. Gerchberg and W.O. Saxton. A practical algorithm for the determination of phase from image and diffraction plane pictures. 1972.

[13] NVIDIA Corporation. Nvidia rtx ray tracing, 2021.

[14] Charity Lu, Alex Roetter, and Amy Schultz. Types of ray tracing, 1997.

[15] Matthew B. Mirman. Mentisoculi: Mentisoculi pytorch path tracer, 2018.

A

Appendix

A.1. SLM Specifications

This thesis simulates an SLM modeled after the Pluto 2 (VIS-096) SLM. It has the following specifications:

Table A.1: Specifications of the SLM on which this thesis is based. Extracted from [8].

Active area	15.36 mm×8.64 mm
Resolution nominal	1920 × 1080
Pixel pitch	8.0 μm
Wavelength (nm)	450-650
Maximum phase shift @ 633 nm	2.5π

A.2. JSON configuration file format

In figure A.1 an example JSON configuration file is shown. It consists of two important sections: `topology` and `settings`. The `topology` section describes what the inputs and outputs of the network are, including the suitable ranges, shape and type. The `settings` section describes what scenario and profile are used. In addition, it describes what model is used and what hyperparameters the model will use. In this specific example a network with two hidden layers of size 20 is used.

A.3. Git repository

The code used throughout this thesis to run experiments and simulations can be found on the official Delft University of Technology GitLab instance: Gitlab repository.

A.4. Extra Results

In this appendix section results that are too much to fit into the results chapter are displayed. This concerns figures A.2 to A.5 and table A.2.

Figure A.1: Example JSON configuration file used in this project.

```
{
    "topology": {
        "inputs": {
            "bias": {
                "type": "binary",
                "size": [1],
                "range": [1, 1]
            }
        },
        "outputs": {
            "control_z": {
                "type": "float",
                "size": [5, 5],
                "range": [0, 6.148]
            }
        }
    },
    "settings": {
        "imports": ["base/slm"],
        "N": 1024,
        "slm_scale": 0.02,
        "patience": 10,
        "profile": "random_profile",
        "scenario": "slm_planewave",
        "model": "basic_PINN",
        "layers": [20, 20],
        "lossfunction": "MSEIntensity",
        "_seed": 1.0
    }
}
```

Table A.2: Comparison of convergence speed for the MSE and Intensity MSE loss function. The following criterion is used: the amount of epochs it takes for the loss to have decreased by at least 90%. For the MSE loss function the value between brackets indicates what the criterion does when the intensity MSE loss function is used. This gives averages: 16.2 (8.0) epochs for the MSE loss function and 32.6 (15.9) for the Intensity MSE loss function. The values between brackets denote the sample standard deviation. For the MSE loss function, we additionally have on average 29.6 (9.2) when using the intensity MSE to calculate the criterion.

Seed	**MSE**	**Intensity MSE**
1.0	10 (19)	16
2.0	12 (22)	25
3.0	21 (30)	56
4.0	28 (41)	41
5.0	10 (36)	25

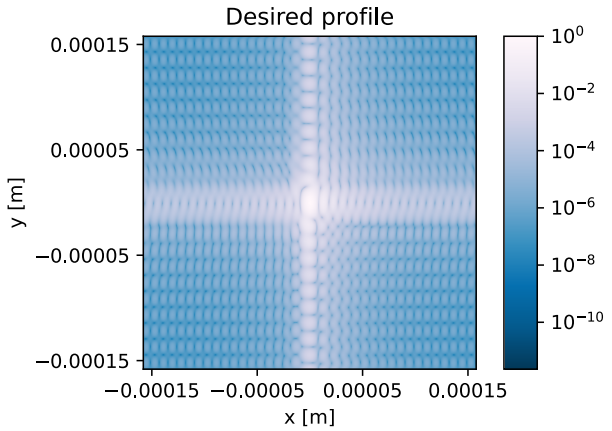


Figure A.2: Plot of the intensity profile with seed 1.0 in the single planewave experiment, but plotted on a logarithmic scale to enhance the hidden aliasing effect.

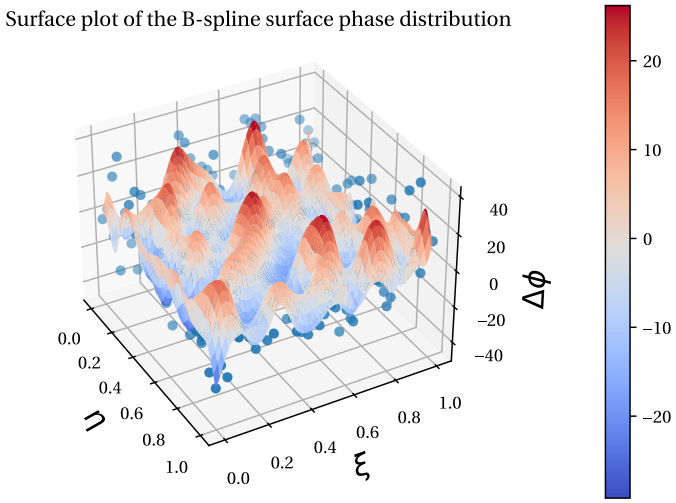


Figure A.3: The 3D surface of the found B-spline surface for recreating the university logo with 20×20 control points. The blue points indicate the control points used to construct the B-spline surface. The values of the phase shown here are in radians.

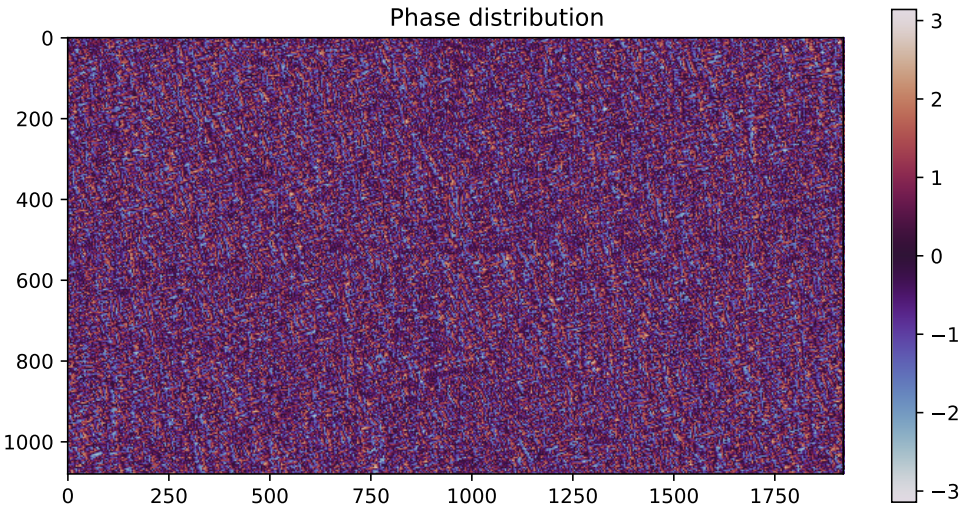


Figure A.4: Phase distribution as found by the Gerchberg-Saxton algorithm [12] to recreate the university logo. The values shown are the phase in radians. This image has been created using the script in the Gitlab Repository created by Alex Heemels.

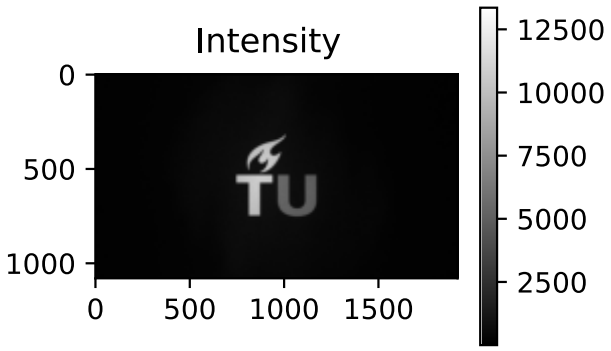


Figure A.5: Result of recreating the university using the Gerchberg-Saxton algorithm. This is the result in the far field using the phase distribution shown in figure A.4.