Investigation of Alternative Contactless Optical Surface Reconstruction Methods

RICKARD BOLIN, ERIC ROSTEDT

Master's thesis
2021:E7

LUND INSTITUTE OF TECHNOLOGY
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Abstract

Object and surface reconstruction in 3D is commonly used in production tolerance validation. Depending on the required level of accuracy, a multitude of different methods are available, each with their own advantages and disadvantages.

This thesis aims to investigate alternative surface reconstruction methods which utilise that all possible light paths are known along a given one-dimensional curve on the surface of a material with known refractive index. This makes it possible to calculate the expected light intensity for the curve. By comparing the expected intensity to a reference intensity from when there is no surface to reflect off, it is possible to deduce information about the surface.

Several methods utilising this information to reconstruct the surface of an object are investigated and evaluated. The types of investigated methods range from one-dimensional iterative methods to Convolutional Neural Network based Encoder-Decoder architectures.

The evaluation shows that it is possible to deduce information about the general shape of the surface, but that the non-linear nature of the problem makes it difficult to identify any fine details. The methods showing the most promising results use a combination of principal components and simple geometric relationships in the data to reconstruct the surface.

Keywords: Geometric transformation, Principal Component Analysis, Helmholtz equation, CNN based Encoder-Decoder

Acknowledgements

We would like to express our sincerest gratitude and appreciation to all who have made this thesis possible. Among those are Maria Sandsten, Filip Cederquist and Gustav Harrysson, who have provided valuable feedback and suggestions of improvement. We would also like to offer a special thanks to our supervisor Johan Lindström for his guidance and interesting ideas throughout this thesis.

Rickard Bolin & Eric Rostedt

Contents

Chapter 1

Introduction

Reconstruction of objects in 3D is commonly used in many different applications. Techniques such as LiDAR point clouds and stereo images can be used to reconstruct large objects and environments, whereas techniques such as Moiré deflectometry and structured light reconstruction can be used to reconstruct the surface of small objects with very high accuracy[4][7].

This thesis aims to investigate alternative contactless optical reconstruction methods that use emitters and detectors to generate light and record the signal strengths after the light has travelled along the surface of a material to estimate the shape of the surface in real time. If the surface could be estimated to a certain degree of accuracy, this method could be used as a quick test during production to assure that production tolerances are met. Alternatively, for the use cases that can adapt to variations in the surface as long as it is known, the estimation could make it possible to increase the production tolerances, resulting in a lower production cost.

The proposed method utilises that, along a given one-dimensional curve on the surface of a material with known refractive index, it is possible to use simple ray tracing techniques to find the path of all rays from one side of the curve to the other. This means that it is possible to estimate how much of the light generated by an emitter on one end of the curve that will reach the detector on the other end, either directly or after reflecting off the surface. It turns out that the surface has a large impact on the amount of detected light.

The proportion between the detected light and expected detected light when there is no surface to reflect off will be referred to as boost factor, and will be described in detail in section 2.1.3. The illustration in figure 1.1 shows how more light paths are enabled as the surface becomes more curved.

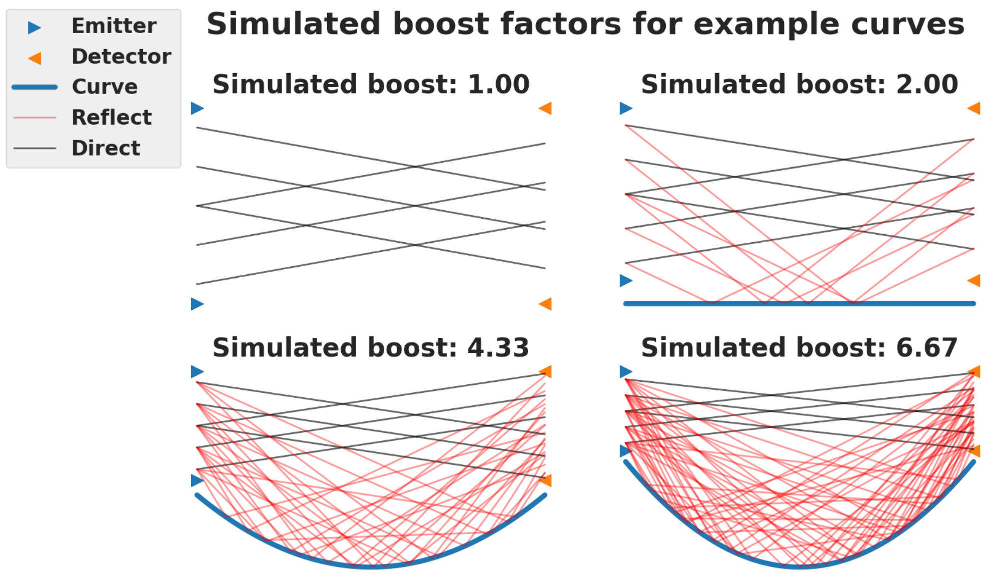


Figure 1.1: Boost factor examples. The top left illustrates the boost factor when there is no underlying curve, top right when there is a flat curve. The two lower figures show how the boost factor increases as the curve becomes more curved.

In the case where there is no material to reflect off, no additional light will reach the detector, resulting in a boost factor of one. With a perfectly flat and large surface, the amount of light reaching the detector will approximately double, resulting in a boost factor of two. As the surface becomes more and more curved, more possible paths for the light are enabled and the boost increases.

Reconstruction techniques such as time-of-flight, structured light and Moiré deflectometry generally work quite well, and this thesis is not an attempt to develop a method with state-of-the-art accuracy or speed. Instead, the methods investigated in this thesis should be considered a complement to the toolbox of existing techniques to be used when the environment and context is especially well suited. The methods proposed in this thesis can be used in a similar fashion as structured light techniques as the optical signals containing information about the surface can be recorded in real time, making it possible to examine how forces applied in different ways affect the deformation over time.

The aim of this thesis is to:

- Investigate different methods to reconstruct a surface using optical emitters and detectors at the edge of the surface.
- Evaluate the different methods on both simulated and real data.

Chapter 2

Theory

In this chapter, a rigorous derivation and explanation of the concept of boost and how it can be simulated is provided, which will be necessary to be able to understand the methods and results.

Additionally, an overview of some fundamental statistics and optics is provided, as well as some theory behind the building blocks of the investigated methods.

2.1 Fundamental optics and boost

2.1.1 Luminous intensity as a function of angle and distance

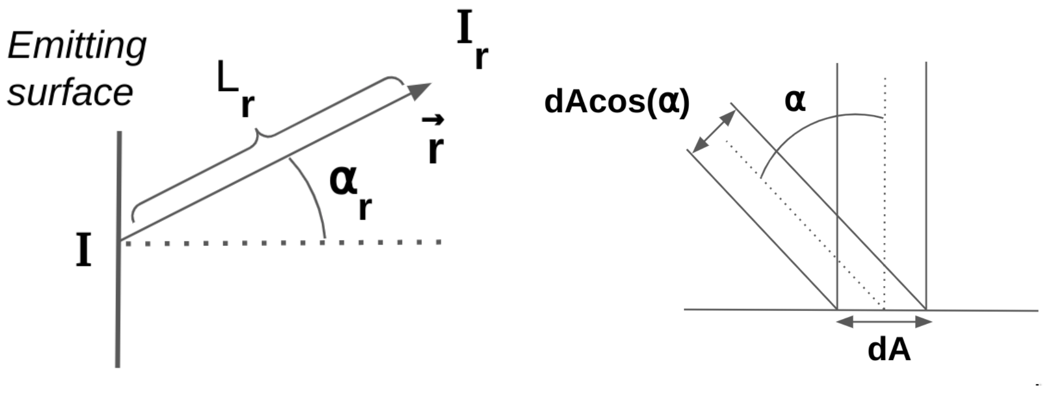


Figure 2.1: Illustration of the light intensity dependence on angle and distance.

The intensity I of light emitted decreases proportionally to the inverse of the squared distance L_r from the emitting surface since the light is distributed as the surface of a sphere, which grows proportionally to the square of the radius of the sphere. This is commonly referred to as the inverse square law [12] (p. 12):

$$I_r \propto \frac{I}{L_r^2}.$$

The intensity I_r is also directly proportionally to the cosine of the angle between the normal of the surface and the ray $\vec{r}$. To the right in figure 2.1 it is illustrated how much of an emitting surface that can be utilised when viewed from an angle α. This is known as Lambert's cosine law [12] (p. 13):

$$I_r \propto I\cos(\alpha_r). \tag{2.1}$$

2.1.2 Fresnel coefficients

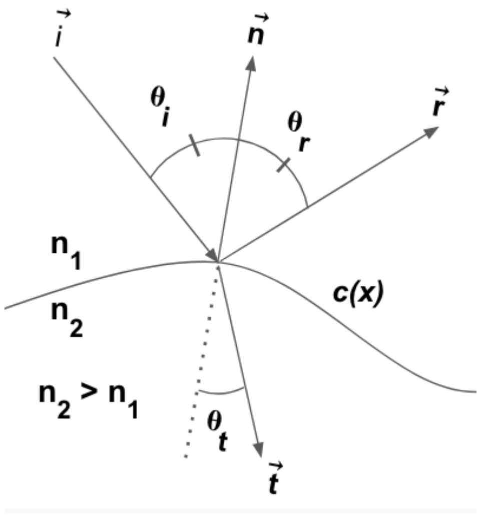


Figure 2.2: Illustration of Snell's law, which describes reflection and transmission when an incoming ray intersects a curve which separates two different media.

Reflection and transmission are physical phenomena occurring when a wave passes from one medium to another. For a differentiable curve $c(x)$ which separates the two media as in figure 2.2, the normalised normal vector pointing perpendicularly to the surface and away from the second medium is found as:

$$\vec{\boldsymbol{n}} = \frac{1}{\sqrt{[c'(x)]^2+1}}\begin{bmatrix}-c'(x)\\1\end{bmatrix}.$$

Let the vector $\vec{\boldsymbol{i}}$ represent a light ray, $\vec{\boldsymbol{r}}$ be the reflected ray and $\vec{\boldsymbol{t}}$ be the transmitted ray after hitting the interface. For simplicity, let the vectors be normalised, i.e have length of 1.

The angle of incidence θ_i is the angle between the incoming vector $\vec{\boldsymbol{i}}$ and the normal vector $\vec{\boldsymbol{n}}$ and is calculated using the dot product:

$$-\vec{\boldsymbol{i}} \cdot \vec{\boldsymbol{n}} = ||-\vec{\boldsymbol{i}}||_2 \cdot ||\vec{\boldsymbol{n}}||_2 \cos(\theta_i) \implies \theta_i = \arccos\left(-\vec{\boldsymbol{i}} \cdot \vec{\boldsymbol{n}}\right).$$

where the implication holds iff the vectors $\vec{\boldsymbol{i}}$ and $\vec{\boldsymbol{n}}$ are normalised. When the incoming ray intersects the curve, it will split into two parts. One is reflected and one is transmitted to the other medium. The law of reflection states that the reflective angle is the same as the angle of incidence. The reflected vector can be calculated using the householder transformation [5]:

$$\vec{\boldsymbol{r}} = \vec{\boldsymbol{i}} - 2(\vec{\boldsymbol{i}} \cdot \vec{\boldsymbol{n}})\vec{\boldsymbol{n}}.$$

The angle of the transmitted ray depends on the *refractive indices* of the media. The refractive index n_j of a medium j is defined as

$$n_j = \frac{v_j}{c},$$

where v_j is the speed of light in medium j and c is the speed of light in vacuum. The relation between the angle of incidence θ_i and the transmitted angle θ_t for a light ray travelling from medium 1 to medium 2 follows Snell's law [12] (p. 17), which states:

$$sin(\theta_i)n_1 = sin(\theta_t)n_2. \tag{2.2}$$

An illustration of Snell's law can be seen in figure 2.2.

To find how much of the light is reflected and transmitted respectively, Fresnel's formulae are used. The relation depends on the incident angle θ_i, the refractive indices of the two media and the polarisation of the incident light.

The Fresnel formulae [12] (p. 495-496) gives the reflection coefficient for parallel polarised light as

$$R_p(\theta_i) = \left(\frac{n_{rel}\cos\theta_i - \cos\theta_t}{n_{rel}\cos\theta_i + \cos\theta_t}\right)^2,$$

and for perpendicular polarised light as

$$R_s(\theta_i) = \left(\frac{\cos\theta_i - \cos\theta_t}{\cos\theta_i + \cos\theta_t}\right)^2,$$

where n_{rel} is the relative refractive index $\frac{n_2}{n_1}$. By solving for θ_t in equation (2.2), it can be eliminated from the equations

$$R_p\left(\theta_i\right) = \left(\frac{n_{rel}^2 \cos\theta_i - \sqrt{n_{rel}^2 - \sin^2\theta_i}}{n_{rel}^2 \cos\theta_i + \sqrt{n_{rel}^2 - \sin^2\theta_i}}\right)^2 \tag{2.3}$$

and

$$R_s\left(\theta_i\right) = \left(\frac{\cos\theta_i - \sqrt{n_{rel}^2 - \sin^2\theta_i}}{\cos\theta_i + \sqrt{n_{rel}^2 - \sin^2\theta_i}}\right)^2. \tag{2.4}$$

Assuming that the material does not absorb any of the light, the law of conservation of energy gives that:

$$\begin{aligned} T_p(\theta_i) &= 1 - R_p(\theta_i). \\ T_s(\theta_i) &= 1 - R_s(\theta_i). \end{aligned}$$

In the case where the incident light is unpolarised, the amount of parallel and perpendicular polarised light is equal [3]. The effective reflection and transmission factors can then be calculated as

$$\begin{aligned} R_{eff}(\theta_i) &= \frac{R_p(\theta_i) + R_s(\theta_i)}{2}. \\ T_{eff}(\theta_i) &= \frac{T_p(\theta_i) + T_s(\theta_i)}{2}. \end{aligned} \tag{2.5}$$

2.1.3 Boost factor

Along a differentiable one-dimensional curve on the surface of a material with known refractive index, it is possible to find the path of all rays from one side of the curve to the other. Thus, it is possible to estimate how much of the light generated by an emitter on one end of the curve that will reach the detector on the other end, either directly or by reflecting off the surface. The increase in detected light caused by light reflecting off the curve compared to when there is no curve to reflect off is called the *boost factor*. The general formula for the boost factor is defined as

$$\text{Boost factor } = \frac{p_{curve} + p_{direct}}{p_{direct}}, \tag{2.6}$$

where p_{curve} is the proportion of the total light reaching the detector by first reflecting off the curve and p_{direct} is the proportion of the total light reaching the detector without any reflections. This section will explain in detail how p_{curve} and p_{direct} are derived and calculated. To simplify the theory and calculations, the length L of the curve is assumed to be much larger than the height of the emission and detection points y_e and y_d (see figure 2.3). This results in only rays with small emission and detection angles φ_e and φ_d being likely to reach the detector, as well as most incidence angles towards the curve being close to 90°, which results in almost no light getting transmitted through the material.

Consider a light source and a detecting surface as in figure 2.3.

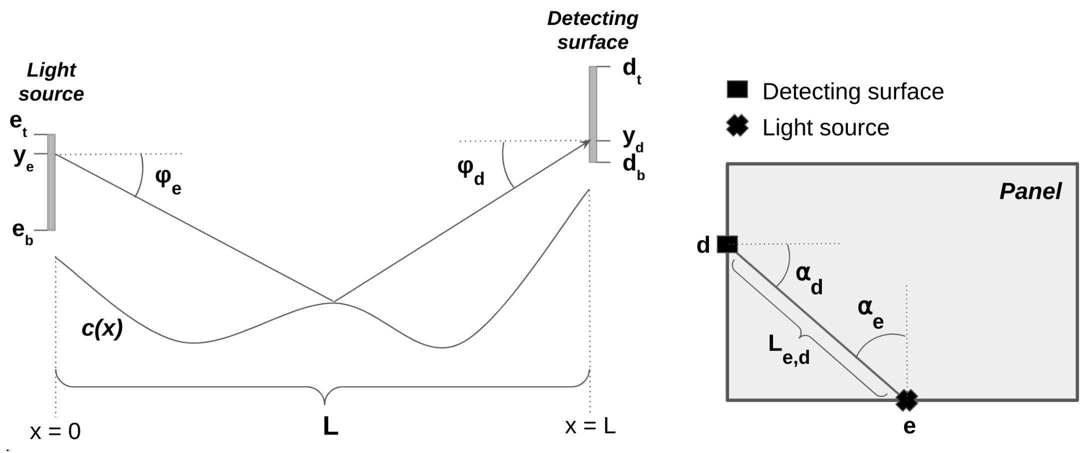


Figure 2.3: Light source and detecting surface. The left figure shows the one-dimensional curve from the light source to the detecting surface. The right figure shows the light source and detecting surface in the geometry from a bird's-eye view.

Let all light be emitted from a point y_e on the light source be separated into only five rays, as in figure 2.4.

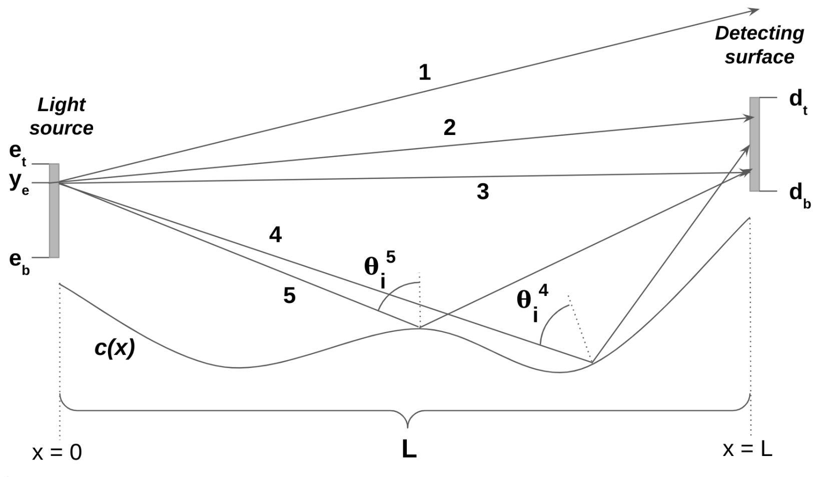


Figure 2.4: Five light rays emitted from the point y_e at the light source.

According to Lambert's cosine law (see section 2.1), the intensity is proportional to the cosine of the angle between the light source and the surface detecting the light. As illustrated to the right in figure 2.3, the light is both emitted and detected at an angle. If each ray carries the same intensity I, the emitted intensity from the light source in the direction towards the detecting surface for each ray ℓ is $I\cos(\alpha_e)\cos(\varphi_e^\ell)$ and the detected intensity will be proportional to $\cos(\alpha_d)\cos(\varphi_d^\ell)$. In the example in figure 2.4, the first ray misses the detecting surface completely, the second and third ray hits the detecting surface without hitting the curve below and the fourth and fifth ray hits the detecting surface by reflecting off the curve. The detected intensities of the reflected rays are

$$I_4 = \frac{R_{eff}(\theta_i^4)\cos(\alpha_e)\cos(\alpha_d)\cos(\varphi_e^4)\cos(\varphi_d^4)I}{L^2},$$
$$I_5 = \frac{R_{eff}(\theta_i^5)\cos(\alpha_e)\cos(\alpha_d)\cos(\varphi_e^5)\cos(\varphi_d^5)I}{L^2},$$

where R_{eff} is the effective reflection factor from equation (2.5). Without a curve to reflect off, the total detected intensity would be

$$I_{direct} = I\cos(\alpha_e)\cos(\alpha_d)\frac{\cos(\varphi_e^2)\cos(\varphi_d^2)+\cos(\varphi_e^3)\cos(\varphi_d^3)}{L^2}.$$

Since L is assumed to be much larger than the height of the points y_e and y_d, the emission and detection angles φ_e and φ_d are close to zero, the angles of incidence θ_i are close to 90°. This results in $\cos(\varphi_e) \approx \cos(\varphi_d) \approx 1$ and, when $\cos(\theta_i) \approx 0$ is inserted into the Fresnel equations (2.3) and (2.4), the effective reflection coefficient is found to be close to one:

$$R_p(\theta_i) \approx \left(\frac{0\cdot n_{rel}^2 - \sqrt{n_{rel}^2-1}}{0\cdot n_{rel}^2 + \sqrt{n_{rel}^2-1}}\right)^2 = 1,$$
$$R_s(\theta_i) \approx \left(\frac{0 - \sqrt{n_{rel}^2-1}}{0 + \sqrt{n_{rel}^2-1}}\right)^2 = 1.$$

This simplifies the expressions considerably:

$$I_4 \approx I_5 \approx \frac{I\cos(\alpha_e)\cos(\alpha_d)}{L^2},$$
$$I_{direct} \approx \frac{2I\cos(\alpha_e)\cos(\alpha_d)}{L^2}.$$

These simplifications are applied throughout the entire derivation of the boost factor.

The boost factor is calculated as the total detected intensity divided by the detected intensity if the curve were not present, which in this case is

$$\begin{aligned}\text{boost factor} &= \frac{I_{direct} + I_4 + I_5}{I_{direct}} \\ &\approx \frac{\cos(\alpha_e)\cos(\alpha_d)(2I + I + I)}{2I\cos(\alpha_e)\cos(\alpha_d)} \\ &= 2.\end{aligned}$$

Note that the intensities, length and dependencies on α cancels out. Essentially, the boost concept can be thought of as the total number of rays hitting the detector divided by the number of rays hitting the detector without reflecting off the curve. Since all rays between an emitter-detector pair depend on the the length L and angles α_e and α_d in the same way, they will always cancel out when calculating the boost. Therefore, α_e, α_d and L are left out in the rest of the calculations in this section.

In practice, the light is emitted in the entire 180° range. Therefore, the discrete formulation must be extended to a continuous one. Assume uniformly distributed intensity over the light source and uniform sensitivity distribution along the detecting surface. First, consider only the rays hitting the detector without reflecting off the curve. Then, for each point y_e along the emitter, there exists an angle $\varphi_{direct}(y_e)$, within which all rays hit the detector. Figure 2.5 illustrates the angle $\varphi_{direct}(y_e)$:

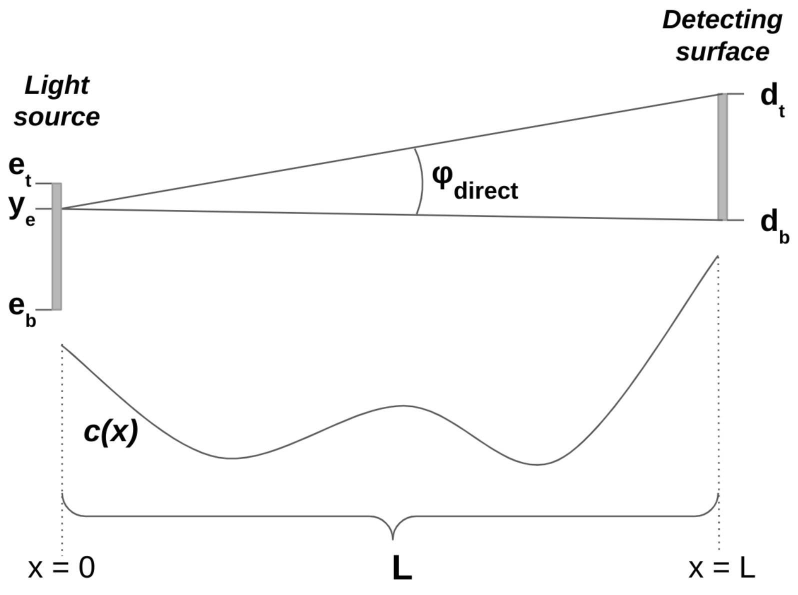


Figure 2.5: Illustration of the direct hit angle φ_{direct}.

Let e_t and e_b be the top and bottom of the light source respectively and d_t and d_b be the top and bottom of the detecting surface. By fixating a point y_e on the light source between the bounds e_t and e_b, $\varphi_{direct}(y_e)$ can be calculated as:

$$\varphi_{direct}(y_e) = \arctan\left(\frac{d_t - y_e}{L}\right) - \arctan\left(\frac{d_b - y_e}{L}\right).$$

The proportion of rays hitting the detector directly is

$$\frac{\varphi_{direct}(y_e)}{\pi} = \frac{1}{\pi}\left[\arctan\left(\frac{d_t - y_e}{L}\right) - \arctan\left(\frac{d_b - y_e}{L}\right)\right].$$

Since the intensity is assumed to be uniformly distributed over the emitting surface and equal for all angles, the total intensity I_{total} in the direction towards the detector is calculated as:

$$I_{total} = \int_{e_b}^{e_t}\int_{-\pi/2}^{\pi/2} I d\beta dy = I\pi(e_t - e_b).$$

The proportion p_{direct} in relation to the total intensity is thus:

$$\begin{aligned}
p_{direct} &= \frac{1}{I_{total}}\int_{e_b}^{e_t}\int_{-\pi/2}^{\pi/2} \frac{I\varphi_{direct}(y_e)}{\pi} d\beta dy_e \\
&= \frac{1}{I\pi(e_t - e_b)}\frac{I}{\pi}\pi\int_{e_b}^{e_t} \varphi_{direct}(y_e) dy_e \\
&= \frac{1}{\pi(e_t - e_b)}\int_{e_b}^{e_t} \arctan\left(\frac{d_t - y_e}{L}\right) - \arctan\left(\frac{d_b - y_e}{L}\right) dy_e.
\end{aligned}$$

The rays hitting the detector by reflecting off the curve will now be examined. Define the indicator function $\iota_d(y_d(\varphi_e, y_e))$ as

$$\iota_d(y_d(\varphi_e, y_e)) = \begin{cases} 1 & \text{if } d_b \le y_d(\varphi_e, y_e) \le d_t \\ 0 & \text{else} \end{cases}$$

where $y_d(\varphi_e, y_e)$ is the position of the ray emitted from elevation y_e with angle φ_e at $x = L$. If the ray is reflected backwards and never reaches $x = L$, then $y_d(\varphi_e, y_e) :=$

∞. The proportion p_{curve} detected after reflecting off the curve in relation to the total intensity can then be calculated as:

$$\begin{aligned} p_{curve} &= \frac{I \int\limits_{e_b}^{e_t} \int\limits_{-\pi/2}^{\pi/2} \iota_d(y_d(\varphi_e, y_e)) d\varphi_e dy_e}{I_{total}} \\ &= \frac{1}{\pi(e_t - e_b)} \int\limits_{e_b}^{e_t} \int\limits_{-\pi/2}^{\pi/2} \iota_d(y_d(\varphi_e, y_e)) d\varphi_e dy_e. \end{aligned} \tag{2.7}$$

Thus, expressions for both p_{curve} and p_{direct} are derived, and equation (2.6) can now be used to calculate the boost factor for a given curve.

2.1.4 Simulation

To calculate the boost factor for a curve $c(x)$ is to solve equation (2.6). The intensity proportion p_{direct} has a closed form solution, whereas p_{curve} must be approximated by simulation. The simulation depends on the ability of finding the normal to $c(x)$, hence a necessary assumption is that $c(x)$ is differentiable. Consider a discretised version of the integral (2.7). Let Y be the set of vertex points along the emitting surface and Φ be the set of sampled angles. Then p_{curve} can be approximated as:

$$p_{curve} \approx \frac{1}{|Y||\Phi|} \sum_{y_e \in Y} \sum_{\varphi_e \in \Phi} \iota_d(y_d(y_e, \varphi_e)).$$

To improve the resolution, only rays reflecting off the curve at least once are sampled to get a biased estimation which can be compensated for retroactively. The range of angles to sample rays from is easily calculated as

$$\varphi_{curve} = \begin{cases} \arctan\left(\frac{L}{|y - c(L)|}\right) & \text{if } y \geq c(L) \\ \frac{\pi}{2} + \arctan\left(\frac{L}{|y - c(L)|}\right) & \text{else} \end{cases}$$

and is illustrated in figure 2.6.

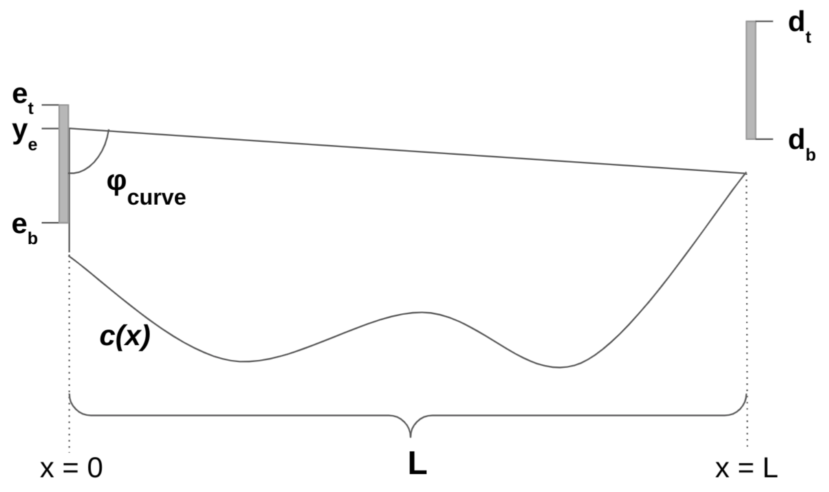


Figure 2.6: Illustration of the range of angles for which light rays emitted from a point y_e will intersect the curve $c(x)$.

Denote the biased estimation as p^b_{curve}. The compensation is then carried out as:

$$p_{curve} = \frac{\varphi_{curve}}{\pi} p^b_{curve}.$$

The simulation algorithm is summarised with the pseudo code in algorithm 1.

```
Input: Rays from light source
while rays are still reflecting off material do
    find points $i$ on curve where the rays will reflect
    for all remaining rays do
        if $i$ is between emitter and detector then
            calculate reflections by finding the normal of the curve at $i$
        else
            remove ray from remaining rays
        end
    end
end
count the number of rays hitting the detector
```

Algorithm 1: Algorithm for simulation of boost gain

As a future reference for the reader, the boost factor is two when a flat curve is present and the angles of incidence onto the curve are large. This can reasoned about in the following way: Consider a light source and a perfectly flat curve ($\hat{\boldsymbol{n}} = [0, 1]^\top \ \forall x$). From the light source to any point on the detecting surface, there are exactly two paths: the direct path and the path with one reflection. Assuming that the light source has full field of view of 180° towards the detecting surface, the path with one reflection off the flat curve will always exist. Therefore, the number of paths doubles compared to when the curve is not present. Thus, when the angles of incidence are large, the detected intensity at the detecting surface doubles. Figure 1.1 in the introduction displays the light rays and simulated boost factor for a flat curve.

2.2 Theoretical foundation of the investigated methods

The problem this thesis aims to solve is to find the surface that causes the boost of each scanline over a surface.

One of the investigated methods updates one or a few scanlines at a time with a gradient descent scheme. To avoid using finite difference estimation of the simulation, the gradient is approximated using a neural network and backpropagation.

The other investigated methods use all scanlines at the same time. Since there are a lot of scanlines, the dimensionality has to be reduced. A few different ways of reducing the data are investigated, for example by describing the data as a linear expansion of different bases and by using Convolutional Neural Network (CNN) based Encoder-Decoder architectures. Thereafter, a way of relating the reduced boost data to the reduced surface data it needed. For this, similarity transformations and neural networks are investigated.

In this section, the theoretical foundation needed for everything mentioned above is presented in a short and concise manner.

2.2.1 Helmholtz equation in a rectangle

One of the bases used to reduce the dimensionality of the surface data consists of the eigenfunctions to Helmholtz equation on a rectangle. Therefore, let $S(x,y)$ denote a twice continuously differentiable surface function in the rectangle $\Omega = \{(x,y) \mid 0 \leq x \leq L,\ 0 \leq y \leq H\}$. Furthermore let $S(x,y)$ be zero along the boundary of the rectangle. The eigenfunction-eigenvalue pairs to the negative Laplacian is found by solving the Helmholtz equation [17] (p. 313-315):

$$\begin{aligned}
&-\Delta S(x,y) = \lambda S(x,y), \quad S \in \mathcal{D} \qquad (2.8)\\
&\Omega = \{(x,y) \mid 0 \leq x \leq L,\ 0 \leq y \leq H\},\\
&\mathcal{D} = \{S \in \mathcal{C}^2(\Omega) \mid S(0,y) = S(L,y) = S(x,0) = S(x,H) = 0\}.
\end{aligned}$$

Solving this problem, the following eigenfunctions $s_{m,n}$ with corresponding eigenvalues $\lambda_{m,n}$ are obtained:

$$\begin{aligned} s_{m,n}(x,y) &= -sin\left(\frac{m\pi}{L}x\right) sin\left(\frac{n\pi}{H}y\right). \\ \lambda_{m,n} &= \pi^2\left[\left(\frac{m}{L}\right)^2 + \left(\frac{n}{H}\right)^2\right]. \\ m,n &\in \mathbb{Z}^+. \end{aligned}$$

Note that this family of eigenfunctions are not uniquely determined. For example, the eigenfunctions $\widetilde{s}_{m,n}(x,y) = asin\left(\frac{m\pi}{L}x\right) sin\left(\frac{n\pi}{H}y\right)$ also satisfies the problem formulation 2.8. The choice of using $a = -1$ as scaling is to make the eigenfunction $s_{1,1}(x,y)$ bowl shaped. For the full derivation, see appendix A.1.

2.2.2 Principal Component Analysis

To reduce the dimension of data while preserving as much information as possible, Principal Component Analysis (PCA) can be used to find the most prominent features. The dimensionality can then be reduced by only using some number of these features, or *principal components*, to describe the data. A simple example of PCA on two dimensional data can be seen in figure 2.7.

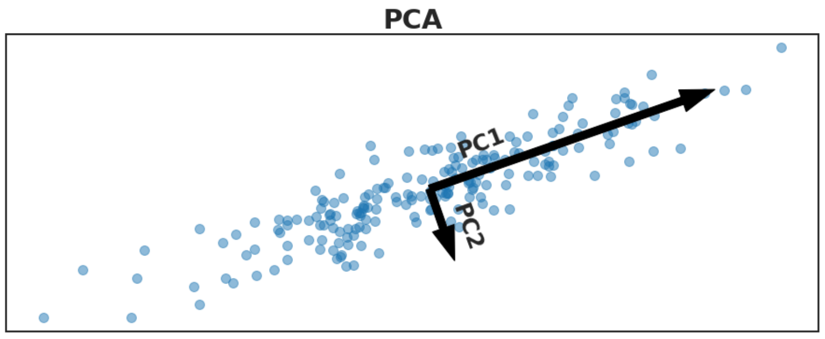


Figure 2.7: Example of Principal component analysis. The most prominent feature is found in the PC1 direction and the less prominent in the PC2 direction, which is orthogonal to PC1.

Consider a data set with vectors $\{\boldsymbol{x}_i\}_{i=1}^m$. To find the principal components, the covariance matrix $\boldsymbol{\Sigma}$ of the data set is constructed. The covariance matrix is symmetric and positive semi-definite, hence its eigenvalue factorisation can be written as

$$\boldsymbol{\Sigma} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top.$$

The diagonal matrix $\boldsymbol{\Lambda}$ contains the (non-negative) eigenvalues sorted in descending order, and the orthogonal matrix $\boldsymbol{V}$ holds the corresponding eigenvectors as column vectors. These eigenvectors are the principal components (orthogonal features) of the data set, where the eigenvector corresponding to the largest eigenvalue accounts for most of the variance in the data set, the eigenvector corresponding to the second eigenvalue accounts for the second largest amount of variance and so on. For a full derivation as well as a more efficient method for calculating the principal components, see appendix A.2.

To determine how many principal components that should be used to reduce the dimensionality of the data, an *energy* measurement is used. Assume that the covariance matrix has n eigenvalues. The energy of the covariance matrix is defined as $E = \sum_{j=1}^{n} \lambda_j$, where $\{\lambda_j\}_{j=1}^{n}$ are the eigenvalues of the covariance matrix. To calculate the number of components needed to preserve a certain proportion of the total energy, assume that the eigenvalues are sorted in descending order and let $E_k = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{n} \lambda_j}$. The number of features needed is the smallest number k such that $E_k / E \geq p$, where p is the proportion of energy to preserve.

2.2.3 Similarity transformations

A combination of translation, reflection, rotation and/or scaling is called a similarity transformation [9]. An example of a similarity transformation is illustrated in figure 2.8.

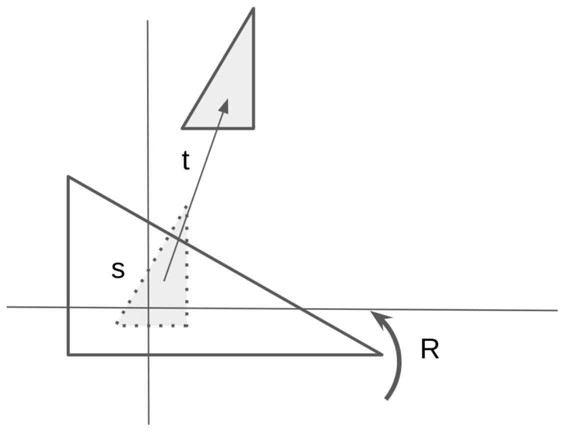


Figure 2.8: Illustration of a similarity transformation. The triangle is rotated 90° anticlockwise, translated up to the right and shrunken.

The transformation matrix of a similarity transformation has the following appearance:

$$\boldsymbol{T} = \begin{pmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{pmatrix},$$

where $\boldsymbol{R}$ is a rotation matrix, $\boldsymbol{t}$ is a translation vector and s is a positive scalar. Let $\{\boldsymbol{x}_i\}_{i=1}^N$ and $\{\boldsymbol{y}_i\}_{i=1}^N$ be corresponding sets of points. If the optimal similarity transformation $\boldsymbol{T}^*$ between the sets is known, it can be used to estimate where the corresponding point $\boldsymbol{y}_{new}$ would be for a new point $\boldsymbol{x}_{new}$, and is obtained by applying the transformation, i.e:

$$\begin{bmatrix} \boldsymbol{y}_{new} \\ 1 \end{bmatrix} = \boldsymbol{T}^* \begin{bmatrix} \boldsymbol{x}_{new} \\ 1 \end{bmatrix}.$$

When a similarity transformation is used to align two sets of corresponding points, the optimal transformation depends on the metric, but a common choice is to minimise the Euclidean norm between the two sets of points. The optimal similarity transform with respect to the Euclidean norm can be found using Umeyama's algorithm, which is described in appendix A.3.

2.2.4 CNN based Encoder-Decoder

As seen in section 2.2.2, PCA finds the hyperplane best describing the data in the selected number of dimensions. Thus, dimensionality reduction with PCA works well if the data can be described by linear patterns, but falls short when the data exhibits non-linear patterns. An example of this can be seen in figure 2.9.

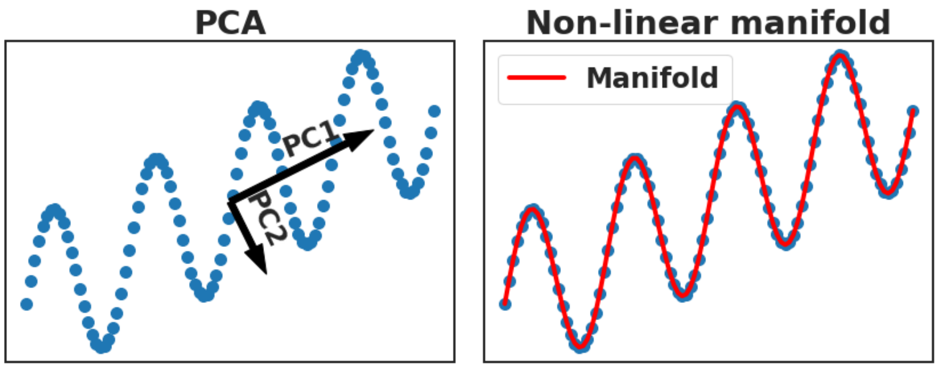


Figure 2.9: Illustration of when ordinary principal component analysis fails to capture non-linear features (left). Alternative dimension reduction methods could however take non-linearity into account and find non-linear manifolds.

The principal components fail to capture the underlying sinusoid pattern of the data. Therefore, a CNN-based Encoder-Decoder architecture can be used to reduce the dimensionality of the data instead of PCA. An Encoder-Decoder network with linear activations trained to generate itself, known as an autoencoder, can essentially be viewed as a PCA without the orthogonality constraints on the components [13].

With non-linear activations, the Encoder-Decoder network can learn non-linear manifolds, as illustrated in figure 2.9.

An illustration of a typical CNN-based encoder and decoder can be seen in figure 2.10.

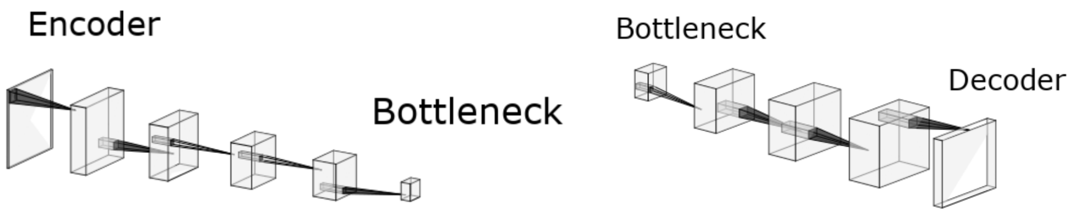


Figure 2.10: Illustration of a typical encoder-decoder structure [8].

When the output of the encoder has smaller dimensions than the input, an information bottleneck is created. This forces the encoder to learn to represent the input data in fewer dimensions, essentially learning to recognise important features of the data[2]. If the decoder then learns to reconstruct the input only using the information in the bottleneck, the input is successfully encoded in a lower dimension.

2.2.5 Neural network estimation of gradient using backpropagation

A neural network trained to approximate a function can also be used to estimate the partial derivatives of its inputs. To do this, it is necessary to find how the input parameters affects the loss function. This is done using the backpropagation algorithm [15]. Let L be the number of layers in the network and (ℓ) denote the ℓ:th layer. Furthermore let $f^{(\ell)}(\cdot)$, $\boldsymbol{z}^{(\ell)}$, $\boldsymbol{a}^{(\ell)}$ and $\boldsymbol{W}^{(\ell)}$ denote the activation function, pre-activation values, post-activation values and the weights respectively at the ℓ:th layer and let $\mathcal{L}(y, \hat{y})$ be the loss function. The gradient of the loss function with respect to the input $\boldsymbol{p}$ of the network is then calculated as

$$\frac{d\mathcal{L}}{d\boldsymbol{p}} = \left[\prod_{\ell=1}^{L} \left(\frac{df^{(\ell)}(\boldsymbol{z}^{(\ell)})}{d\boldsymbol{z}^{(\ell)}} \boldsymbol{W}^{(\ell)} \right)^{\top} \right] \frac{d\mathcal{L}}{d\boldsymbol{a}^{(L)}}. \tag{2.9}$$

For the full derivation of the gradient, as well as clarification of the notation, see appendix A.4.

Chapter 3

Data Collection

3.1 Collection of real data

To test the methods on real data, a setup is constructed.

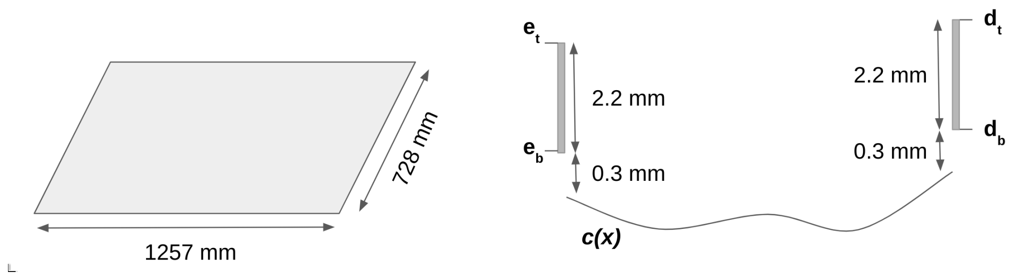


Figure 3.1: Illustration of the setup. The left figure shows the dimensions of the panel and the right how the emitters and detectors are placed in relation to the panel, i.e at an elevation of 0.3 mm above the edge points.

The setup, illustrated in figure 3.1, consists of a thin panel with dimensions 1257 by 728 mm. The panel is equipped with a total of 144 emitters and 144 detectors, 48 on each long side and 24 on each short side, placed at an elevation of 0.3 mm above the panel. The width of both the emitting and detecting surface is 2.2 mm.

The setup is constructed such that for all pairs of emitters and detectors:

1. The difference between the largest and smallest value of the curve $c(x)$ between them is small in relation to its length L.
2. The emitter and detector are at similar heights.
3. The emitter and detector are not elevated much above the curve.

4. The widths of the emitting and detecting surfaces are small.

Due to these properties, the light rays hitting the detecting surface by reflecting off the curve have had large angles of incidence, which matches the assumptions made in the boost factor derivation in section 2.1.3.

Two different types of data are collected from the setup. The first is the shape of the panel surface, which is measured using an ultrasonic sensor. To obtain different surfaces, the panel is manually deformed. The sensor has a resolution exceeding 0.069 mm, which makes it suitable for measurements of point pressure deformations. Measurements are made in a grid of 25×15 equidistant points along the surface.

The second type of data is a recording of detected light when the panel is deformed, without any exterior interaction or interference. This data is simply the detected signal strengths and not the boost factor for each scanline. Since the boost factor of a perfectly flat surface is two, the boost factor for each signal can be computed by comparing the recorded signal to a reference obtained from a flat surface:

$$\text{Surface boost factor} = \frac{2 \cdot \text{Recorded signal}}{\text{Recording of flat signal}}.$$

To record signals from a flat surface, the surface is manually deformed until the panel is approximately flat. Figure 3.2 illustrates an attempt of making the panel flat:

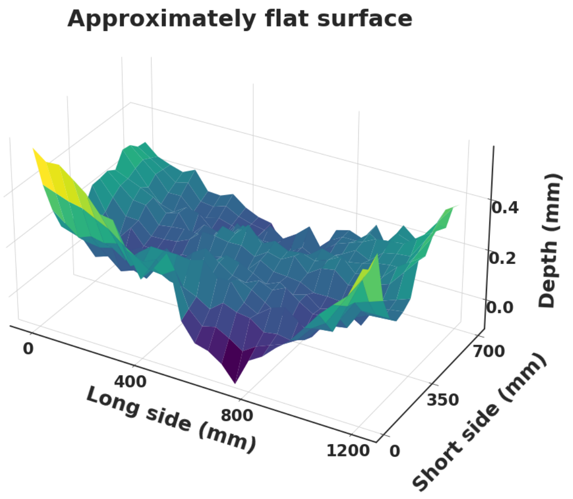


Figure 3.2: Illustration of the approximated flat panel created by manually deformation.

The panel deviates about 0.2 mm from the mean at the most extreme points.

3.2 Data generation

Collection of real data is time consuming due to the measurements of the shape of the surface being slow. The possibility to generate reasonable surfaces is therefore of great interest. The boost simulator can then be used to estimate the boost for each surface. The generated surfaces need to reflect reality in that the surfaces should be possible to create by applying a reasonable amount of force to the panel.

The generated surfaces are based on twelve handpicked basis surfaces, which are displayed in figure 3.3:

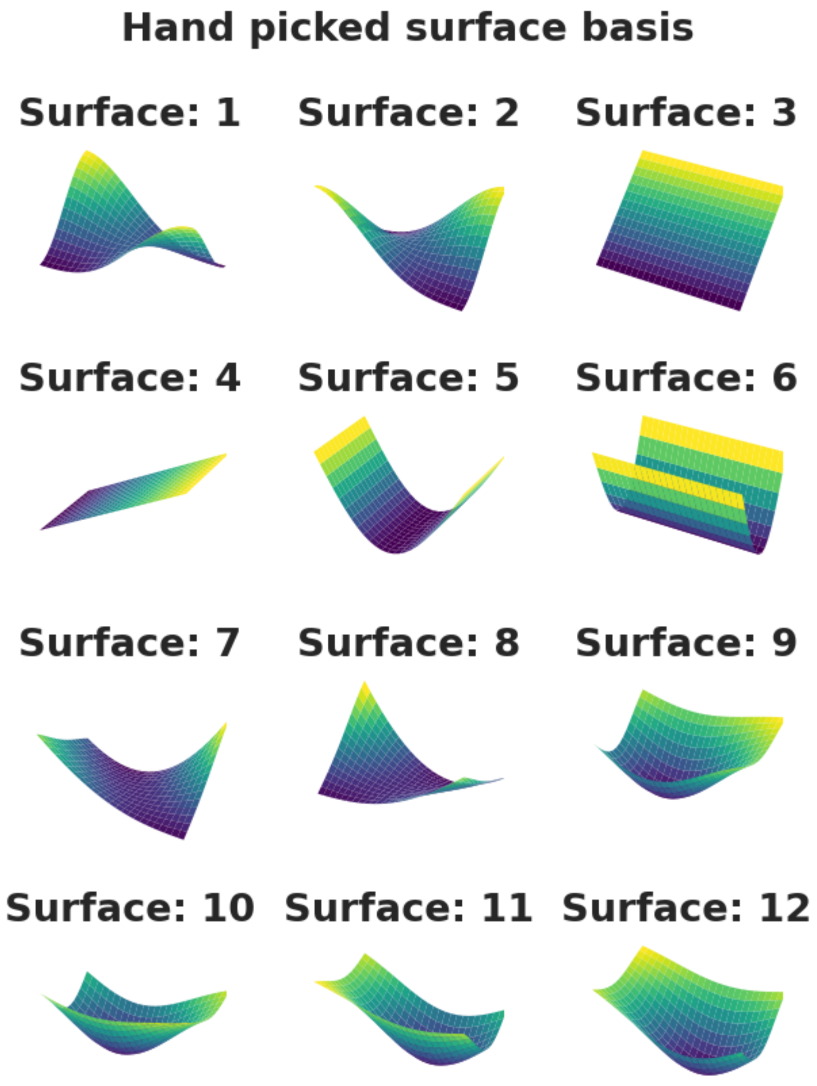


Figure 3.3: Illustration of the twelve handpicked basis surfaces.

In essence, the surfaces are generated as a linear combination of the basis surfaces in figure 3.3, but with some additional intermediate steps adding more variation and smoothing the surfaces to better resemble a the measured surfaces. The full data generation algorithm can be found in appendix C.

By using this algorithm to generate reasonable surfaces combined with the simulation to generate the corresponding boost matrices, a large amount of training data can be generated. This opens up the possibility to investigate methods relying heavily on training data, which would not be reasonable if all training data had to be measured and recorded from the setup. Two examples of surfaces generated by the algorithm are illustrated in figure 3.4.

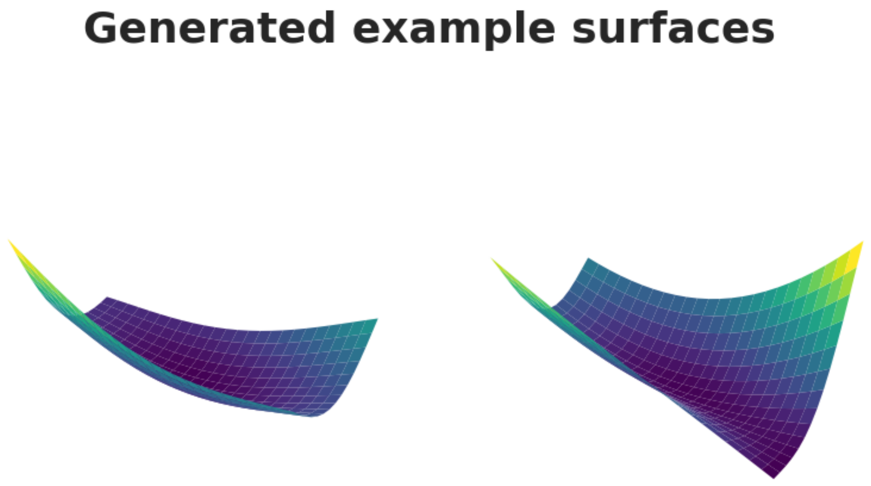


Figure 3.4: Two examples of generated surfaces.

3.3 Data exploration

The aim of this section is to take a look at the data, investigate how the boost depends on the underlying curve and to make a brief comparison between the simulated and measured data. The surfaces are intuitive and easily illustrated. An example surface is displayed in figure 3.5.

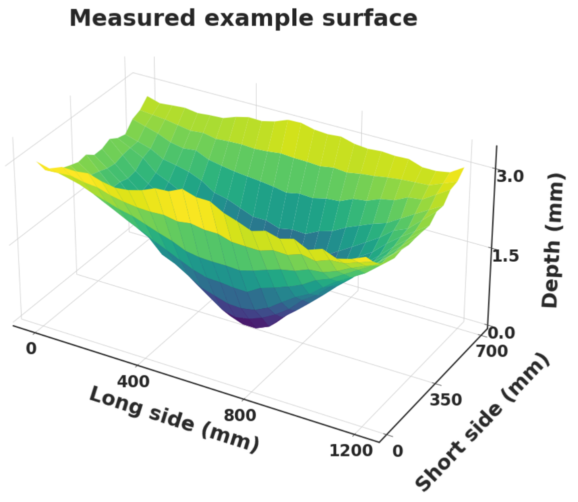


Figure 3.5: Illustration of a 'bowl' shaped surface.

Note the dimensions of the surface. The length and width of the surface is much larger than the depth, which is the case for all measured and generated surfaces.

The boost factors of all scanlines, i.e all emitter-detector pairs, over a surface will be illustrated as a *boost matrix*. Let the components around the periphery of the surface be enumerated such that both emitter and detector number one is in the lower left corner and increases in the anticlockwise direction around the edge of the surface. The intensity measured by the d:th detector from the e:th emitter is denoted $I_{e,d}$. Consider the geometry shown in figure 3.6:

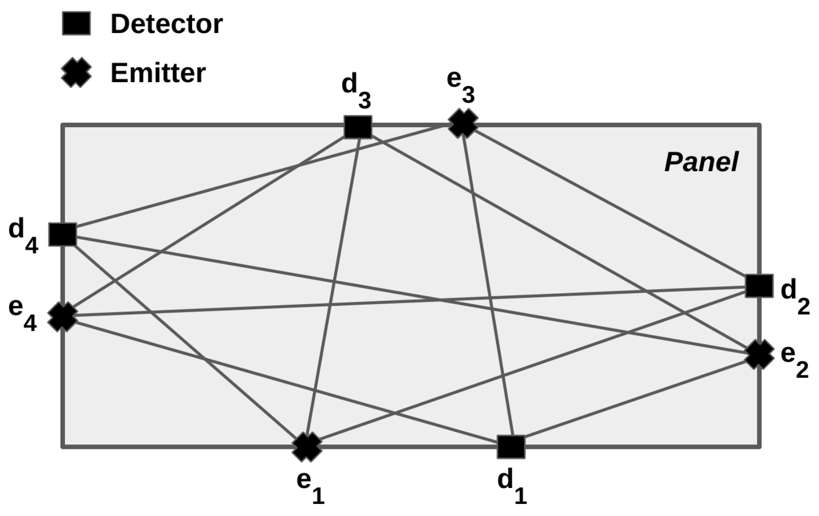


Figure 3.6: Scanline geometry with four emitters and four detectors.

The scanline intensities $I_{e,d}$ from each emitter to each detector makes up a *signal matrix* with the following appearance:

$$\text{signal matrix} = \begin{bmatrix} 0 & I_{1,2} & I_{1,3} & I_{1,4} \\ I_{2,1} & 0 & I_{2,3} & I_{2,4} \\ I_{3,1} & I_{3,2} & 0 & I_{3,4} \\ I_{4,1} & I_{4,2} & I_{4,3} & 0 \end{bmatrix}$$

The diagonal elements are zero since the angle from an emitter to a detector on the same edge as itself is 90°, which according to Lambert's law results in no light reaching the detector. To obtain the boost matrix, the signal matrix is divided elementwise by the corresponding elements in the signal matrix for the same geometry but without any surface for the light to reflect off. Since the real setup has 144 emitters and 144 detectors, the full size boost matrices will have dimensions (144, 144). Figure 3.7 illustrates the boost matrix for the surface in figure 3.5:

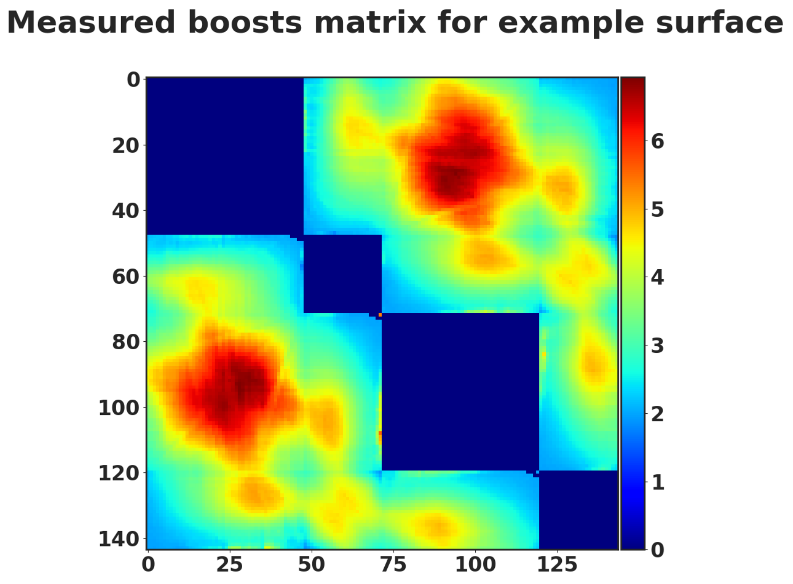


Figure 3.7: The measured boost matrix for the example surface in figure 3.5.

The boost matrix is interpreted in the following way:

1. The component ordering starts in the lower left of the surface and then moves anticlockwise.
2. The rows correspond to the different boosts between one specific emitter and all detectors. So for example row zero corresponds to the boosts from the emitter in the lower left corner to all the detectors.

3. The columns correspond to the boosts between all emitters and one specific detector.
4. The dark blue regions are where the components lie on the same edge.
5. Example: The region where the rows are 71-119 and the columns are 0-47 (the box with red region in the lower left) corresponds to the scanlines going from the long side at the top to the long side at the bottom of the surface, and the off-diagonal of the box corresponds to the completely vertical scanlines.

To get a better understanding of why a boost matrix for some surface looks the way it does, it is important to look at how the boost for each individual scanline depends on the surface. The number of rays reaching the detecting surface increases as the curve becomes more parabolic. Consider a curve $c(x) = a\left(x - \frac{L}{2}\right)^2$ with length L. The curve is a parabola centred at $x = \frac{L}{2}$. By varying a, the curvature of the parabola is varied. The hypothesis is that as a increases, the boost should also increase. Figure 3.8 illustrates the relation between the coefficient a and the boost.

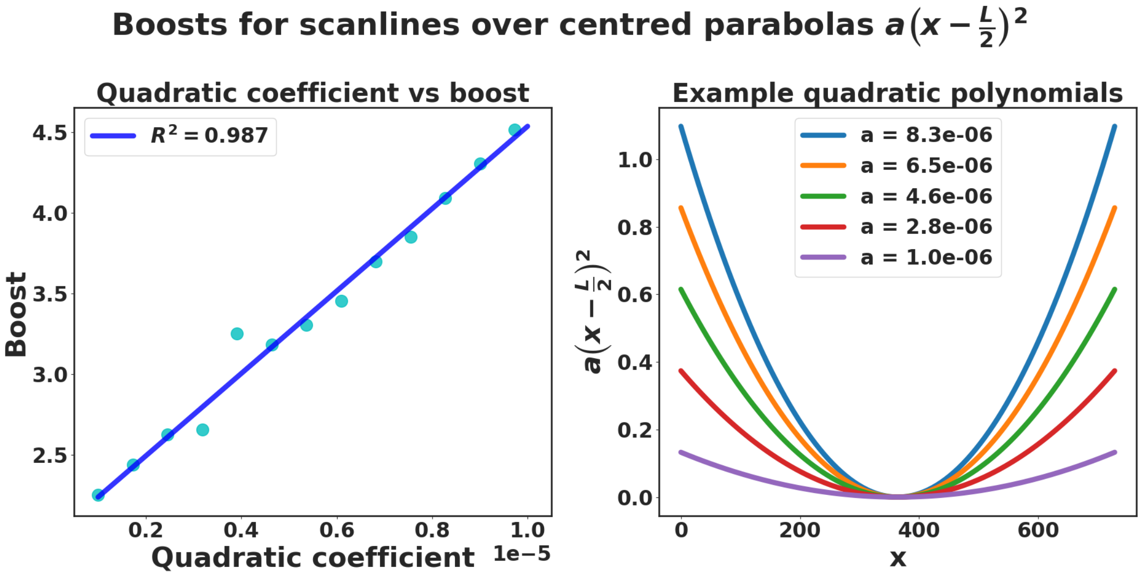


Figure 3.8: Relation between curvature of centred parabola and boost.

It appears from figure 3.8 that the boost depends linearly on the coefficient a in the interval $[10^{-6}, 10^{-5}]$, which is in the range of the quadratic coefficient for the deformations on the real surface. Next, the effect of the length of the curve on the boost is examined. For the centred parabolas $c(x) = \left(x - \frac{L}{2}\right)^2$, this is easy to answer. Since the boost depends linearly on the curvature a of the centred parabolas, the boost will also depend on the length, since a depends on the length. On the edges, $c(x)$ equals some elevation c_0, making it possible to construct the following equation:

$$c_0 = a(0 - \frac{L}{2})^2 \iff a = 4\frac{c_0}{L^2}.$$

Hence, given a centred parabola with some fixed elevation at the edges, the boost decreases proportionally to the square of its length. Even though the curves are almost never perfectly centred parabolas in practice, consider 'bowl' shaped surface functions, for example the measured shape in figure 3.5. By extracting one-dimensional curves along these types of shapes, it is possible to compare how the boost depends on the depth of the curve, which should have a similar appearance as the centred parabola relation. To remove the length dependence, consider only vertical curves going from the bottom to the top of the surface. The relation between the depth and the boost for these curves are illustrated in figure 3.9:

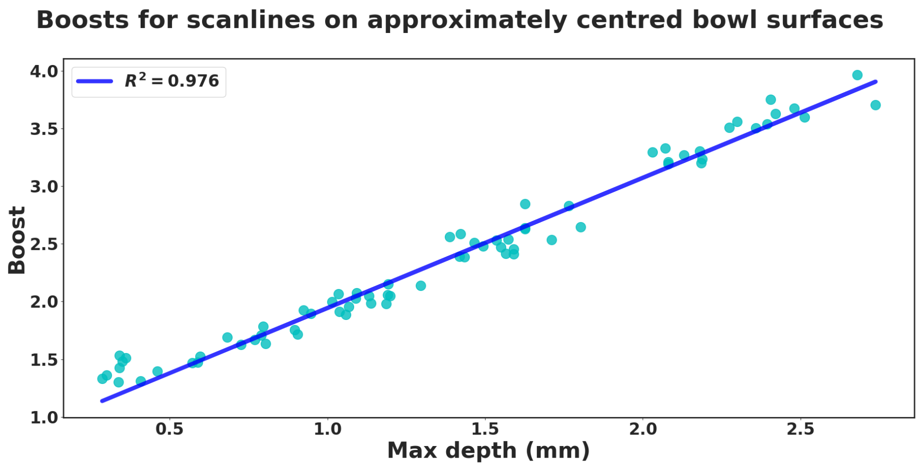


Figure 3.9: Relation between depth and boost for vertically extracted one-dimensional curves where the underlying 2D surfaces are approximately centred 'bowl' surfaces.

It is obvious from figure 3.9 that the max depth of a curve has a large impact on the boost, at least for this family of curves. However, the lowest point will of course not always be close to the the middle of the curve. Thus, the impact of skewness of the curve on the boost has to be investigated. This is done by studying a family of third degree polynomials $c(x)$ with length L fulfilling the following criteria:

1. $c(0) = c(L) = c_0 := 2.$
2. $c(\frac{L}{2} + \delta) = \min_{x \in [0,\, L]} c(x) := -1\,.$
3. $c'(\frac{L}{2} + \delta) = 0.$

As the parameter δ is varied, the lowest point is shifted away from the centre. Due to the symmetry of reflections, a shift to the left should affect the boost in the same way as the a shift to the right, assuming that the magnitude of the shifts are the same. The effect of δ on the boost is displayed in figure 3.10

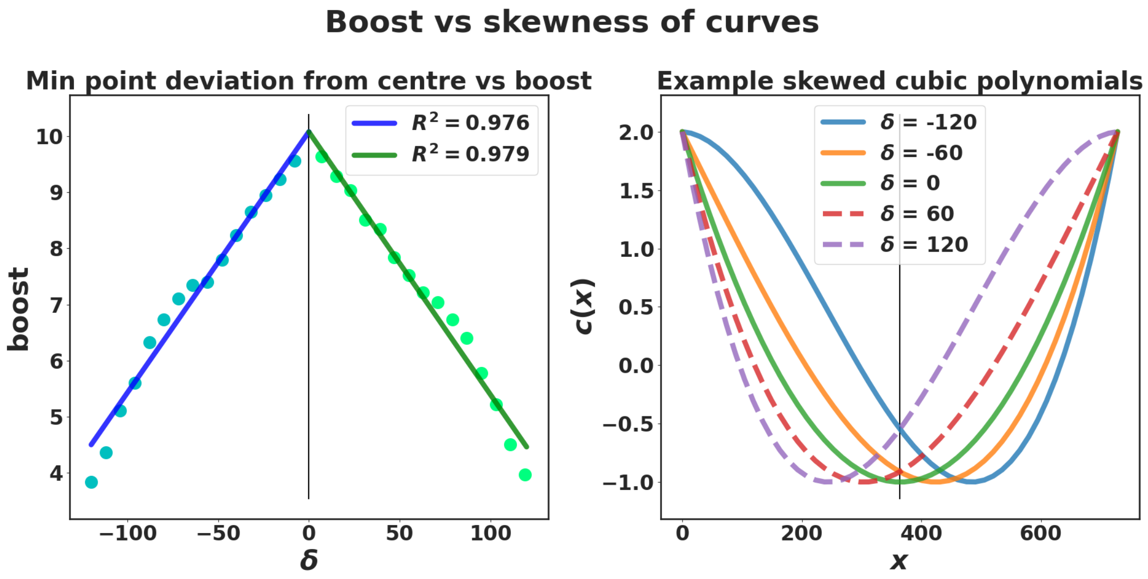


Figure 3.10: Relation between skewness and boost.

The boost decreases proportionally to $|\delta|$, at least for cubic polynomials which fulfil the proposed criteria. Now, as touched on briefly in the analysis of the relation between skewness and boost, there exist a natural symmetry for the boost. Assume that some curve $c(x)$ with length L yields a boost b, then the curve $c(L - x)$ also yields the same boost b. Hence, for a given boost, there exists at least two different curves yielding that boost. An example of this is illustrated in figure 3.11:

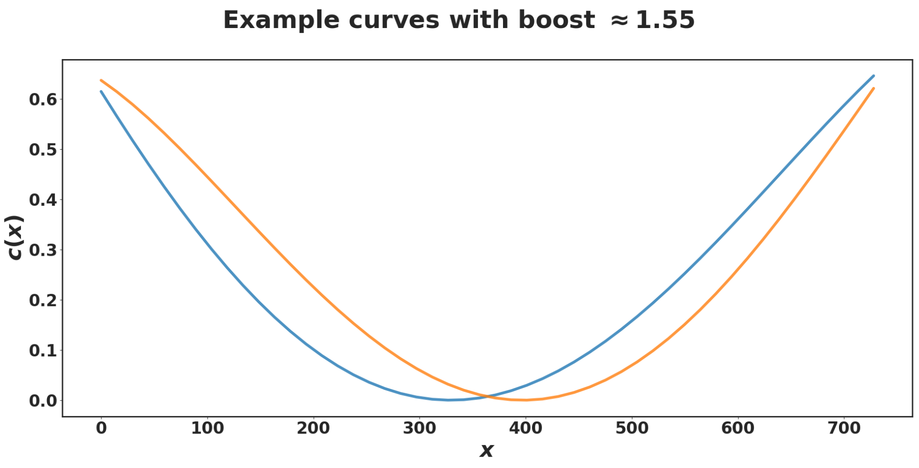


Figure 3.11: Mirrored curves both yielding a boost of 1.55.

However, these are not the only ones, rather there exists lots of possible curves yielding very similar boosts. This can be seen in figure 3.12, where all curves yield a boost in the interval $[1.55 \pm 0.005]$:

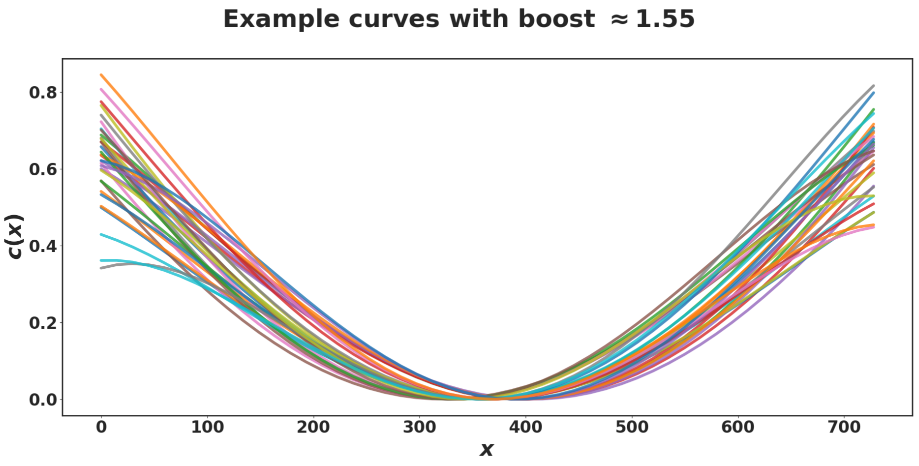


Figure 3.12: Several curves with very similar boosts.

At least a cubic polynomial is necessary to approximate a curve since the boost is dependent on both its curvature and skewness. However, since the real data is collected on a large panel, the surface may contain for example two modes or some other property making the cubic approximations insufficient, which can be seen in figure 3.13.

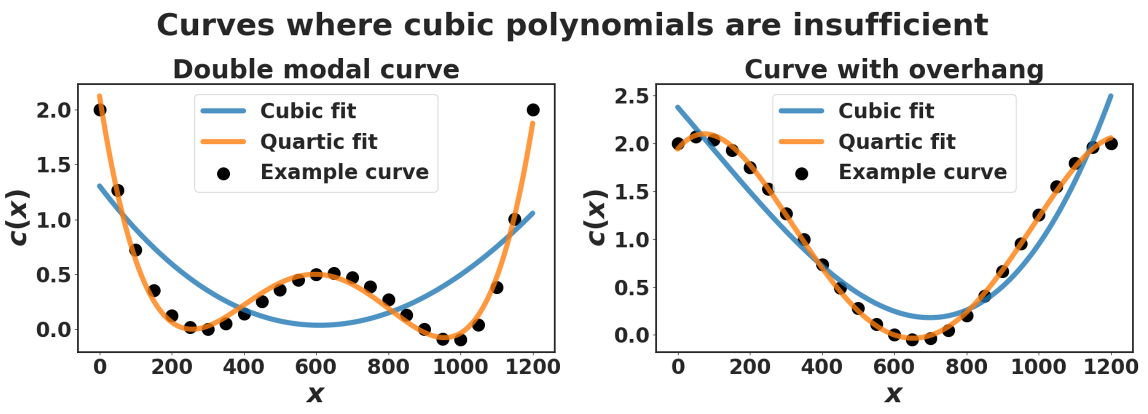


Figure 3.13: Examples where cubic polynomial underfits, left: double modal curve, right: curve with overhang at the edges.

Therefore, using cubic polynomials is not sufficient. However, a too high degree of the polynomial might cause overfitting, as illustrated in figure 3.14:

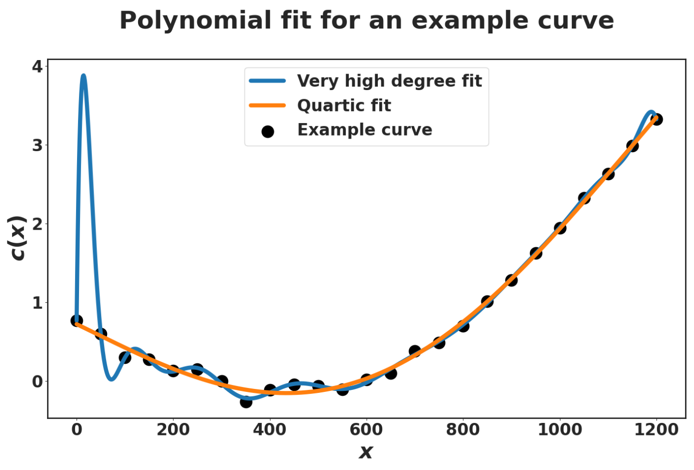


Figure 3.14: Overfitted polynomial.

Considering this trade-off, a fourth degree approximation is deemed a reasonable choice.

Using the information derived above, the appearance of the boost matrix in figure 3.7 can be explained. First, since the example surface in figure 3.5 is quite parabolic, the maximum boost should be found for the scanline travelling over the centre at the shortest possible distance. This is the case for scanline from one long side to the other over the centre, which is approximately around the elements (23, 95) and (95, 23). Looking at the boost matrix, this is indeed the case. Furthermore, the boost matrix should always be symmetric, since the curve $c(x)$ between emitter-detector pair (i, j) has the same boost as the curve $c(L - x)$ between pair (j, i). Lastly, the measured boost matrix is compared with the simulated boost matrix in figure 3.15

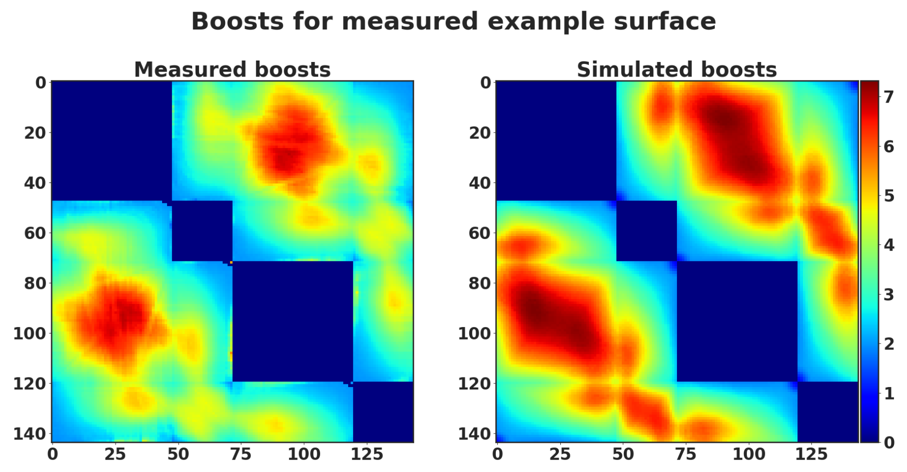


Figure 3.15: Comparison of measured boost matrix (left) and simulated boost matrix (right) for the example surface in figure 3.5.

The boost matrices are similar, but there are also obvious differences. The differences may be caused by several factors, such as:

- Loss of precision in the simulation due to discretisation.
- Simulation performed on polynomial approximations of the real curves.
- The supposedly flat reference panel is not actually perfectly flat.
- The recorded signal contains noise.

Chapter 4

Analysis

Consider a target surface described by the function $S(x,y)$. The problem at hand is to reconstruct a surface described by the function $\widetilde{S}(x,y)$ which is as close to $S(x,y)$ as possible. Hence, the problem can be written as a minimisation problem:

$$\underset{\widetilde{S}(x,y)}{\operatorname{argmin}} \frac{1}{LW} \int_0^W \int_0^L \Big(S(x,y) - \widetilde{S}(x,y) \Big)^2 \, dxdy. \tag{4.1}$$

Since the only thing known about the surface to reconstruct is its boost matrix, the problem has to be rewritten to reconstruct a surface producing a boost matrix as similar to the target as possible.

Let $\boldsymbol{B}$ be the measured boost matrix for the surface $S(x,y)$ and $\mathcal{B}$ be the function mapping a surface to its boost matrix. An alternative least square minimisation problem can then be constructed, where the target function is in terms of the boost matrices instead:

$$\underset{\widetilde{S}(x,y)}{\operatorname{argmin}} \quad \frac{1}{2} \; ||\boldsymbol{B} - \mathcal{B}(\widetilde{S}(x,y))||_2^2 \tag{4.2}$$

The simulation described in section 2.1.4 is considered a sufficiently accurate approximation of the function $\mathcal{B}$. Thus, the problem lies in developing a method capable of finding the target surface using only its boost matrix. However, the relationship between small changes in the surface and the effect on the boost factor of each individual scanline is complex and non-linear, making it difficult to predict how a change in the surface actually affects the overall boost matrix without having to run the full simulation for every modification of the surface.

With these issues in mind, the following methods of reconstructing the target surface are investigated:

Iteratively update a small batch of scanlines:

Simplify the problem by only considering how a change in the surface would affect one or a few scanlines at a time. Since the number of scanlines considered at a certain iteration is small, it is possible to simulate the boost and approximate the gradient for each scanline. A gradient descent algorithm is then used to iteratively find a surface matching the measured boost. Both estimation of the gradient directly from the simulation and by mapping the simulation to a neural network is investigated.

Approximate boost matrices as linear combinations of boost basis matrices:

Investigate the possibility of simply ignoring the non-linearity of the problem and assuming that each boost matrix can be sufficiently well approximated by a linear combination of some boost basis matrices. The weight of each basis in the linear combination is then mapped to weights of basis surfaces corresponding to each boost basis matrix, which are finally used to reconstruct the predicted surface. Assuming that it is possible to ignore the non-linearity of the problem, the first task in this method lies in finding reasonable basis sets that can be used to approximate the boost matrices and surfaces. For this, three different approaches of basis selection is investigated:

- Manual surface basis selection.
- Using eigenfunctions from the Helmholtz equation in a rectangle.
- Using principal components.

The second task is to find a reasonable way of relating the boost and surface basis weights. For this, geometric transformations and neural networks are investigated.

Use CNN-based Encoder-Decoder architectures to encode the boost matrix and decode the surface:

The possibility of using all information, including the non-linear part, of the problem is investigated by training CNN-based Encoder-Decoder architectures to reconstruct the surface from the boost matrix.

4.1 Update individual scanlines iteratively

Since problem 4.2 is large and difficult on its own, the main idea of this method is to split it up. Instead of trying to find the entire surface directly, a possibility would be to make an initial guess $\widetilde{S}(x,y)$, look at individual scanlines and update $\widetilde{S}(x,y)$ iteratively. The surface $S(x,y)$ is assumed to be sufficiently smooth and can be accurately approximated by a polynomial surface of degree p. The initial guess can then be made as an arbitrary p:th degree polynomial surface. Let P be a p:th degree polynomial surface basis, i.e:

$$P = [1,\ x,\ y,\ x^2,\ xy,\ y^2, \ldots,\ x^p, \ldots,\ x^k y^{p-k} \ldots y^p]^\top.$$

The number of coefficient for describing a p:th degree surface is $q = \frac{(p+1)(p+2)}{2}$. Let $\boldsymbol{c} = [c_0,\ c_1,\ \ldots\ c_{q-1}]^\top$, then $\widetilde{S}(x,y)$ can be written as $\widetilde{S}(x,y) = \boldsymbol{c}^\top P$. The main advantage for this assumption is that the parameterised line $y = kx + m$ from an emitter to a detector will yield a one-dimensional curve with degree p along the surface, i.e

$$\widetilde{S}(x,\ y = kx + m) = \sum_{k=0}^{p} a_k x^k = \boldsymbol{a}^\top \boldsymbol{x}, \quad \boldsymbol{x} = \left[1,\ x,\ x^2,\ \ldots,\ x^{p-1},\ x^p\right]^\top$$

Let b_m be the measured boost along a scanline and $b_p(\boldsymbol{a})$ be the estimated boost for the corresponding curve. The coefficients $\boldsymbol{a}$ can then be updated using a gradient descent scheme [1]. The gradient is calculated from the loss function

$$\mathcal{L}(\boldsymbol{a}) = [b_m - b_p(\boldsymbol{a})]^2. \tag{4.3}$$

However, a continuity constraint exists for $\widetilde{S}(x,y)$. One way to enforce the constraint would be to use the following scheme:

1. Make an initial guess of $\widetilde{S}(x,y)$.
2. Evaluate $\widetilde{S}(x,y)$ on a mesh to get a scatter of points.
3. Choose a batch of scanlines randomly.
4. Extract one-dimensional curves by parameterising along each scanline in the batch.
5. Update the coefficients for the one-dimensional curves using gradient descent on the loss functions.
6. Scatter points along each updated curve.
7. Perform a least square polynomial surface fit using the scatter points from the mesh and the updated curve.
8. Set $\widetilde{S}(x,y)$ to be the new fitted curve and repeat 2 to 8 until a termination criterion is met.

Two main issues have not yet been touched upon. The first is how to make an initial guess. The primary issue with initial guesses for this problem is, as seen in the figure 3.12 from the section 3.3, the amount of possible curves yielding the same boost.

Therefore, it can not be guaranteed that the gradients actually modify the individual curves in the correct way. Thus, the method may be sensitive to initial guesses and may only be suitable to refine predictions from other methods. The second issue concerns the calculation of the gradient. The most straightforward method is to use a finite difference approximation of the boost function. However, due to the nature of the setup and that the boost function is a simulation, the gradient is not particularly kind. The partial derivatives for $b_p(\boldsymbol{a})$ (of a fourth degree polynomial) are illustrated in figure 4.1.

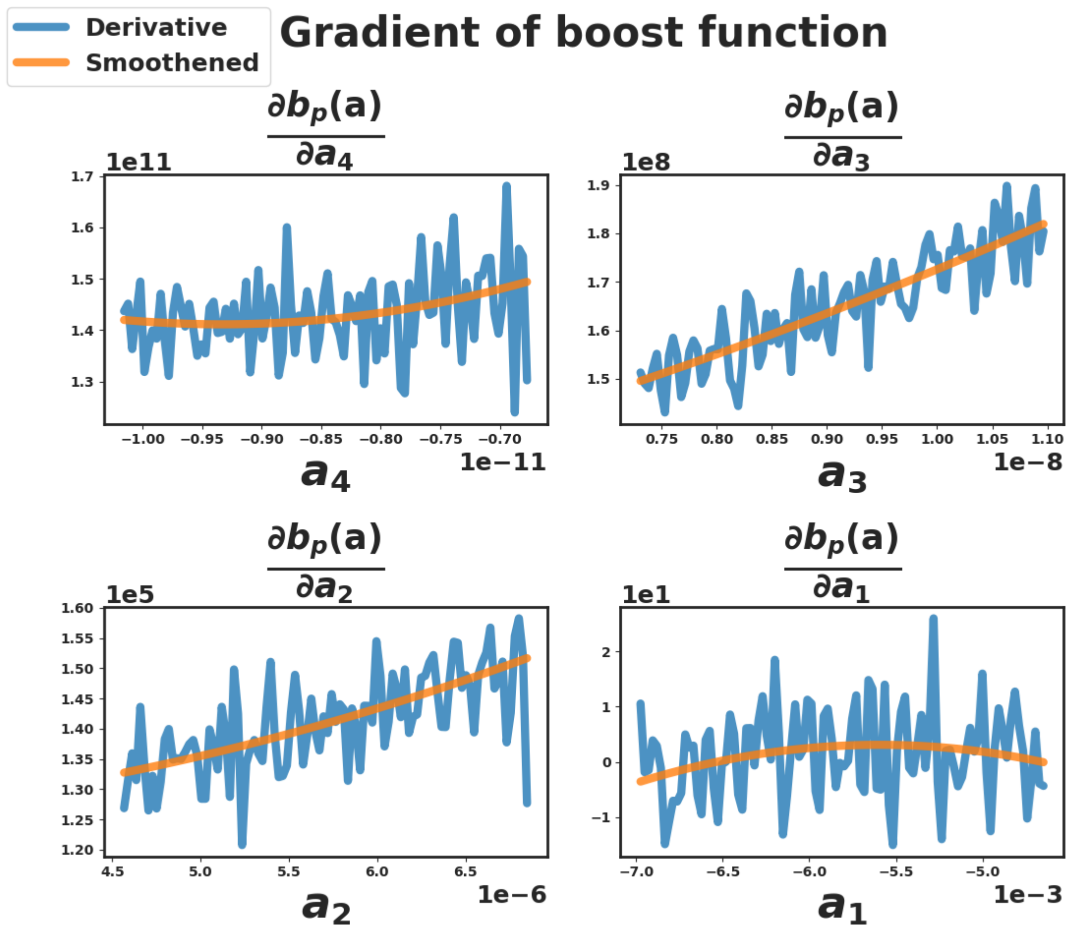


Figure 4.1: Illustration of partial derivatives of the boost function for fourth degree polynomials. The constant term is excluded.

Since the simulation is constructed such that the emitter and detector always have an offset of 0.3 mm from the curve at $x = 0$ and $x = L$, the constant term of the polynomial only shifts the curve and emitters in space without affecting the resulting boost. Therefore, the partial derivative w.r.t the constant term of the polynomial is constant and not included in figure 4.1.

The partial derivatives are very ragged and their sizes suggests the need of some kind of scaling to make the method numerically stable. Furthermore, calculating gradients in this way is expensive. Instead, the simulation is mapped to a neural network and the gradient approximated by using the backpropagation algorithm.

Consider a network that approximates $b_p(\boldsymbol{a})$. The network has p input nodes, $p-1$ for the coefficients $\boldsymbol{a}$, and one for the length of the scanline. The gradient of the loss function w.r.t the input p of the network is calculated using equation (2.9) and is used as an approximated gradient direction. The architecture of the network can be found in appendix B.1.

4.2 Linear basis expansion and space alignment

The idea behind this method is to find basis surfaces and boost basis matrices, then use linear basis expansions to express surfaces with corresponding boost matrices as linear combination of their respective basis. The weights are then used as a training set to estimate a map from the weights for the linear basis expansion of a boost matrix to the weights describing the surface function. To make the process of finding the weights for the linear basis expansion of a surface function easier, consider $\vec{S}$ which is the vectorised (transformed into a column vector [10]) surface matrix constructed by evaluating the surface function at the points (x_i, y_j) where

$$x_i = \frac{i}{m}L,\ i \in \{0, 1, \dots, m\}, \quad y_j = \frac{j}{n}W,\ j \in \{0, 1, \dots, n\}.$$

Let $\{\vec{S}_i\}_{i=1}^{M}$ be the surface basis which $\vec{S}$ is linearly expanded by. By letting $\boldsymbol{S} = [\vec{S}_1,\ \vec{S}_2,\ \dots,\ \vec{S}_M]$, $\vec{S}$ can be written as:

$$\vec{S} \approx \boldsymbol{S}\boldsymbol{w}_{surface}. \tag{4.4}$$

Similarly, the vectorised boost matrix $\vec{B}$ with basis vectors $\{\vec{B}\}_{j=1}^{N}$, can be written as:

$$\vec{B} \approx \boldsymbol{B}\boldsymbol{w}_{boost}, \quad \text{where } \boldsymbol{B} = [\vec{B}_1,\ \vec{B}_2,\ \dots,\ \vec{B}_N]. \tag{4.5}$$

The weight vectors $\boldsymbol{w}_{surface}$ and $\boldsymbol{w}_{boost}$ in equations (4.4) and (4.5) respectively, can both be estimated using least squares.

$$\hat{\boldsymbol{w}}_{surface} = \left(\boldsymbol{S}^\top \boldsymbol{S}\right)^{-1} \boldsymbol{S}\vec{S}, \tag{4.6}$$

$$\hat{\boldsymbol{w}}_{boost} = \left(\boldsymbol{B}^\top \boldsymbol{B}\right)^{-1} \boldsymbol{B}\vec{B}. \tag{4.7}$$

Now by finding the weight vectors $\hat{\boldsymbol{w}}_{surface}$ and $\hat{\boldsymbol{w}}_{boost}$ for a training set where both the surfaces and their corresponding boost matrices are know, a map $f(\boldsymbol{w}_{boost}) = \boldsymbol{w}_{surface}$ can be estimated. Then the surface estimation problem can be split up into three steps:

1. Find the weights of the projection of the boost matrix onto each basis vector by solving equation (4.7).
2. Map w_{boost} to $w_{surface}$ using $f(\boldsymbol{w}_{boost}) = \boldsymbol{w}_{surface}$.
3. Reconstruct the surface by inserting $\boldsymbol{w}_{surface}$ into equation (4.4).

The flow of the method is illustrated in figure 4.2.

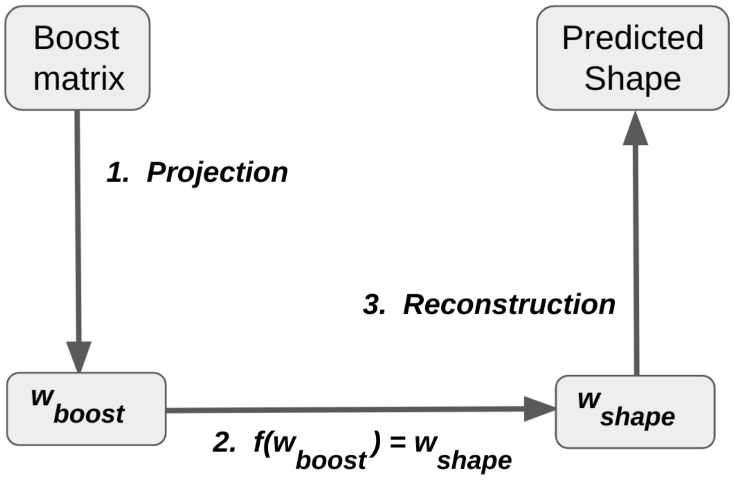


Figure 4.2: Flow of the method.

However, three main issues have not been addressed. First is the possible non-linear relationship between boost and surface, making the assumption that vectorised boost matrices can be linearly expanded inaccurate. However, the hypothesis for this method is that the relationship is linear enough to be able to be approximated as linear. The two other issues concern the choice of basis and estimation of the map $f(x)$. Section 4.2.1 examines possible basis surfaces and section 4.2.2 examines possible ways to estimate the map $f(x)$.

4.2.1 Choice of basis

There are numerous possible basis choices, all with different advantages and disadvantages. Three methods of constructing the basis are examined:

1. Construct a set of basis surfaces by hand. Thereafter, use the boost simulator to construct the corresponding boost matrices.

2. Solve Helmholtz equation in a rectangle and use the eigenfunctions as a surface basis. Then, use the boost simulator to construct the corresponding boost matrices.
3. Find principal components of the training set to find an appropriate basis for the respective space.

1. Use hand picked surfaces as a basis and simulate corresponding boost matrices
The first investigated basis for the boosts and surfaces are hand-picked surfaces and their corresponding boost matrices. The chosen surface basis consists of the surfaces introduced in section 3.2, figure 3.3 for the surface generation algorithm. The corresponding boost matrices are obtained by finding the simulated boost of each surface.

If all or most commonly recurring surfaces are known, the weights of each basis surface could act as an easily interpreted classification of the general appearance of the surface. Additionally, if a new, distinct possible surface is introduced, the basis is very simple to extend with the new surface. There are two main issues with this basis. The first is that since the basis is, to an extent, the same as the basis used to generate the simulated surfaces, it might be too tailored towards that particular set of surfaces and may not generalise well. The other issue is that the hand-picked basis will not be orthogonal. Since the non-linearity of the boost matrix expansion is already ignored, the lack of orthogonality might cause the map $f(x)$ to be too hard to estimate accurately.

2. Use the eigenfunctions from solving Helmholtz equation in a rectangle as a surface basis and simulate boost matrices
The second surface basis is chosen as a family of solutions (eigenfunctions) to Helmholtz equation (2.8), described in section 2.8. There are at least two reasons for going through the trouble of finding this family of eigenfunctions instead of picking a set of arbitrary surfaces as a basis. The first is the pairwise orthogonality of the eigenfunctions, which might make the analysis of the boost matrices in the boost space easier if at least some of the properties of the surface basis are present in the boost basis as well. The second reason is the favourable geometrical properties of the Laplacian. The Laplacian at a certain point can be thought of as a measure of the non-conformity of the point compared to its immediate surroundings [18]. Thus, solving Helmholtz equation results in a set of basis surfaces with slow variations in depth along the surface, which is a reasonable constraint for the kind of deformations this method is meant to model.

The solution to Helmholtz equation grants an infinite set of orthogonal surfaces, for example the family

$$s_{m,n}(x,y) = -\sin\left(\frac{m\pi}{L}x\right)\sin\left(\frac{n\pi}{H}y\right).$$

By keeping only a subset of these eigenfunctions, the underlying surface will not be able to be exactly reconstructed using a linear combination of the eigenfunctions, but approximately well if sufficiently many are kept. The set of eigenfunctions that is used is $\{s_{m,n}(x, y) : (m, n) \in \{1, 2, 3\} \times \{1, 2, 3\}\}$. The boost simulator is then used to find boost matrices for the chosen subset of surfaces. The surfaces are illustrated in figure 4.3.

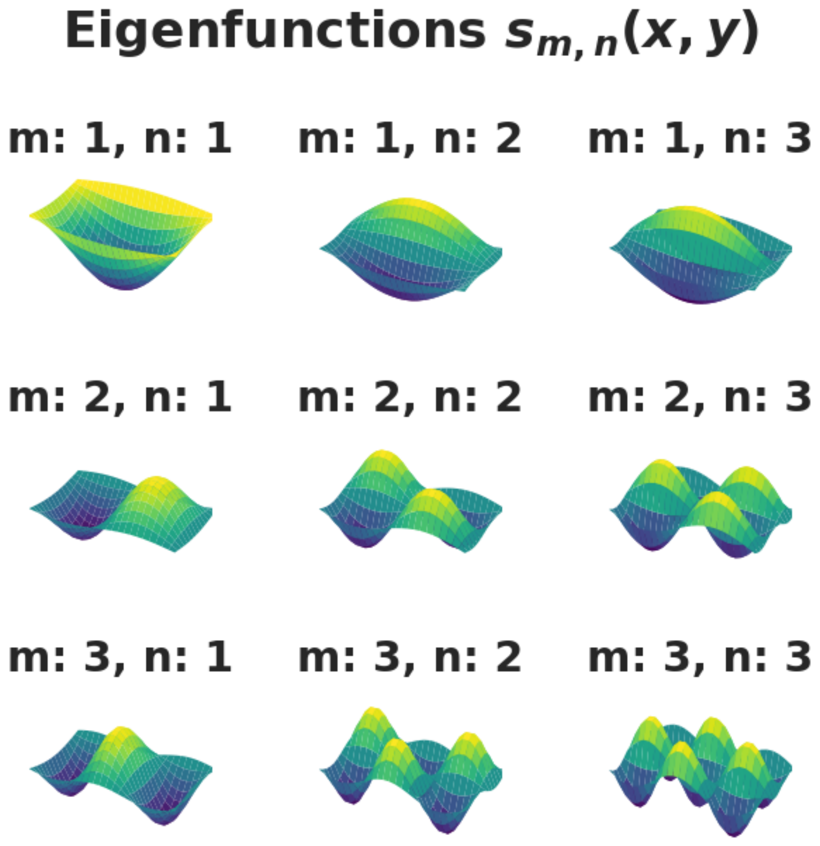


Figure 4.3: Nine eigenfunctions which constitutes the Helmholtz basis.

The simulation is constructed with the assumption that the emitter and detector are placed very close to the curve. The eigenfunctions in equation (4.2.1) consist of sinusoids that are zero along the edge of the surface. Therefore, most surfaces will block the light for some scanlines. This causes the very distinct regions and patterns in the boost matrix seen in for example figure 4.4, which corresponds to the basis surface $s_{2,1}$ in figure 4.3.

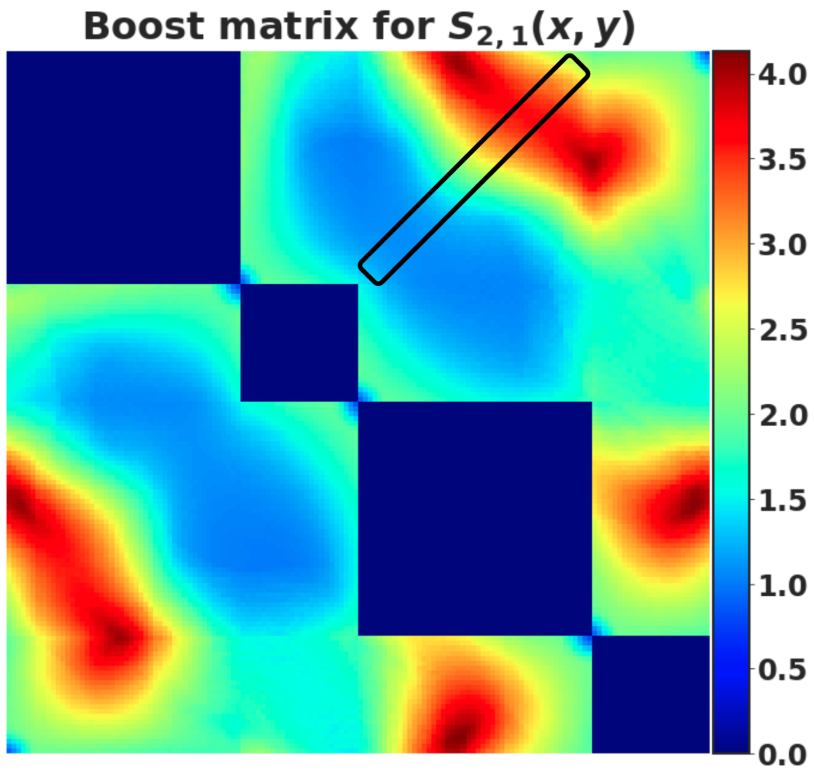


Figure 4.4: Boost matrix for basis surface $s_{2,1}(x, y)$.

The black rectangle in figure 4.4 highlights the part of the boost matrix corresponding to the straight scanlines from one long side to the other. By looking at surface $s_{2,1}$ in figure 4.3, it is easy to see that about half of those scanlines will be blocked by the surface, which shows up as the transition from blue to red inside the black rectangle. It might be possible to use the distinct patterns of each basis to extract information about certain regions of the surface.

3. Using Principal Components as basis
To construct this basis, a training set consisting of surfaces with corresponding boost matrices is needed. Thereafter, PCA is performed on the surface training set as well as on the boost matrix training set respectively. The first m components from the surface PCA alongside the first n components from the boost PCA is used as the basis. For some background regarding why principal components might be a suitable choice of the individual basis, see appendix A.2. The mean surface and mean boost matrix of the training set is illustrated in figure 4.5.

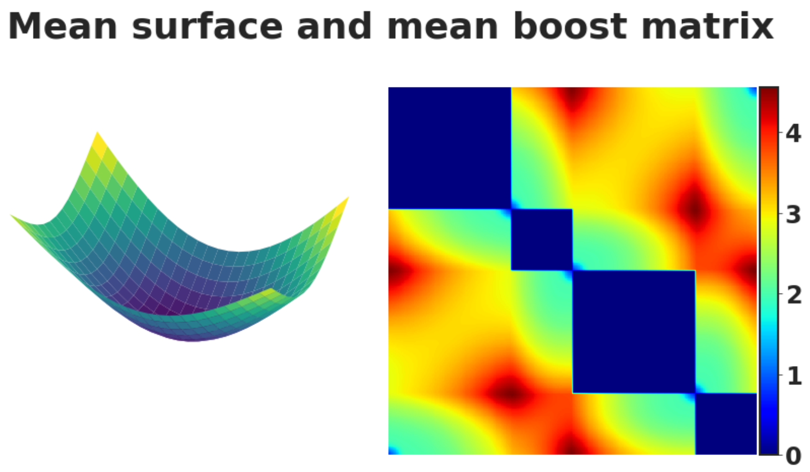


Figure 4.5: Illustration of mean surface and mean boost matrix in the training set.

Hence, the average surface in the data set is a quite 'bowl' shaped surface. The first two principal components are visualised in figure 4.6.

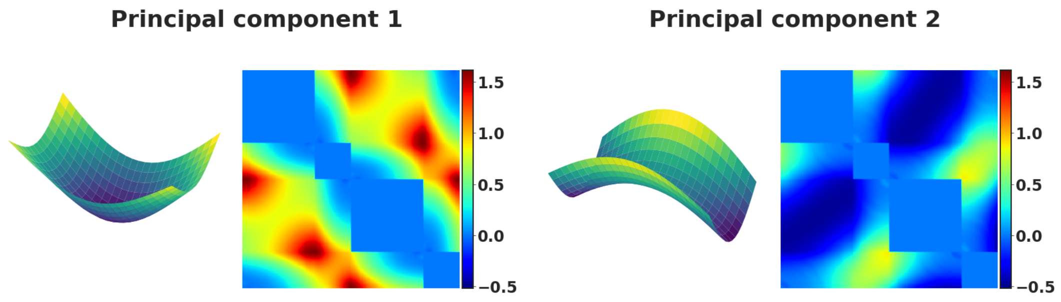


Figure 4.6: Visualisation of the first two principal components from respective space.

To properly determine a suitable number of components to accurately describe the surfaces as well as the boost matrices, the energy metric introduced in section 2.2.2 can be used. Calculating the energy levels for the surface PCA and the boost PCA respectively yields the results illustrated in figures 4.7 and 4.8:

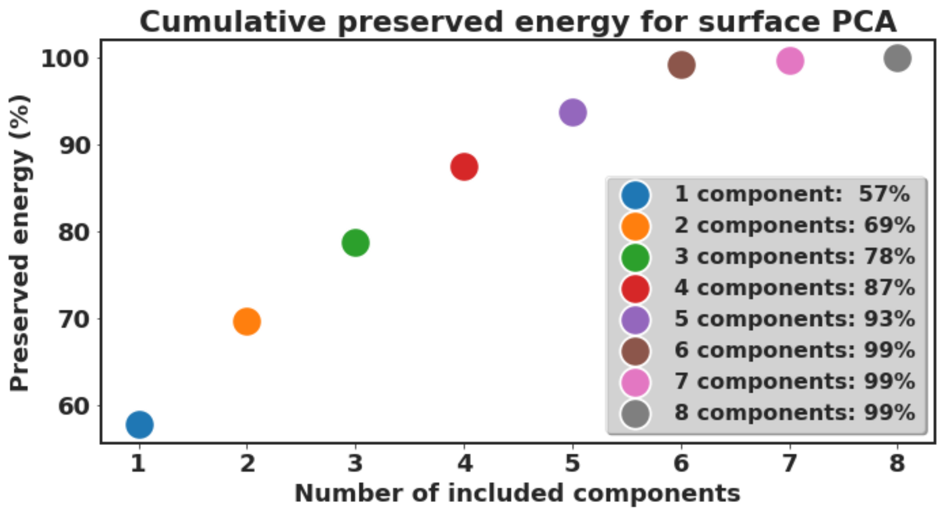


Figure 4.7: Cumulative preserved energy (explained variance) when including more surface principal components.

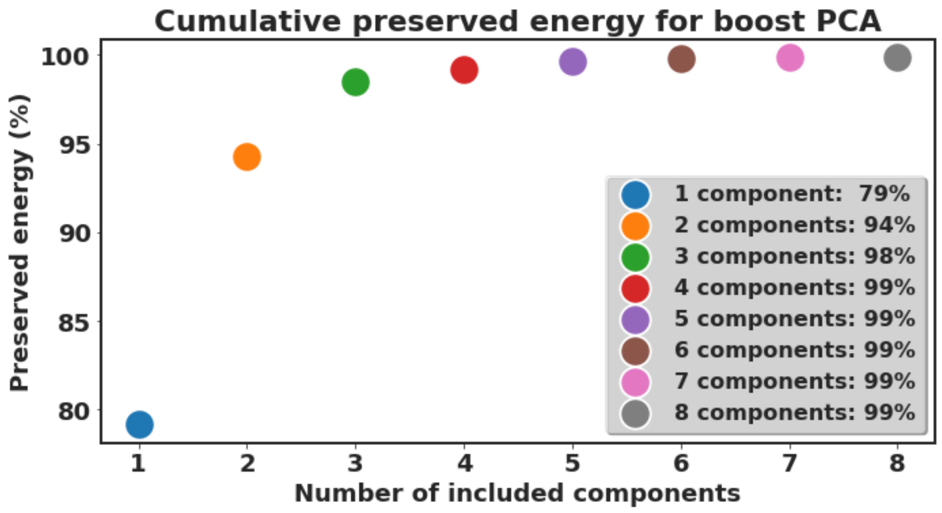


Figure 4.8: Cumulative preserved energy (explained variance) when including more boost principal components.

How much energy that should be preserved is not by any means obvious. Since the issue of non-linearity is ignored and only a few components are actually needed to preserve a very high amount of energy, it may be reasonable to choose the rather conservative amount of 99% percent. To preserve 99% energy, the number of components needed in the surface PCA is six, according to figure 4.7. However, from figure 4.8, only four components are needed to preserve 99% energy for the boost PCA.

There are several reasons to choose principal components as respective basis. They are orthogonal and describe common characteristics of the respective data set, which are both preferred properties of a basis. There are however at least two issues. The first is poor performance on surfaces or boost matrices not included in the training data. The other is that principal component analysis only finds linear manifolds in the data. To find non-linear manifolds, kernel PCA [16] can be used. However, the issue of finding a suitable kernel might itself be as difficult as the original problem. Therefore, methods using CNN based Encoder-Decoder architectures with non-linear activations are investigated in section 4.3 instead of trying to find reasonable kernels for kernel PCA.

4.2.2 Weight space map estimation

There are numerous ways to estimate the map of weights in the boost factor space to the surface space. Two procedures using training data are proposed which consist of:

1. Aligning weight vectors in respective spaces using geometric transformations.
2. Estimating the map using an artificial neural network.

1. Similarity transformation
Consider the vector spaces V and W which are spanned by the vectorised boost basis matrices $\{\vec{B}_i\}_{i=1}^{M}$ and $\{\vec{S}_i\}_{i=1}^{N}$ respectively. A boost vector $\vec{B} \approx \boldsymbol{B w}_{boost}$ can be viewed as a point in the linear expansion of the boost. Similarly, a surface function $\vec{S} \approx \boldsymbol{S w}_{surface}$ can be seen as a point in the linear basis expansions of the surface. The decomposition of each boost matrix and surface function will then generate point clouds in their respective space which represent the weight of each basis of the surface or boost matrix. This opens up the possibility to use point cloud alignment algorithms as a mapping from weights in the boost space to weights in the surface space.

Geometric transformations between the spaces are investigated to see if it would be possible to find a simple, intuitive map between them. The simplest kind of geometric transformation producing reasonable results is preferred, which in this case turns out to be the similarity transformation, described in section 2.2.3. A similarity transformation has fewer degrees of freedom than both affine and projective transformations, making it less prone to overfitting. Therefore, even though both affine and projective transformation would by definition yield an equal or better result than the similarity transformation on the training set, it would not necessarily generalise to perform well on the test set.

2. Map weights using an artificial neural network
The above mentioned method to map weights from one space to another is linear, which might not be enough since a linear combination of surfaces does not neces-

sarily result in a boost matrix that is a linear combination of the corresponding boost basis matrices. To address this non-linearity, fully connected artificial neural networks are trained to map vectors in the boost factor space to vectors in the surface space. The model architectures can be found in appendix B.2, B.3 and B.4.

4.3 CNN-based Encoder-Decoder

Since the map between a surface and the corresponding boost may very well be non-linear, the Encoder-Decoder architectures might perform better than PCA at reducing dimensions, as described in section 2.2.4. Two architectures and training approaches are investigated, one straight-forward approach and one restricting the network to smooth and realistic surfaces. Detailed descriptions of kernel sizes, number of layers, activation functions and more can be found in appendix B.5.

4.3.1 Simple Encoder-Decoder

The straight-forward Encoder-Decoder architecture takes a boost matrix as input and outputs a predicted surface. The network is trained using a MSE reconstruction loss between the predicted and target surface:

$$\mathcal{L}(Y, \widetilde{Y}) = \frac{1}{MN} \sum_{i=0}^{M} \sum_{j=0}^{N} \left(Y_{i,j} - \widetilde{Y}_{i,j} \right)^2,$$

where $\widetilde{Y}$ is the output of the decoder. The loss function in equation (4.3.1) is a discretised version of the main problem formulation in equation (4.1). An overview of the architecture is illustrated in figure 4.9.

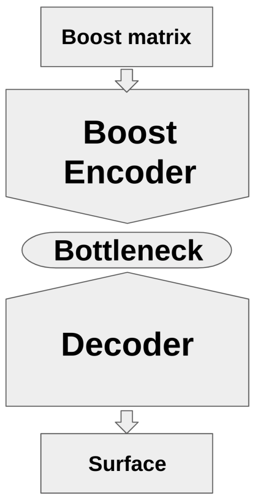


Figure 4.9: Flowchart of a simple Encoder-Decoder architecture.

4.3.2 Surface autoencoder combined with boost encoder

To ensure a smooth and continuous looking reconstruction, a slightly more complex architecture is proposed. The architecture is illustrated in figure 4.10.

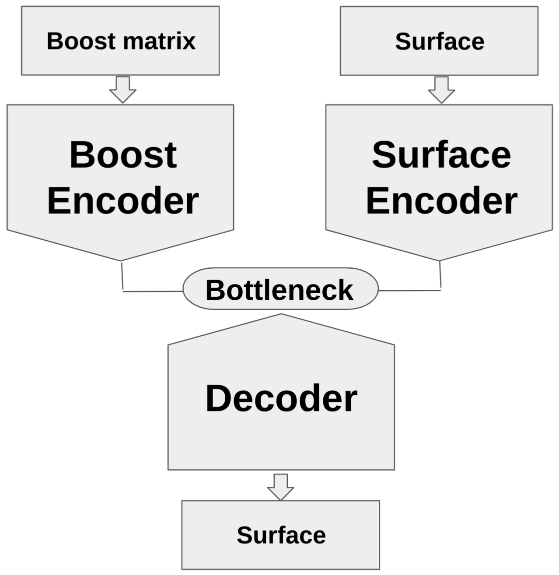


Figure 4.10: Flowchart of combined autoencoder and surface encoder architecture.

Instead of the simple Encoder-Decoder architecture in section 4.3.1, an undercomplete surface autoencoder is used to learn a low-dimensional representation of the surfaces. The boost encoder is trained in tandem to encode into the same low-dimensional representation as the surface encoder encodes the surface to. Lastly, the encoder part of the autoencoder is simultaneously trained to encode the surface in a way that the boost encoder can also achieve by penalising differences in the output of the two encoders. The goal of this strategy is to make both encoders produce the same bottleneck layer when given the boost and surface from a corresponding boost-surface pair as input. By letting the surface autoencoder decide the content of the bottleneck layer, the predicted surface is hypothesised to be smoother and more continuous looking compared to the result of the simple Encoder-Decoder network in section 4.3.1.

With this architecture, the training challenge is split up into a few more easily managed parts:

1. Train the surface encoder to find a low-dimensional representation of the surface.
2. Train the decoder to reconstruct the surface from the low-dimensional representation.

3. Train the boost encoder to find a low-dimensional representation of the boost matrix matching the low-dimensional representation of the corresponding surface.
4. Train the surface encoder to recognise what information the boost encoder has access to and thus make it easier for the boost encoder to learn the low-dimensional representation.

The first two parts are simply the autoencoder part of the network, where the surface encoder and decoder are trained to encode the surface and decode the bottleneck layer respectively to minimise the reconstruction loss in equation (4.3.1). The third part consists of training the boost encoder to minimise the mean squared error between its output and the output of the surface encoder. The final part is achieved by including a part of the loss of the boost encoder to the reconstruction loss of the surface autoencoder.

Chapter 5

Results

To properly evaluate the methods, three data sets are used. The first is a training set consisting of 5000 generated surfaces with simulated boost matrices. The second is a test set of 1000 surfaces and simulated boost matrices generated in the same way. This data set is referred to as the *generated* test set. The last data set consists of 34 measured surfaces combined with their measured signal matrices, which is then divided by the approximated flat panel signal to get an estimation of the boost matrices. This last data set will be referred to as the *real* test set. Many of the surfaces in the real test set are rather different from the more common generated surfaces. This is by design to test how well the methods can generalise and adapt.

5.1 Iterative method

The iterative method proposed in section 4.1 does not manage to converge for any test surfaces. The gradients generally point in directions that update the curves to yield boosts closer to the real boosts than before the update. However, since there are a large set of curves with approximately the same boost, the gradients can update the curves in directions that are not necessarily correct. The refit step in the algorithm, which takes one-dimensional curves and produces a surface, does not manage to counteract the issue of having equal boost for many curves. Thus, there are no results to show for this method since all tests diverges.

5.2 Comparison of weight space alignment techniques

Finding the optimal similarity transformation between the surface and boost point clouds of the training data and applying it on weights from a sample of surfaces and boost matrices grants the alignment illustrated in figures 5.1 to 5.3. Only the alignment of the weights of two basis shapes in the respective space are illustrated for sake of clarity.

Figure 5.1 illustrates the alignment using the handpicked surface basis weights.

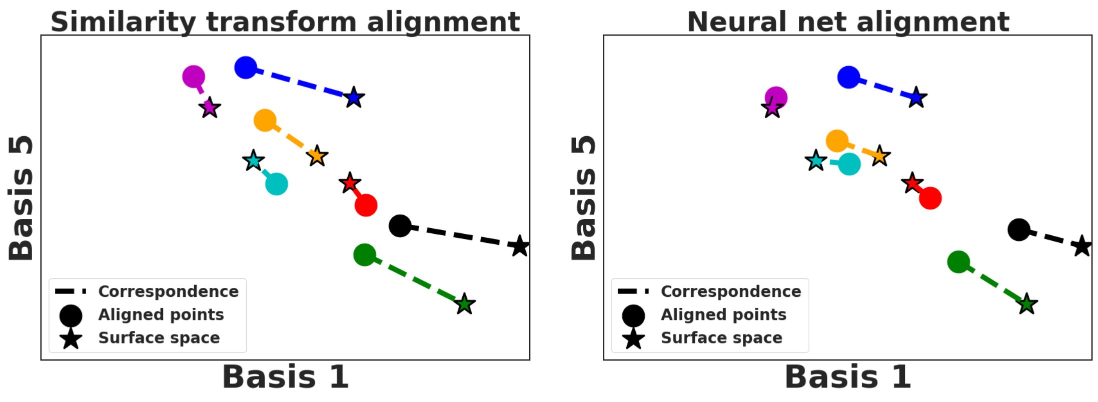


Figure 5.1: Handpicked basis: Comparison between using similarity transformation (left) and neural network (right) as point cloud mapping.

The stars in figure 5.1 are where the correct $w_{surface}$ are located, while the circles illustrate the respective function approximations $f(w_{boost})$. The neural network appears to align the example points better than the similarity transformation.

Figure 5.2 illustrates the alignments for the Helmholtz basis weights.

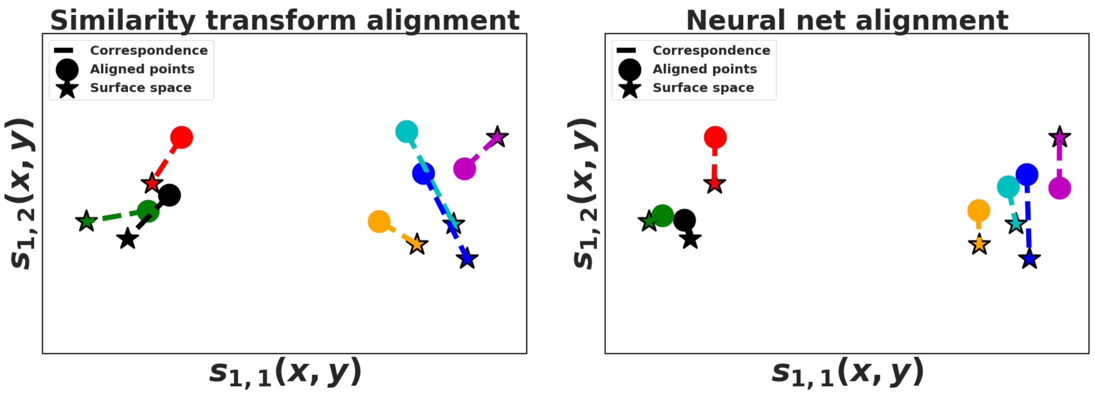


Figure 5.2: Helmholtz basis: Comparison between using similarity transformation (left) and neural network (right) as point cloud mapping.

The alignments in figure 5.2 indicate a slightly better performance of the neural network than the similarity transformation on the example points.

Lastly, figure 5.3 illustrates the alignments using the PCA basis weights.

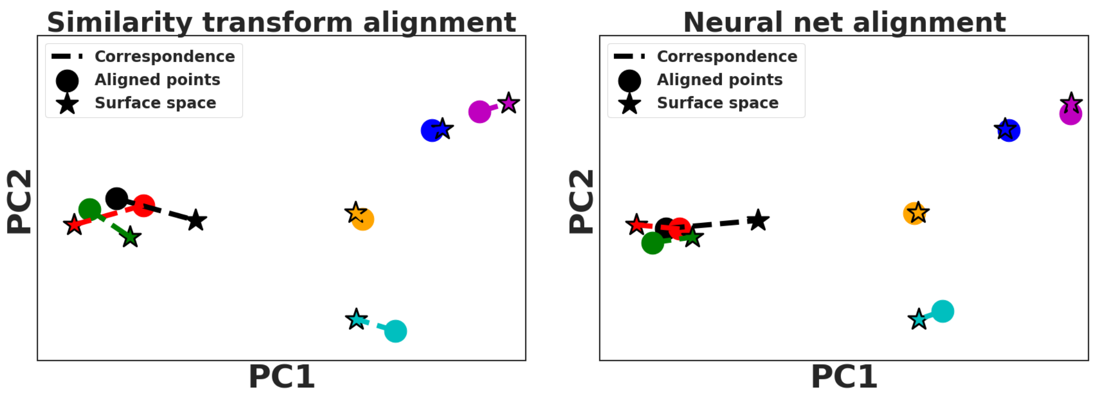


Figure 5.3: PCA basis: Comparison between using similarity transformation (left) and neural network (right) as point cloud mapping.

The neural network appears to perform slightly better on the example points using the PCA basis as well.

The *root mean squared error* (RMSE) of the target and aligned points is used to evaluate the performance objectively. The root mean squared error is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(\vec{p}_i - \vec{t}_i)^2} \tag{5.1}$$

where $\vec{p}$ and $\vec{t}$ are the prediction and target vector respectively. The average RMSE for the generated test set is summarised in table 5.1.

Table 5.1: Average RMSE for the different alignment methods on the generated test set. Bold methods outperform the others. Since the different bases span different spaces, they cannot be compared. Hence, comparisons are made row-wise.

Basis	***Alignment method***	
	Sim. Transf.	**Neural Net.**
Handpicked	0.265	**0.199**
Helmholtz	0.130	**0.067**
PCA	1.786	**1.558**

The neural network approach outperforms the similarity transformation for all different bases on the generated test set. The average RMSE for the different methods and bases on the real test set is summarised in table 5.2:

Table 5.2: Average RMSE for the different alignment methods on the real test set. Bold methods outperform the others. Since the different bases span different spaces, they cannot be compared. Hence, comparisons are made row-wise.

Basis	*Alignment method*	
	Sim. Transf.	**Neural Net.**
Handpicked	**2.143**	2.393
Helmholtz	**0.196**	0.282
PCA	2.953	**2.823**

Contrary to the generated test set, the similarity transformation approach outperforms the neural network on both the handpicked and the Helmholtz basis. The neural network does however perform slightly better on the PCA basis.

5.3 Reconstructions

The main goal is not to produce the best point cloud alignments, but rather to reconstruct the surface as well as possible. The methods are evaluated using three different metrics. The first uses the RMSE measure defined in equation (5.1), but with prediction vectors of size 25×15 describing the depth at equidistant points over the surface. The second metric is a normalised version of the RMSE to counteract the fact that surfaces with larger depths produce larger errors than surfaces with smaller depths in the RMSE measure. The normalised RMSE is defined as:

$$\text{Normalised RMSE} = \frac{1}{m}\sqrt{\frac{1}{d}\sum_{i=1}^{d}(\vec{p}_i - \vec{t}_i)^2}.$$

where m is the largest absolute value found in either $\vec{p}$ or $\vec{t}$. The last metric is the maximum deviation which is defined as:

$$\text{Maximum deviation} = \max_i |\vec{p} - \vec{t}|$$

For all three metrics, smaller values are preferred for a good reconstruction.

5.3.1 Evaluation on the generated test set

Table 5.3: Average values for all surfaces in the generated test set, bold numbers indicate that the method performs best with regard to that metric. Comparisons are made column-wise.

Method	RMSE (mm)	Norm. RMSE	Max Dev. (mm)
Handpicked + Sim. Transf.	0.205	0.094	0.641
Handpicked + Neural Net.	**0.147**	**0.068**	**0.405**
Helmholtz + Sim. Transf.	0.324	0.155	1.094
Helmholtz + Neural Net.	0.296	0.132	1.127
PCA + Sim. Transf.	0.183	0.088	0.463
PCA + Neural Net.	0.188	0.089	0.444
Simple Encoder-Decoder	0.179	0.084	0.626
Autoencoder + Boost encoder	0.169	0.080	0.482

In table 5.3 it can be seen that using the handpicked basis combined with a neural network map performs the best on the generated test data in regard to all three metrics. The worst performing, in terms of the two RMSE metrics, is using the Helmholtz basis combined with a similarity transformation. The worst in regard to the average max deviation is the Helmholtz basis combined with a neural network. Figures 5.4 and 5.5 illustrate a reconstruction using the handpicked basis combined with a neural network (the best) as well as one using the Helmholtz basis combined with a similarity transformation (the worst).

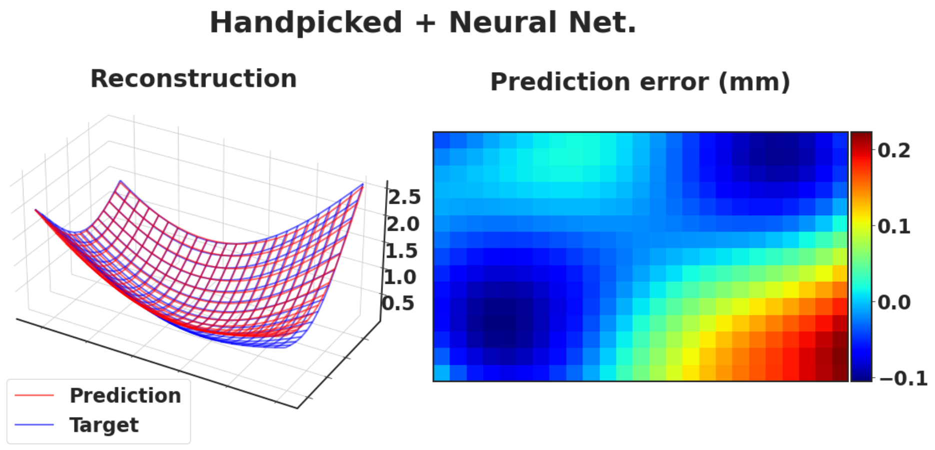


Figure 5.4: Example reconstruction using the handpicked basis combined with a neural network.

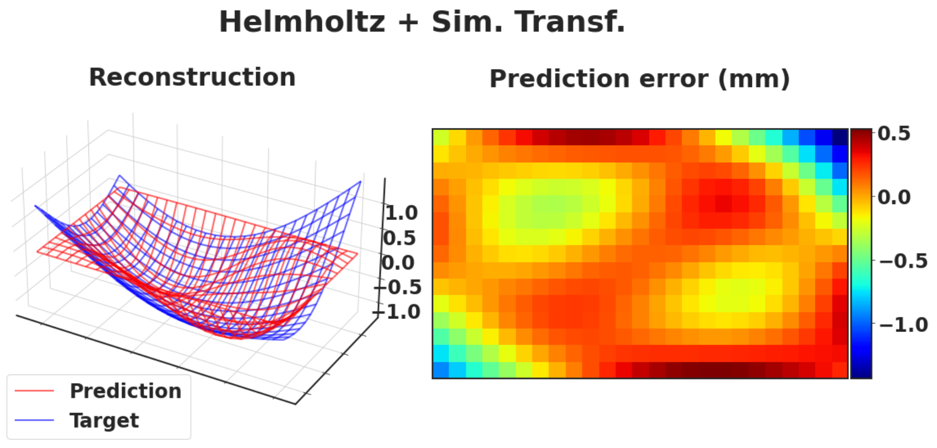


Figure 5.5: Example reconstruction using Helmholtz basis combined with a similarity transform. Note that this is the same target surface as in figure 5.4 but shifted down.

From figures 5.4 and 5.5 it can be seen that the handpicked basis combined with a neural network reconstructs the surface very well, while the Helmholtz basis using similarity transformation struggles. The Helmholtz basis has the issue that the surface which it tries to reconstruct does not fulfil the boundary conditions. Furthermore, the target surface is very different from the basis surfaces in the Helmholtz basis, hence it is quite likely that the target surface simply cannot be constructed accurately by the basis surfaces. The methods using PCA basis as well as the Encoder-Decoder methods perform only slightly worse than the handpicked basis combined with a neural network. Figures 5.6 and 5.7 illustrate the reconstructions using PCA basis accompanied by a similarity transform as well as a surface autoencoder combined with a boost encoder on the same target surface.

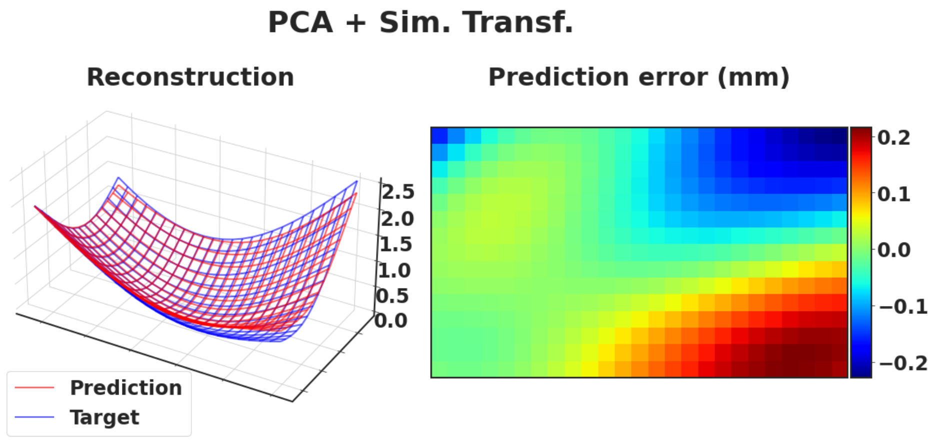


Figure 5.6: Example reconstructions using the PCA basis combined with a similarity transformation.

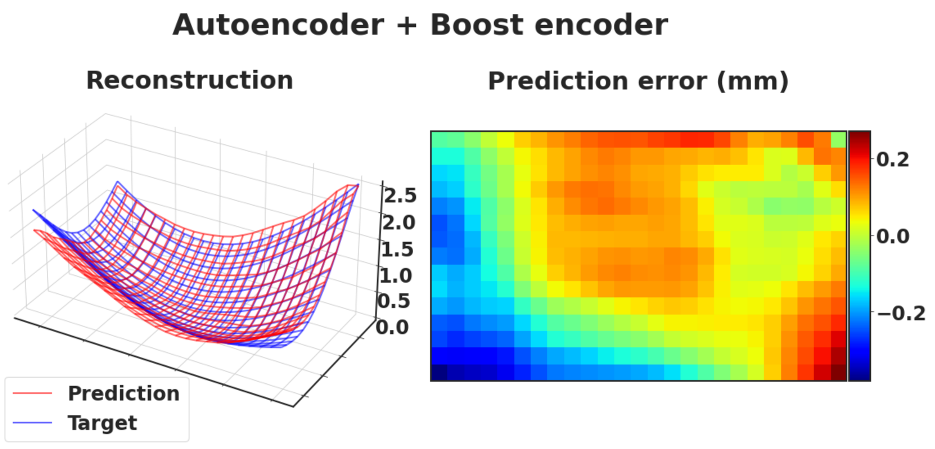


Figure 5.7: Example reconstruction using the autoencoder combined with the boost encoder.

It can be seen in figures 5.6 and 5.7 that those methods manage to reconstruct the surfaces rather well. One thing to note that differentiates the Encoder-Decoder methods is that since they are not tied to any basis surfaces, they are not forced to be as regular and smooth as the reconstructions of other methods. This can for example be seen in the upper right of the reconstruction in figure 5.7, where the corner has an irregular shape.

5.3.2 Evaluation on the real test set

Table 5.4: Average values for all surfaces in the real test set, bold numbers indicate that the method performs best with regard to that metric. Comparisons are made column-wise.

Method	RMSE (mm)	Norm. RMSE	Max Dev. (mm)
Handpicked + Sim. Transf.	0.648	0.257	1.787
Handpicked + Neural Net.	1.052	0.402	2.545
Helmholtz + Sim. Transf.	0.337	0.151	0.986
Helmholtz + Neural Net.	0.579	0.208	1.365
PCA + Sim Transf.	0.244	0.124	0.783
PCA + Neural Net.	**0.211**	**0.111**	**0.663**
Simple Encoder-Decoder	0.410	0.205	1.143
Autoencoder + Boost encoder	0.373	0.184	1.110

The results from table 5.4 tell a different story than the one in table 5.3. On the real test set, the handpicked method combined with a neural network performs *worst*, rather than best as for the generated test set. The best method on the real test set is the one using the PCA basis combined with a neural network. Figures 5.8 and 5.9 illustrate example reconstructions for the PCA basis combined with a neural network (best) as well as one for the handpicked basis combined with a neural network (worst):

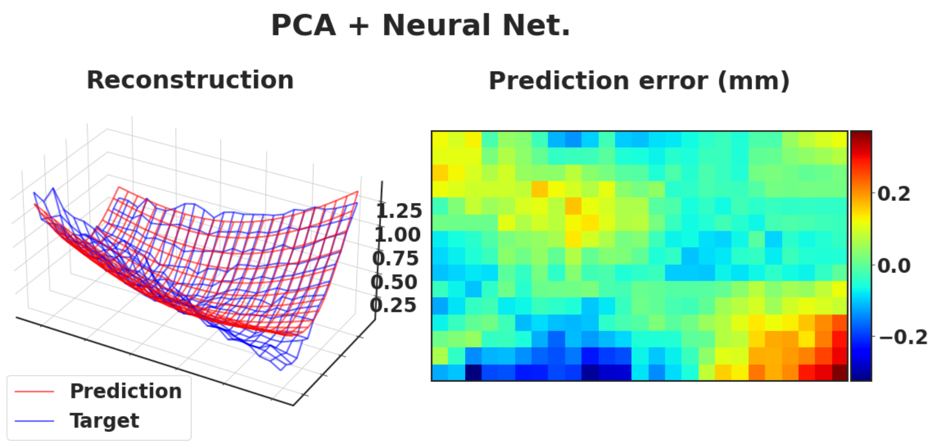


Figure 5.8: Example reconstruction using the PCA basis combined with a neural network.

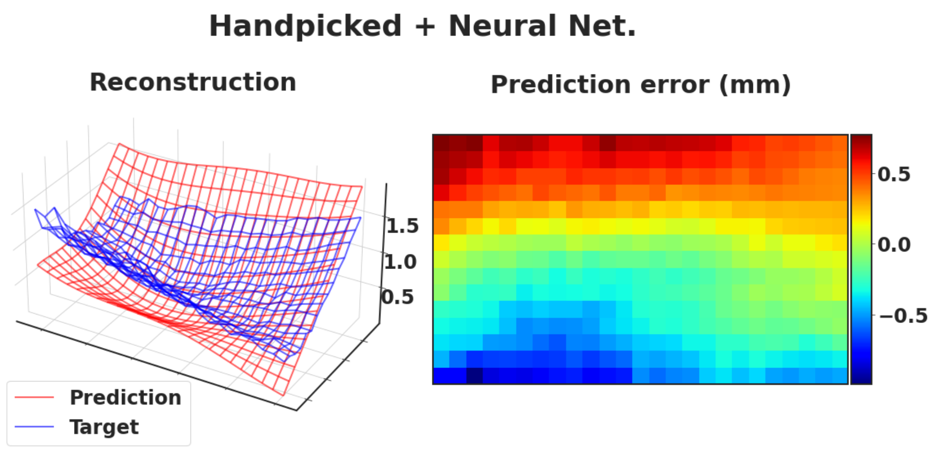


Figure 5.9: Example reconstruction using the handpicked basis combined with a neural network.

It can be seen from the reconstructions in figures 5.8 and 5.9 that the PCA method combined with a neural network performs rather well, while the handpicked method combined with a neural network performs poorly. It should be noted that all methods perform worse on the real test set, except for the Helmholtz basis combined with a similarity transform which stays about the same. This could be explained by the fact that many of the surfaces in the real test set is quite well conditioned for the

Helmholtz basis. Figure 5.10 illustrates a reconstruction where the target function is well suited for the Helmholtz basis:

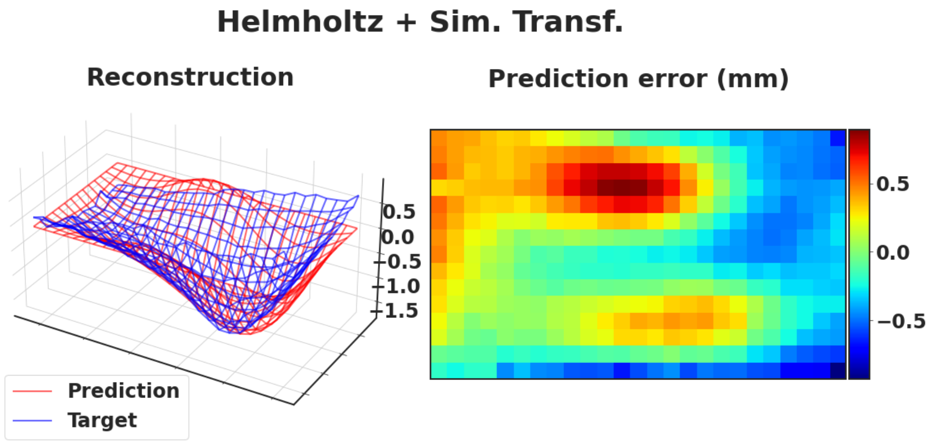


Figure 5.10: Example reconstruction using the Helmholtz basis combined with a similarity transform.

Since off-centred 'bowl' shaped surfaces are captured by the basis surfaces in the Helmholtz basis, and the fact that the target surface in figure 5.10 almost fulfils the boundary conditions, the method can reconstruct the surface rather accurately. The performance of the Encoder-Decoder methods ranks somewhere in the middle of all methods. An example reconstruction using a surface autoencoder combined with a boost encoder method is illustrated in figure 5.11:

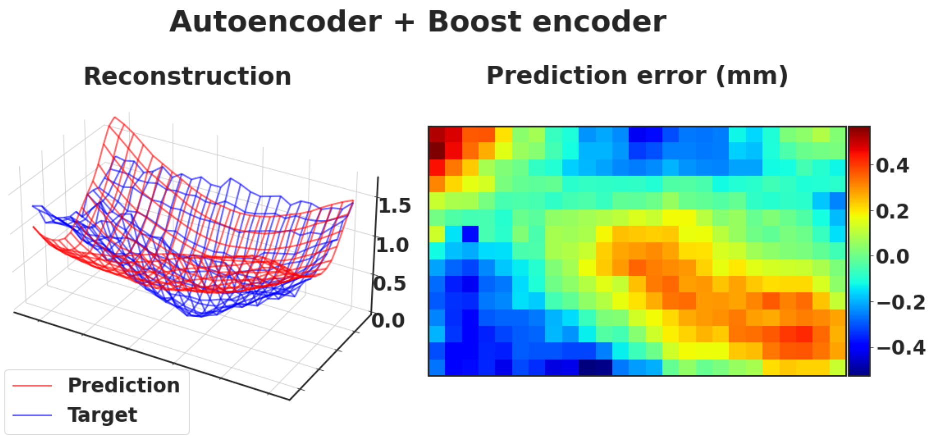


Figure 5.11: Example reconstruction using combined with a boost encoder.

The reconstruction in figure 5.11 is not awful, but not satisfactory either. The method does not seem be able to capture the steep and almost pointy surface. The issue with the method most likely lies in the fact that the training data is too ‘nice’, resulting in the method not generalising well.

Considering that the methods using the PCA basis are clearly number one and two on the real test set and also performing very well on the generated test set, they are deemed the two best. Out of them, in terms of the metrics, the edge is given to the one using the neural network mapping. Several more reconstructions using the PCA basis combined with a neural network mapping can be found in appendix D.

Chapter 6

Discussion

The aim of this thesis is to investigate the possibility to reconstruct a surface by using optical emitters and detectors at the edge of the surface and to evaluate different reconstruction methods on both simulated and real data.

The evaluation shows that it is possible to deduce information about the general shape of the surface, but that most methods still experience some reconstruction errors and that they do not generalise very well. The non-linear nature of the problem makes it difficult to make any kind of prediction regarding finer details.

All methods show promising results on the generated test data, but PCA combined with a neural network performs best on the real test data, closely followed by PCA combined with similarity transformation.

Some conclusions can be drawn from the results in tables 5.1, 5.2, 5.3 and 5.4:

- All methods except those using the Helmholtz basis show a rather large drop in reconstruction performance from the generated test data to the real test data, which indicates that they do not manage to generalise to the previously unseen surfaces present in the real test data.
- Alignment using similarity transformation seems to generalise better than using neural networks, at least when the real test data contains lots of new surfaces.
- Handpicked basis performs well on generated data but poorly on real data, which is to be expected since the generated test data was (partly) generated with the same surfaces that were used as the handpicked basis.
- Methods using the Helmholtz basis do not perform well on the generated test data since the method assumes that the edge is set to zero, which is rarely the case for the generated surfaces. The real data is slightly better conditioned and yields results about as good as the results of the Encoder-Decoder architectures.

- Similarity transform and neural network yield very similar results together with the PCA basis, both for generated and real data. This indicates that the relationship between the first few principal components of boost and surfaces might almost be linear, despite the fact that the original problem is non-linear.
- The Encoder-Decoder architectures work well on the training data, but do not generalise well since the real data contains lots of noise, which the networks are not trained to handle.

6.1 Review of the different methods

Iterative method

As mentioned in section 5.1, since there is a large set of curves yielding approximately the same boost, the gradients update the curves in directions that are not necessarily correct. Since the refit does not manage to counteract this, some additional information might be required for this method to work. One possibility is to add some weighting function to prefer the more common curve shapes, and hence penalise improbable and uncommon curves. Another issue is that the gradients rely on a neural network which might occasionally grant poor directions. The advantage, and possibly even necessity, of using a neural network to calculate the gradients is that it is much cheaper than using the simulator, and also more numerically stable.

If this method worked, it would probably be able reconstruct the surfaces with much higher accuracy than what can be achieved with the other proposed methods, especially for the more fine-grained details. Furthermore, it would only require training data for the neural network approximation of the boost function.

Handpicked basis

The handpicked basis performs well on the generated test set since the generated test set is (partly) generated using the surfaces in the handpicked basis. It then performed very poorly on the real test set, since the real test set contains rather different surfaces. So the method does not generalise well. However, it was an idea worth investigating for two reasons. First, it is a quite natural approach if only a few types of surfaces are common. Then the handpicked basis could be the set of those, and thereby be likely to work well. Second, it is easy to extend the basis when a new kind of surface is discovered. Additionally, the possibility to interpret the largest weight of each basis as a kind of classification of the general surface makes the method even more appealing. Lastly assuming that there is a way to improve the performance of this method with some additional effort, each basis surface might be able to be related to certain kind of deformations or defect, meaning it might even be possible to deduce information about the cause of the deformation, and not only the surface, directly from the distribution of basis weights.

Helmholtz basis

There are at least two reasons why the Helmholtz basis does not perform well. First and foremost, the boundary conditions on the surfaces are almost never fulfilled, making the problem ill-conditioned from the start. Second, to accurately reconstruct for instance a surface as the one in figure 5.5 would require a lot of eigenfunctions due to the sinusoidal nature of the eigenfunctions. In other words, the appearance of the eigenfunctions are very different from the surfaces they are supposed to reconstruct.

The main benefit of the method is that, independent of any training data, infinite number of orthogonal basis surfaces can be constructed, making it theoretically possible to reach an arbitrarily high level of precision. However, when testing using 16 basis shapes instead of 9, the results on the real test set became poorer. This is most likely caused by the map between spaces becoming much larger and more complex. Lastly, as briefly mentioned in section 4.2.1, something interesting about this method is that since the different eigenfunctions will block light in different regions, it might be possible to use the distinct patterns of each basis to extract information about certain regions of the surface.

PCA basis

Both an advantage and disadvantage of PCA compared to the other methods is that the training data is used to find the surface basis. The fact that the surfaces are closely related to the data as well as being orthogonal creates better conditions for the method to work than the other ways of choosing a basis. Of course, this means that a large part of the success of the method depends on the training data being diverse and that it contains variations of the possible surfaces that it might need to reconstruct. Since six surface components manages to preserve almost all energy in the data, the method is from the start better conditioned than the Helmholtz method, which needs as many as nine basis surfaces to produce a decent reconstruction.

The boost basis of the PCA is also found using the training data, which means that it is not exactly corresponding the surface basis. This opens up a few more possibilities compared to the other methods. For example, the possibility to choose different amount of basis components might have contributed to the improved result of the PCA method compared to the others, since the neural network map could be simplified from 6 input - 6 output to 4 input - 6 output, reducing the number of parameters needed for the mapping. An additional benefit is that, since the boost basis is found by PCA as well, the boost basis matrices are pairwise orthogonal.

Alignment using similarity transformation

It is surprising that similarity transformations managed to estimate the map between the spaces as well as it did. Since the boost is not generally linear, it was not expected that the map between the spaces could be done by a linear map, which also has so few degrees of freedom. Since the degrees of freedom are so low, it is not as prone to overfitting as the neural networks. The simplicity, cheapness and memory efficiency of the similarity transformation makes it very compelling.

A disadvantage with this type of transformation is that it requires an equal amount of components in both spaces, which might not always be optimal. For example, the PCA method needed 6 surface components to preserve 99% energy, but only 4 components to preserve the same amount of energy in the boost data.

Alignment using neural network

Aligning the boost and surface spaces using a neural network works about as expected. It performs very well on the generated test set, but struggles on the real test set, which consists of quite different surfaces. This is possibly due to overfitting to the training set. Since the generated test set is rather similar to the training set it is expected for the networks to perform well, because the network is well fitted to it. However, the poor performance on the real test set shows that more varied and diverse training data is needed to train a network that generalizes well. The noteworthy exception is using the PCA basis. The reason for this is unclear, but it could be due to the fact that the PCA basis have both an orthogonal surface basis and an orthogonal boost basis, making the reconstruction easier.

Another issue of using the neural network is that the calculations are expensive compared to other simpler transformations and requires quite a lot of infrastructure to perform a simple inference. These problems are possible to work around, but may not be worth the effort since similarity transformation seems to work almost equally well.

Lastly, an advantage of using neural networks is, as mentioned earlier, that they do not have the restriction of equal number of inputs and outputs that similarity transformations have. Instead, the number of inputs can be based directly on how much energy that needs to be preserved.

Encoder-Decoder architectures

The Encoder-Decoder architectures often manage to capture the general shape of the surfaces, but handles new surfaces poorly. The main advantage with this method compared to the ones based on linear combination of some basis is that it has the possibility to make more detailed reconstructions since it has access to more information than for example the PCA based methods. The combined surface autoencoder and boost encoder performs slightly better than the simple Encoder-Decoder, but still worse than the PCA linear combination method, despite having access to more information. It is most likely possible to improve the performance of these methods a lot by tuning the architecture, adding some regularisation to the loss function during training and by generating more realistic training data.

6.2 Future research

- Study the iterative method more in-depth and investigate if it is possible to make it work by for example weighting to gradients to prefer the most common shapes. This is probably the most interesting future research building upon this thesis since it would make it possible to reconstruct the surfaces with much higher accuracy than what is achieved in this thesis.
- Investigate the Helmholtz basis method more in depth. As mentioned earlier, having access to an infinite number of orthogonal basis surfaces might make it possible to reconstruct surfaces with an arbitrary level of accuracy. Additionally, it would be done without having to use any slow and expensive gradient descent algorithm.
- Use a setup that does not utilise the simplifications made in this thesis. For example, investigate how well the methods work for larger angles of incidence. Would noise and other disturbances of the signal make it impossible to reconstruct the surfaces? If it is possible to have the emitters and detectors further away from the surface, it opens up the possibility to try the methods on surfaces with much larger deformations.
- Collect more real test data with more commonly occurring surface shapes to get a better understanding of exactly what the methods are incapable of reconstructing.
- Add noise and component variation to the boost simulator to make the training data more diverse and methods more robust.
- Find a way to obtain a better flat panel. There is a clear issue with using the approximately flat panel as a reference, since it might cause structural errors in the boost estimation if it is not perfectly flat. An alternative is to calculate the theoretical signals of a perfectly flat panel, which comes with its own set of challenges.

Bibliography

[1] L. C. Böiers. *Mathematical methods of Optimization*. Ed. by Studentlitteratur. Lund, Sweden: Studentlitteratur, 2010, pp. 41–101. ISBN: 9789144070759.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL: https://www.deeplearningbook.org/contents/autoencoders.html.

[3] M. Hébert. "Reflection and transmission of light by a flat interface. Fresnel's formulae". Institut d'Optique Graduate School at ParisTech. 2013. URL: http://paristech.institutoptique.fr/site.php?id=797&fileid=11468.

[4] S. Herbort and C. Wöhler. "An introduction to image-based 3D surface reconstruction and a survey of photometric stereo methods". In: *3D Research* 2 (Sept. 2011), pp. 1–17. DOI: https://doi.org/10.1007/3DRes.03(2011)4.

[5] A. S. Householder. "Unitary Triangularization of a Nonsymmetric Matrix". In: *Journal of the Association for Computing Machinery* 5.4 (Oct. 1958), pp. 339–342. ISSN: 0004-5411. DOI: https://doi.org/10.1145/320941.320947.

[6] W. Kabsch. "A solution for the best rotation to relate two sets of vectors". In: *Acta Crystallographica Section A* 32.5 (1976), pp. 922–923. DOI: https://doi.org/10.1107/S0567739476001873.

[7] O. Kafri and I. Glatt. "Moire Deflectometry: A Ray Deflection Approach To Optical Testing". In: *Optical Engineering - OPT ENG* 24 (Dec. 1985), pp. 944–960. DOI: https://doi.org/10.1117/12.7973607.

[8] A. LeNail. "NN-SVG: Publication-Ready Neural Network Architecture Schematics". In: *Journal of Open Source Software* 4.33 (2019), p. 747. DOI: https://doi.org/10.21105/joss.00747.

[9] G. Lippolis et al. "Automatic registration of multi-modal microscopy images for integrative analysis of prostate tissue sections". In: *BMC Cancer* 13.408 (2013). DOI: https://doi.org/10.1186/1471-2407-13-408.

[10] H. D. Macedo and J. N. Oliveira. "Typing linear algebra: A biproduct-oriented approach". In: *Science of Computer Programming* 78.11 (2013), pp. 2160–2191. ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2012.07.012.

[11] K. Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: https://doi.org/10.1080/14786440109462720.

[12] F. L. Pedrotti, L. M. Pedrotti, and L. S. Pedrotti. *Introduction to Optics, International Edition*. Ed. by E. Fahlgren. 3rd ed. Upper Saddle River, NJ: Pearson, 2006. ISBN: 0-13-197133-6.

[13] E. Plaut. *From Principal Subspaces to Principal Components with Linear Autoencoders*. Dec. 2018. URL: `https://arxiv.org/pdf/1804.10253.pdf` (visited on 02/23/2021).

[14] P. Ramachandran, Z. Barret, and Q. Le. "Searching for Activation Functions". In: 2018. URL: `https://arxiv.org/pdf/1710.05941.pdf`.

[15] R. Rojas. *Neural Networks: A Systematic Introduction*. 1st ed. Berlin, Germany: Springer-Verlag Berlin Heidelberg, 1996, pp. 151–184. ISBN: 978-3-642-61068-4.

[16] B. Schölkopf, A. Smola, and K. R. Müller. "Nonlinear Component Analysis as a Kernel Eigenvalue Problem". In: *Neural Computation* 10.5 (1998), pp. 1299–1319. DOI: `https://doi.org/10.1162/089976698300017467`.

[17] A. Sparr and G. Sparr. *Kontinuerliga System*. Ed. by Studentlitteratur. 2:8. Lund, Sweden: Studentlitteratur, 2000. ISBN: 978-91-44-01355-8.

[18] D. F. Styer. "The geometrical significance of the Laplacian". In: *American Journal of Physics* 83.12 (2015), pp. 992–997. DOI: `https://doi.org/10.1119/1.4935133`.

[19] S. Umeyama. "Least-Squares Estimation of Transformation Parameters Between Two Point Patterns". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.4 (1991), pp. 376–380. DOI: `https://doi.org/10.1109/34.88573`.

Appendix A

Theory Appendix

A.1 Helmholtz equation in a rectangle - Derivation of eigenfunctions and eigenvalues

Let $S(x, y)$ denote a twice continuously differentiable surface function in the rectangle $\Omega = \{(x, y) \mid 0 \leq x \leq L,\ 0 \leq y \leq H\}$. Furthermore, let $S(x, y)$ be zero on the boundary. The Helmholtz equation is then

$$\begin{aligned}
&-\Delta S(x, y) = \lambda S(x, y), \quad S \in \mathcal{D} \qquad \text{(A.1)}\\
&\Omega = \{(x, y) \mid 0 \leq x \leq L,\ 0 \leq y \leq H\},\\
&\mathcal{D} = \{S \in \mathcal{C}^2(\Omega) \mid S(0, y) = S(L, y) = S(x, 0) = S(x, H) = 0\}.
\end{aligned}$$

Using separation of variables, S is rewritten as the product of single-variable functions:

$$S(x, y) = \boldsymbol{X}(x)\boldsymbol{Y}(y).$$

Applying the negative Laplacian results in the following equation:

$$\boldsymbol{X}''_{xx}\boldsymbol{Y} + \boldsymbol{X}\boldsymbol{Y}''_{yy} = -\lambda \boldsymbol{X}\boldsymbol{Y}.$$

Dividing by $\boldsymbol{X}\boldsymbol{Y}$ results in:

$$\frac{\boldsymbol{X}''_{xx}}{\boldsymbol{X}} + \frac{\boldsymbol{Y}''_{yy}}{\boldsymbol{Y}} = -\lambda.$$

Moving the $\boldsymbol{Y}$ fraction to the right side makes it obvious that the left side is only dependent on x and the right side is only dependent on y. For the equation to hold, then both the $\boldsymbol{X}$ fraction and the $\boldsymbol{Y}$ fraction must be constant. Let

$$\begin{aligned} -\frac{\boldsymbol{X}''}{\boldsymbol{X}} &= \mu, \\ -\frac{\boldsymbol{Y}''}{\boldsymbol{Y}} &= \nu, \\ \lambda &= \mu + \nu. \end{aligned}$$

Before solving this equation system, a quick detour to show that the negative Laplacian is a positive semi-definite operator is made. Consider the negative Laplacian in two dimensions on the domain Ω. Furthermore let $u(x,y)$, $v(x,y)$ be arbitrary twice differentiable, two dimensional functions on the domain. Let $(u|\ v)$ be the inner product defined as:

$$(u|v) = \int_{\Omega} uv\, d\Omega.$$

Then the negative Laplacian is positive semi-definite with respect to the inner product iff

$$(u| - \Delta u) \geq 0 \quad \forall (x,y) \in \Omega.$$

Using Green's first identity [17] (p. 403)

$$\begin{aligned} (u| - \Delta u) &= -\int_{\Omega} u \Delta u\, d\Omega \\ &= -\oint_{\partial\Omega} u(\nabla u \cdot \boldsymbol{n})\, dS + \int_{\Omega} \nabla u \cdot \nabla u\, d\Omega \\ &= -\oint_{\partial\Omega} u(\nabla u \cdot \boldsymbol{n})\, dS + \int_{\Omega} ||\nabla u||_2^2\, d\Omega. \end{aligned}$$

Since u is zero on the boundary, the contour integral is zero. Therefore

$$(u| - \Delta u) = \int_{\Omega} ||\nabla u||_2^2\, d\Omega \geq 0. \tag{A.2}$$

So the negative Laplacian is a positive semi-definite operator and hence must have non-negative eigenvalues. Two cases are possible, the eigenvalue is either zero or positive. First, assume that the eigenvalue is zero. The equation system is then transformed to:

$$-\frac{\boldsymbol{X}''}{\boldsymbol{X}} = \mu \iff X(x) = A_1\cos(\sqrt{\mu}x) + A_2\sin(\sqrt{\mu}x)$$
$$-\frac{\boldsymbol{Y}''}{\boldsymbol{Y}} = -\mu \iff Y(y) = B_1\cosh(\sqrt{\mu}x) + B_2\sinh(\sqrt{\mu}x)$$

However, applying the boundary conditions grants that $A_1 = B_1 = \mu = 0$, which yields that

$$\boldsymbol{X}(x) = 0,$$

making $S(x, y) = 0$. This is a solution to the equation, but a rather useless one. Therefore, consider the second case with $\lambda > 0$:

$$-\frac{\boldsymbol{X}''_{xx}}{\boldsymbol{X}} = \mu.$$
$$-\frac{\boldsymbol{Y}''_{yy}}{\boldsymbol{Y}} = \nu.$$
$$\lambda = \mu + \nu.$$

This equation system has the solutions:

$$\boldsymbol{X}(x) = A_1 cos(\sqrt{\mu}x) + A_2 sin(\sqrt{\mu}x).$$
$$\boldsymbol{Y}(y) = B_1 cos(\sqrt{\nu}y) + B_2 sin(\sqrt{\nu}y).$$
$$\lambda = \mu + \nu.$$

Inserting the boundary conditions results in:

$$\boldsymbol{X}(0) = A_1 cos(0) + A_2 sin(0) = 0 \implies A_1 = 0.$$
$$\boldsymbol{X}(L) = A_2 sin(\sqrt{\mu}L) = 0 \implies \mu = \left(\frac{m\pi}{L}\right)^2, \ m \in \mathbb{Z}^+.$$
$$\boldsymbol{Y}(0) = B_1 cos(0) + B_2 sin(0) = 0 \implies B_1 = 0.$$
$$\boldsymbol{Y}(H) = B_2 sin(\sqrt{\nu}H) = 0 \implies \nu = \left(\frac{n\pi}{H}\right)^2, \ n \in \mathbb{Z}^+.$$

Thus, $s_{m,n}(x,y) = A_2B_2sin(\frac{m\pi}{L}x)sin(\frac{n\pi}{H}y)$ are eigenfunctions to the Laplacian. However, due to ambiguity of eigenfunctions, the scaling can be chosen to be any (non-zero) value. To make the eigenfunction $s_{1,1}(x,y)$ bowl shaped, $A_2B_2 = -1$ is chosen. Hence the eigenfunctions are simply $s_{m,n}(x,y) = -sin(\frac{m\pi}{L}x)sin(\frac{n\pi}{H}y)$. Finally by inserting the resulting μ and ν into the eigenvalue expression, the full solution is written as:

$$\begin{aligned} s_{m,n}(x,y) &= -sin\left(\frac{m\pi}{L}x\right) sin\left(\frac{n\pi}{H}y\right). \\ \lambda_{m,n} &= \pi^2\left[\left(\frac{m}{L}\right)^2 + \left(\frac{n}{H}\right)^2\right]. \\ m,n &\in \mathbb{Z}^+. \end{aligned}$$

A.2 Principal Component Analysis

Let $\mathcal{S} = \{\boldsymbol{x}_i\}_{i=1}^m$ be a data set containing data vectors $\boldsymbol{x}_i \in \mathbf{R}^n$. Then create a data matrix:

$$\boldsymbol{X'} = \begin{bmatrix} \boldsymbol{x}_1^\top \\ \boldsymbol{x}_2^\top \\ \vdots \\ \boldsymbol{x}_m^\top \end{bmatrix}.$$

Calculate the mean for each column of the data matrix:

$$\hat{\boldsymbol{\mu}} = \begin{bmatrix} \frac{1}{m}\sum_{i=1}^m x_{i,1} \\ \frac{1}{m}\sum_{i=1}^m x_{i,2} \\ \vdots \\ \frac{1}{m}\sum_{i=1}^m x_{i,n} \end{bmatrix}.$$

Centre the data matrix by subtracting the means, i.e:

$$\boldsymbol{X}_{i,j} = \boldsymbol{X'}_{i,j} - \hat{\mu}_i.$$

Principal component analysis is meant to identity the features that are most prominent in the data set. The way this is achieved is to find which features in the data that carry the most variance [11]. Construct the covariance matrix $\boldsymbol{\Sigma}$ (recall that the data matrix is centred, hence the mean does not need to be subtracted):

$$\boldsymbol{\Sigma} = \frac{1}{m-1}\boldsymbol{X}^\top \boldsymbol{X}.$$

Since $\boldsymbol{\Sigma}$ is obviously symmetric and

$$\begin{aligned} y^\top \boldsymbol{\Sigma} y &= \frac{1}{m-1} y^\top \boldsymbol{X}^\top \boldsymbol{X} y \\ &= \frac{1}{m-1} (\boldsymbol{X}y)^\top \boldsymbol{X}y \\ &= \frac{1}{m-1} ||\boldsymbol{X}y||_2^2 \geq 0 \quad \forall y \neq \vec{0}, \end{aligned}$$

the covariance matrix is a symmetric positive semi-definite matrix, and hence has non-negative eigenvalues. Let the eigenvalues be sorted in descending order. Then the feature that maximises the variance is the eigenvalue corresponding to the first (largest) eigenvalue. Each consecutive eigenvalue-eigenvector pair contributes less and less to the total variance. The covariance matrix $\boldsymbol{\Sigma}$ is expensive to calculate, but there is a shortcut. Use the singular value decomposition on the data matrix, $\boldsymbol{X} = \boldsymbol{USV}^\top$. Thus the covariance matrix can be rewritten as:

$$\boldsymbol{\Sigma} = \frac{1}{m-1}\boldsymbol{X}^\top \boldsymbol{X} = \frac{1}{m-1}\left(\boldsymbol{USV}^\top\right)^\top \left(\boldsymbol{USV}^\top\right) = \boldsymbol{V}\left(\frac{1}{m-1}\boldsymbol{S}^2\right)\boldsymbol{V}^\top.$$

This is the eigenvalue factorisation of the covariance matrix. Hence the eigenvalues and eigenvectors of the covariance matrix can be extracted from the singular value decomposition of the data matrix. So the scheme to find the features that contribute the most to the total variance in descending order is the following:

1. Construct unstandardised data matrix.
2. Standardise the matrix by subtracting the column-wise means and dividing by the column-wise standard deviation.
3. Decompose the standardised data matrix using the singular value decomposition.
4. Let $\{\sigma_i\}_{i=1}^{min(m,\, n)}$ be the diagonal entries of $\boldsymbol{S}$ and $\{v_i\}_{i=1}^{n}$ be the column vectors of V.
5. The eigenvalues of the covariance matrix are extracted as: $\lambda_i = \frac{\sigma_i^2}{m-1}$.
6. The feature vector v_i then contributes $100 \times \frac{\lambda_i}{\sum_j \lambda_j}\%$ to the total variance of the data matrix.

A.3 Umeyama's algorithm - Finding the optimal similarity transformation

Consider the minimisation problem:

$$\begin{aligned} &\underset{\boldsymbol{R},\boldsymbol{t},s}{\operatorname{argmin}} \sum_{i=1}^{N} \|\boldsymbol{y}_i - \boldsymbol{t} - s\boldsymbol{R}\boldsymbol{x}_i\|_2^2 \\ &s.t. \quad \boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{I}, \end{aligned}$$

where $\{(\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ are points corresponding to each other. This problem is solved using Umeyama's algorithm [19], which is an extension of Kabsch algorithm [6]:

1. Calculate mean vectors: $\hat{\boldsymbol{\mu}}_x = \frac{1}{N}\sum_{i=1}^N \boldsymbol{x}_i, \quad \hat{\boldsymbol{\mu}}_y = \frac{1}{N}\sum_{i=1}^N \boldsymbol{y}_i$.
2. Calculate standard deviation: $\hat{\sigma}_x^2 = \frac{1}{N}\sum_{i=1}^N ||\boldsymbol{x}_i - \boldsymbol{\mu}_x||_2^2$.
3. Construct cross-covariance matrix: $\boldsymbol{\Sigma}_{x,y} = \frac{1}{N}\sum_{i=1}^N (\boldsymbol{y}_i - \hat{\boldsymbol{\mu}}_y)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_x)^\top$.
4. Perform singular value decomposition: $\boldsymbol{\Sigma}_{x,y} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$.
5. Define: $\boldsymbol{D} = \text{diag}(\boldsymbol{1},\ \det(\boldsymbol{U}\boldsymbol{V}^\top))$.
6. Set: $\boldsymbol{R}_* := \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$.
7. Set: $s_* := \frac{1}{\hat{\sigma}_x^2}\,\text{tr}(\boldsymbol{S}\boldsymbol{D})$.
8. Set: $\boldsymbol{t}_* := \hat{\mu}_y - s_*\boldsymbol{R}_*\hat{\mu}_x$.
9. Transformation matrix is then: $\boldsymbol{T} = \begin{bmatrix} s_*\boldsymbol{R}_* & \boldsymbol{t}_* \\ \boldsymbol{0}^\top & 1 \end{bmatrix}$.

A.4 Estimation of gradient using a neural network and backpropagation

If a neural network is trained to approximate a function, it can also be used to estimate the partial derivatives of its inputs. Consider the fully connected neural network in figure A.1.

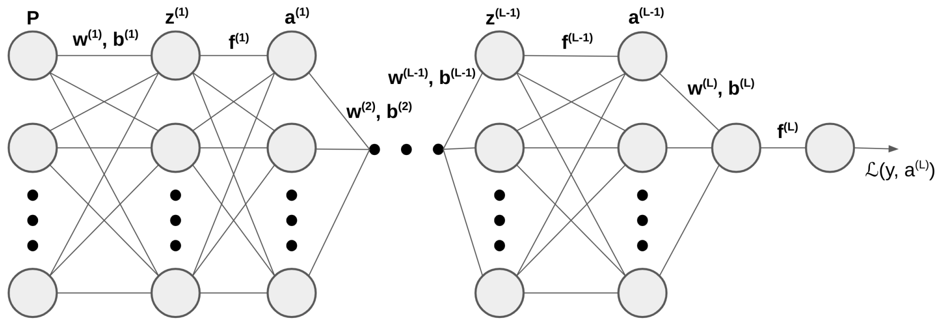


Figure A.1: Fully connected sequential neural network architecture.

where the superscript denotes the layer indices and the network has L layers.

Assume that the input layer consists of p nodes and that the ℓ:th layer has $n^{(\ell)}$ nodes. Let $z_i^{(\ell)}$ and $a_i^{(\ell)}$ denote the pre-activation value and post-activation value respectively for the i:th neuron in the ℓ:th layer in the network. In vector form, the pre-activation and post-activation values can be written as:

$$\boldsymbol{z}^{(\ell)} = \begin{bmatrix} z_1^{(\ell)} \\ z_2^{(\ell)} \\ \vdots \\ z_{n^{(\ell)}}^{(\ell)} \end{bmatrix}, \quad \boldsymbol{a}^{(\ell)} = \begin{bmatrix} a_1^{(\ell)} \\ a_2^{(\ell)} \\ \vdots \\ a_{n^{(\ell)}}^{(\ell)} \end{bmatrix}.$$

Denote $w_{i,j}^{(\ell)}$ as the weight going from the i:th node in layer $\ell - 1$ to the j:th node in layer ℓ. Let $\boldsymbol{W}^{(\ell)}$ be defined as:

$$\boldsymbol{W}^{(\ell)} = \begin{bmatrix} w_{1,1}^{(\ell)} & w_{1,2}^{(\ell)} & \cdots & w_{1,n^{(\ell)}}^{(\ell)} \\ w_{2,1}^{(\ell)} & w_{2,2}^{(\ell)} & \cdots & w_{2,n^{(\ell)}}^{(\ell)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n^{(\ell-1)},1}^{(\ell)} & w_{n^{(\ell-1)},2}^{(\ell)} & \cdots & w_{n^{(\ell-1)},n^{(\ell)}}^{(\ell)} \end{bmatrix}.$$

Furthermore, let $b^{(\ell)}$ denote the bias in layer ℓ. The pre-activation vector $\boldsymbol{z}^{(\ell)}$ can be written as:

$$\boldsymbol{z}^{(\ell)} = \boldsymbol{W}^{(\ell)} \boldsymbol{a}^{(\ell-1)} + b^{(\ell)}. \tag{A.3}$$

Let $f^{(\ell)}(x)$ be the activation function that maps the pre-activation values to the post-activation values in the ℓ:th layer, i.e:

$$\boldsymbol{a}^{(\ell)} = f^{(\ell)}(\boldsymbol{z}^{(\ell)}), \tag{A.4}$$

$$f^{(0)}(\boldsymbol{z}^{(0)}) = \boldsymbol{a}^{(0)} = \boldsymbol{p}, \tag{A.5}$$

where $\boldsymbol{p}$ is the input.

Thereby the output vector $\boldsymbol{a}^{(L)}$ can be calculated recursively:

$$\begin{aligned}
\boldsymbol{a}^{(L)} &= f^{(L)}(\boldsymbol{z}^{L}) \\
&= f^{(L)}(\boldsymbol{W}^{(L)}\boldsymbol{a}^{(L-1)} + b^{(L)}) \\
&= f^{(L)}(\boldsymbol{W}^{(L)}f^{(L-1)}(\boldsymbol{z}^{(L-1)}) + b^{(L)}) \\
&= f^{(L)}(\boldsymbol{W}^{(L)}f^{(L-1)}(\boldsymbol{W}^{(L-1)}\boldsymbol{a}^{(L-2)} + b^{(L-1)}) + b^{(L)}) \\
&= f^{(L)}(\boldsymbol{W}^{(L)}f^{(L-1)}(\boldsymbol{W}^{(L-1)}f^{(L-2)}(...f^{(0)}(\boldsymbol{z}^{(0)})...) + b^{(L-1)}) + b^{(L)}) \\
&= f^{(L)}(\boldsymbol{W}^{(L)}f^{(L-1)}(\boldsymbol{W}^{(L-1)}f^{(L-2)}(...\boldsymbol{p}...) + b^{(L-1)}) + b^{(L)}).
\end{aligned}$$

Let $\mathcal{L}(\boldsymbol{y}, \boldsymbol{a}^{(L)})$ be a loss function where y is the target vector and $\boldsymbol{a}^{(L)}$ is the output from the network. This is the function that the network tries to minimise. However, to use the neural network to get a gradient to update the parameters with, it is necessary to get find how the input parameters affects the loss function. This is done using the backpropagation algorithm [15].

The task at hand is hence to calculate $\nabla_{\boldsymbol{p}}\mathcal{L}$. The following notation is used (c, v and u are only used to showcase the notation and not used in any calculations):

$$\frac{dc}{d\boldsymbol{v}} := \begin{bmatrix} \frac{dc}{dv_1} \\ \frac{dc}{dv_2} \\ \vdots \\ \frac{dc}{dv_n} \end{bmatrix}, \quad \frac{d\boldsymbol{u}}{d\boldsymbol{v}} := \begin{bmatrix} \frac{du_1}{dv_1} & \frac{du_1}{dv_1} & \cdots & \frac{du_1}{dv_n} \\ \frac{du_2}{dv_1} & \frac{du_2}{dv_1} & \cdots & \frac{du_2}{dv_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{du_m}{dv_1} & \frac{du_m}{dv_2} & \cdots & \frac{du_m}{dv_n} \end{bmatrix}.$$

Note that c is some function which takes a vector $\boldsymbol{v}$ as input and produces a scalar output, and that u is function which takes a vector $\boldsymbol{v}$ as input and produces a vector as output.

Using the chain-rule on $\nabla_{\boldsymbol{p}}\mathcal{L}$ yields:

$$\frac{d\mathcal{L}}{d\boldsymbol{p}} = \left(\frac{d\boldsymbol{a}^{(L)}}{d\boldsymbol{p}}\right)^{\top} \frac{d\mathcal{L}}{d\boldsymbol{a}^{(L)}}. \tag{A.6}$$

The equations (A.3) and (A.4) yield:

$$\begin{aligned}
\boldsymbol{a}^{(\ell)} = f^{(\ell)}(\boldsymbol{z}^{(\ell)}) &\implies \frac{d\boldsymbol{a}^{(\ell)}}{d\boldsymbol{p}} = \frac{df^{(\ell)}(\boldsymbol{z}^{(\ell)})}{d\boldsymbol{z}^{(\ell)}}\frac{d\boldsymbol{z}^{(\ell)}}{d\boldsymbol{p}}, \\
\boldsymbol{z}^{(\ell)} = \boldsymbol{W}^{(\ell)}\boldsymbol{a}^{(\ell-1)} + b^{(\ell)} &\implies \frac{d\boldsymbol{z}^{(\ell)}}{d\boldsymbol{p}} = \boldsymbol{W}^{(\ell)}\frac{d\boldsymbol{a}^{(\ell-1)}}{d\boldsymbol{p}}, \\
\boldsymbol{p} = \boldsymbol{a}^{(0)} = f(\boldsymbol{z}^{(0)}) &\implies \frac{df(\boldsymbol{z}^{(0)})}{d\boldsymbol{p}} = \frac{d\boldsymbol{a}^{(0)}}{d\boldsymbol{p}} = \frac{d\boldsymbol{p}}{d\boldsymbol{p}} = \boldsymbol{I}.
\end{aligned}$$

Hence a recursive formula has been found:

$$\begin{aligned}
\frac{d\boldsymbol{a}^{(L)}}{d\boldsymbol{p}} &= \frac{df^{(L)}(\boldsymbol{z}^{(L)})}{d\boldsymbol{z}^{(L)}}\frac{d\boldsymbol{z}^{(L)}}{d\boldsymbol{p}} \\
&= \frac{df^{(L)}(\boldsymbol{z}^{(L)})}{d\boldsymbol{z}^{(L)}}\boldsymbol{W}^{(L)}\frac{d\boldsymbol{a}^{(L-1)}}{d\boldsymbol{p}} \\
&= \frac{df^{(L)}(\boldsymbol{z}^{(L)})}{d\boldsymbol{z}^{(L)}}\boldsymbol{W}^{(L)}\frac{df^{(L-1)}(\boldsymbol{z}^{(L-1)})}{d\boldsymbol{z}^{(L-1)}}\frac{d\boldsymbol{z}^{(L-1)}}{d\boldsymbol{p}} \\
&= \frac{df^{(L)}(\boldsymbol{z}^{(L)})}{d\boldsymbol{z}^{(L)}}\boldsymbol{W}^{(L)}\frac{df^{(L-1)}(\boldsymbol{z}^{(L-1)})}{d\boldsymbol{z}^{(L-1)}}\boldsymbol{W}^{(L-1)}\frac{d\boldsymbol{a}^{(L-2)}}{d\boldsymbol{p}} \\
&= \dots \\
&= \left[\prod_{\ell=L}^{1}\frac{df^{(\ell)}(\boldsymbol{z}^{(\ell)})}{d\boldsymbol{z}^{(\ell)}}\boldsymbol{W}^{(\ell)}\right]\frac{d\boldsymbol{p}}{d\boldsymbol{p}} \\
&= \prod_{\ell=L}^{1}\frac{df^{(\ell)}(\boldsymbol{z}^{(\ell)})}{d\boldsymbol{z}^{(\ell)}}\boldsymbol{W}^{(\ell)}. \qquad \text{(A.7)}
\end{aligned}$$

Inserting the result from equation (A.7) into equation (A.6) grants:

$$\begin{aligned}
\frac{d\mathcal{L}}{d\boldsymbol{p}} &= \left[\prod_{\ell=L}^{1}\frac{df^{(\ell)}(\boldsymbol{z}^{(\ell)})}{d\boldsymbol{z}^{(\ell)}}\boldsymbol{W}^{(\ell)}\right]^{\top}\frac{d\mathcal{L}}{d\boldsymbol{a}^{(L)}} \\
&= \left[\prod_{\ell=1}^{L}\left(\frac{df^{(\ell)}(\boldsymbol{z}^{(\ell)})}{d\boldsymbol{z}^{(\ell)}}\boldsymbol{W}^{(\ell)}\right)^{\top}\right]\frac{d\mathcal{L}}{d\boldsymbol{a}^{(L)}}. \qquad \text{(A.8)}
\end{aligned}$$

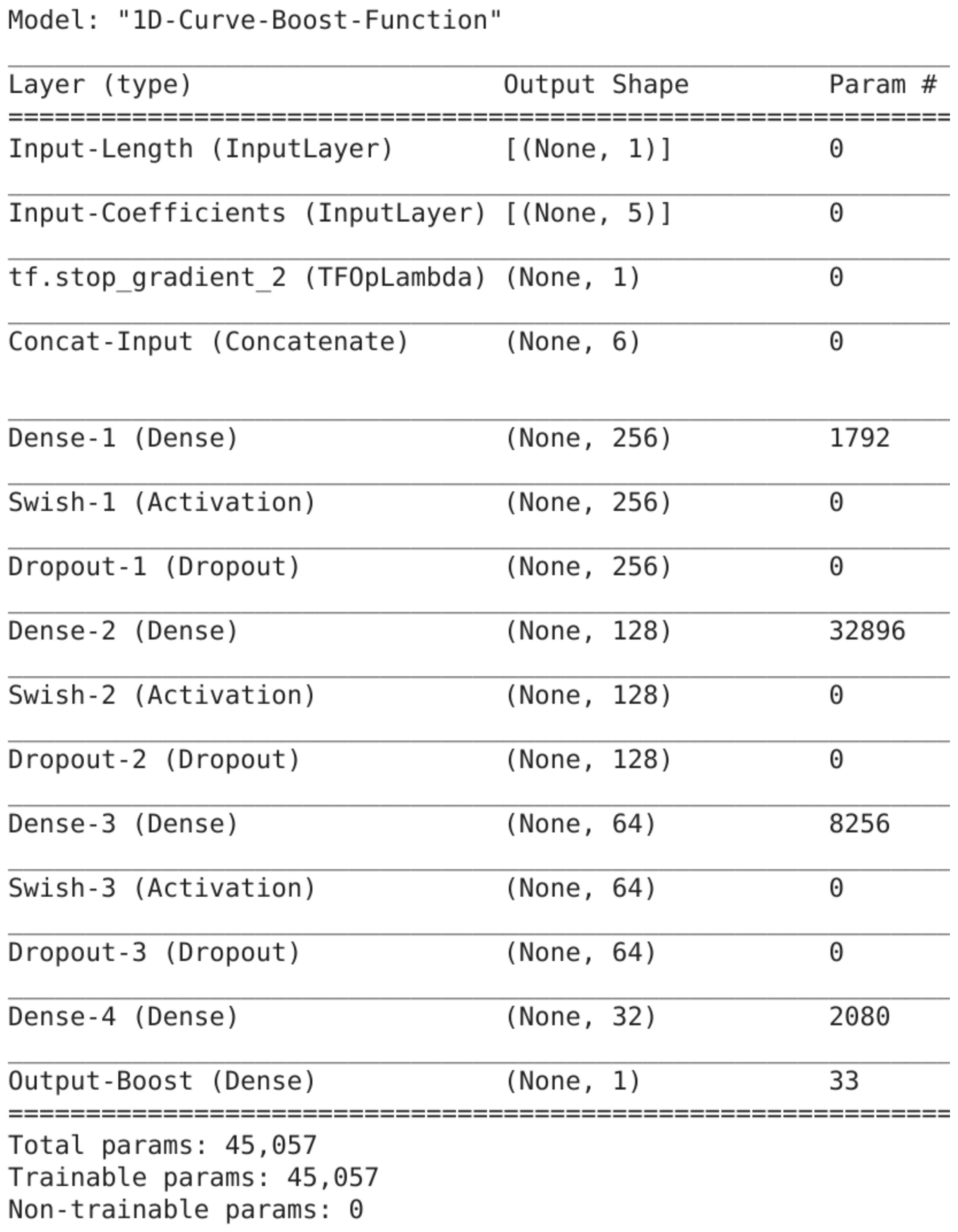
The result in equation (A.8) is the gradient of the loss function with respect to the input $\boldsymbol{p}$.

Appendix B

Model Architectures

Appendix containing the architecture of all neural network models used in this the-
sis.

B.1 One-dimensional curve boost function architecture



```
Model: "1D-Curve-Boost-Function"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input-Length (InputLayer)    [(None, 1)]               0
_________________________________________________________________
Input-Coefficients (InputLayer) [(None, 5)]            0
_________________________________________________________________
tf.stop_gradient_2 (TFOpLambda) (None, 1)              0
_________________________________________________________________
Concat-Input (Concatenate)   (None, 6)                 0

_________________________________________________________________
Dense-1 (Dense)              (None, 256)               1792
_________________________________________________________________
Swish-1 (Activation)         (None, 256)               0
_________________________________________________________________
Dropout-1 (Dropout)          (None, 256)               0
_________________________________________________________________
Dense-2 (Dense)              (None, 128)               32896
_________________________________________________________________
Swish-2 (Activation)         (None, 128)               0
_________________________________________________________________
Dropout-2 (Dropout)          (None, 128)               0
_________________________________________________________________
Dense-3 (Dense)              (None, 64)                8256
_________________________________________________________________
Swish-3 (Activation)         (None, 64)                0
_________________________________________________________________
Dropout-3 (Dropout)          (None, 64)                0
_________________________________________________________________
Dense-4 (Dense)              (None, 32)                2080
_________________________________________________________________
Output-Boost (Dense)         (None, 1)                 33
=================================================================
Total params: 45,057
Trainable params: 45,057
Non-trainable params: 0
```

Figure B.1: Boost function network summary.

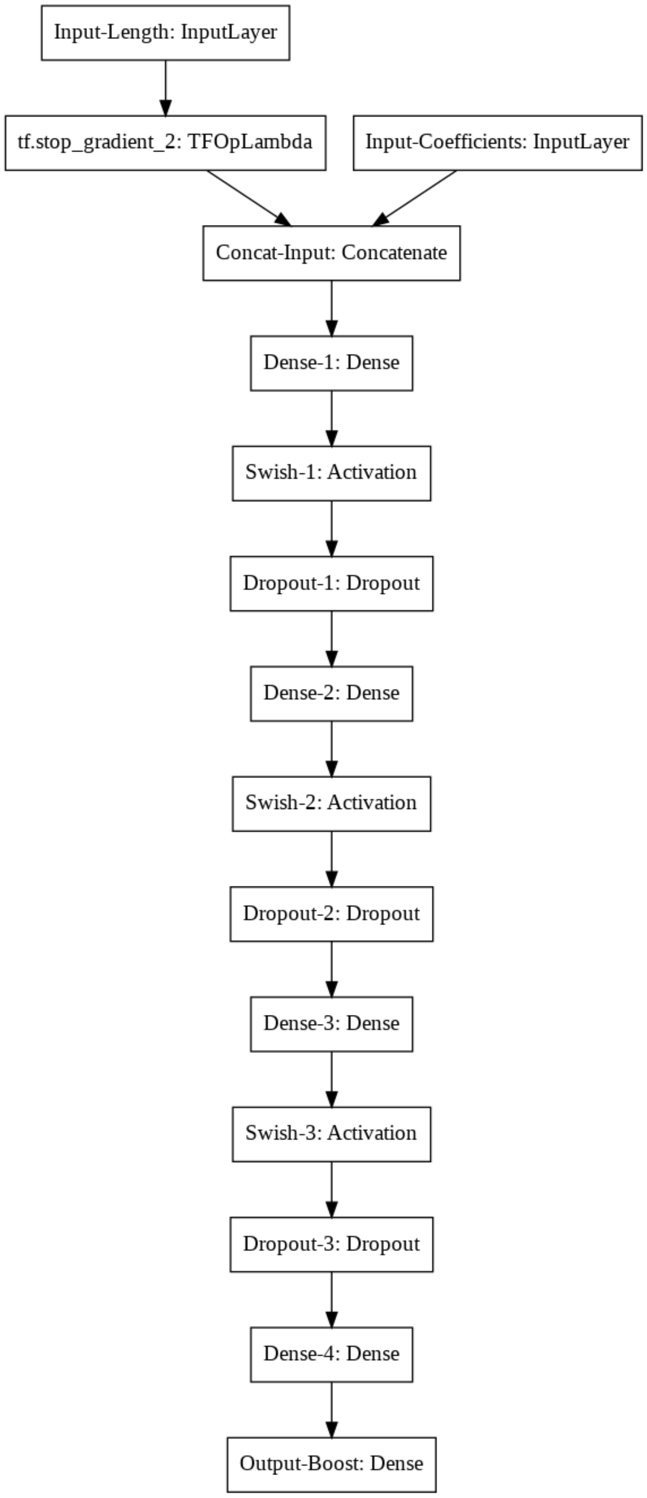


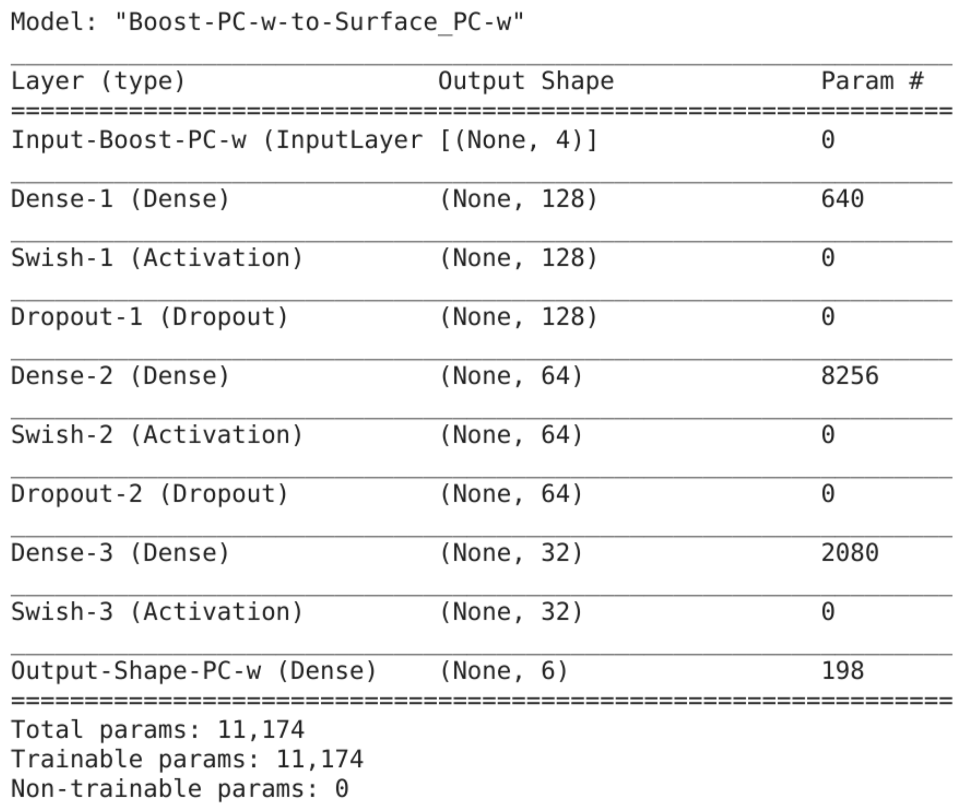
Figure B.2: Boost function network architecture.

- Input 1: 5x1 vector (5 curve coefficients).

- Input 2: 1x1 vector with stopped gradient (Length of curve) .
- Output: 1x1 vector (Boost for a single curve).
- Optimiser: Adam.
- Loss function: MSE of target and predicted boost.

The network consists of four dense layers (including output layer), swish activation [14] after each hidden layer except last one and 20 percent dropout after each activation. The input is split into two separate branches, making it easier to stop the backpropagation into the 'Length' branch and only calculate the gradient w.r.t to the curve coefficients without any contribution from the length (which should of course be constant and not be considered when calculating the gradient). The first branch takes five scaled curve coefficients describing the fourth degree polynomial fit of the curve $c(x)$ as input, and the second branch takes a scaled length as input. The output is simply the predicted boost for the curve described by the curve coefficients.

B.2 Boost PC weights to Surface PC weights architecture



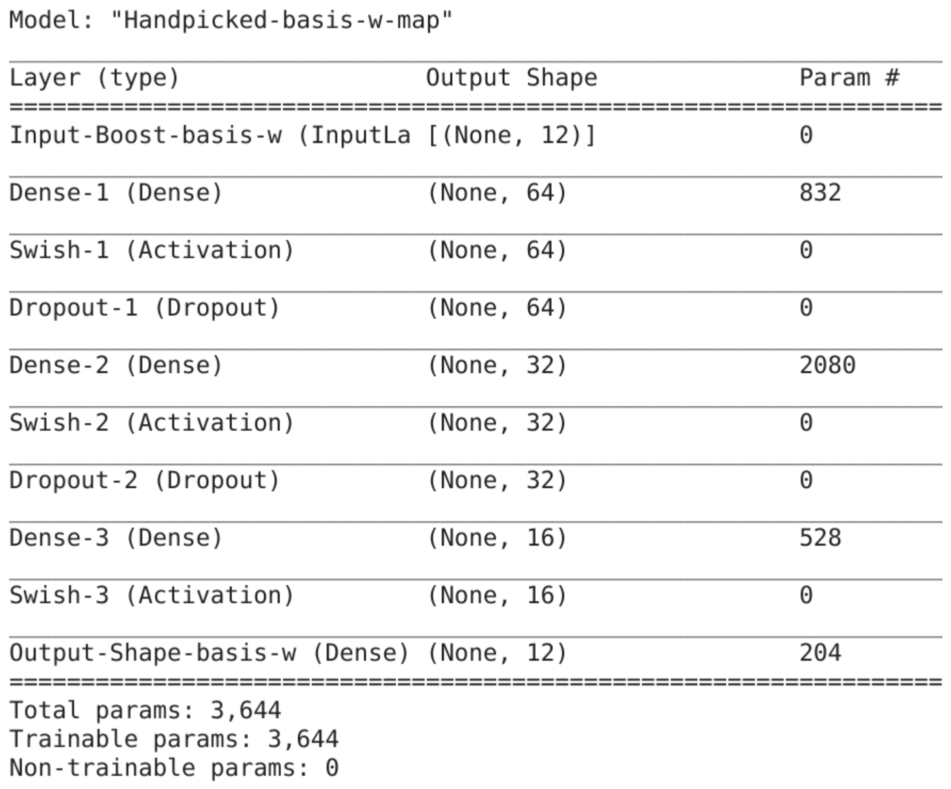
```
Model: "Boost-PC-w-to-Surface_PC-w"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input-Boost-PC-w (InputLayer [(None, 4)]               0
_________________________________________________________________
Dense-1 (Dense)              (None, 128)               640
_________________________________________________________________
Swish-1 (Activation)         (None, 128)               0
_________________________________________________________________
Dropout-1 (Dropout)          (None, 128)               0
_________________________________________________________________
Dense-2 (Dense)              (None, 64)                8256
_________________________________________________________________
Swish-2 (Activation)         (None, 64)                0
_________________________________________________________________
Dropout-2 (Dropout)          (None, 64)                0
_________________________________________________________________
Dense-3 (Dense)              (None, 32)                2080
_________________________________________________________________
Swish-3 (Activation)         (None, 32)                0
_________________________________________________________________
Output-Shape-PC-w (Dense)    (None, 6)                 198
=================================================================
Total params: 11,174
Trainable params: 11,174
Non-trainable params: 0
```

Figure B.3: Summary of boost PC weights to Surface PC weights mapping network.

- Input: 4x1 vector (4 boost principal component weights).
- Output: 6x1 vector (6 surface principal component weights).
- Optimiser: Adam.
- Loss function: MSE between target and predicted surface principal component weights.

The network consists of four our dense layers (including output layer), swish activation after each hidden layer and 20 percent dropout after each activation except the last one. The input takes four scaled boost matrix principal component weights and the output is six scaled surface principal component weights.

B.3 Handpicked basis weight mapping architecture



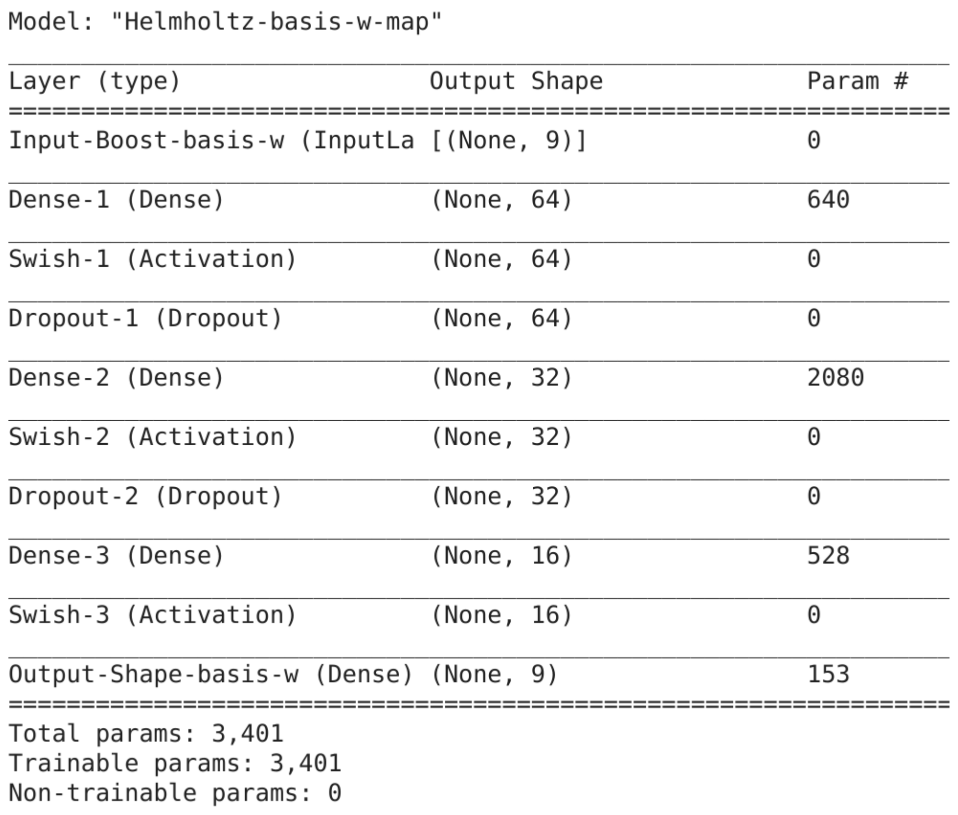
```
Model: "Handpicked-basis-w-map"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
Input-Boost-basis-w (InputLa [(None, 12)]              0         
_________________________________________________________________
Dense-1 (Dense)              (None, 64)                832       
_________________________________________________________________
Swish-1 (Activation)         (None, 64)                0         
_________________________________________________________________
Dropout-1 (Dropout)          (None, 64)                0         
_________________________________________________________________
Dense-2 (Dense)              (None, 32)                2080      
_________________________________________________________________
Swish-2 (Activation)         (None, 32)                0         
_________________________________________________________________
Dropout-2 (Dropout)          (None, 32)                0         
_________________________________________________________________
Dense-3 (Dense)              (None, 16)                528       
_________________________________________________________________
Swish-3 (Activation)         (None, 16)                0         
_________________________________________________________________
Output-Shape-basis-w (Dense) (None, 12)                204       
=================================================================
Total params: 3,644
Trainable params: 3,644
Non-trainable params: 0
```

Figure B.4: Summary of handpicked basis weight mapping network.

- Input: 12x1 vector (12 boost basis weights).
- Output: 12x1 vector (12 surface basis weights).
- Optimiser: Adam.
- Loss function: MSE of target and predicted basis weights.

The network consists of four our dense layers (including output layer), swish activation after each hidden layer and 20 percent dropout after each activation except the last one. The input takes twelve scaled boost basis weights and the output is twelve scaled surface basis weights.

B.4 Helmholtz basis weight mapping architecture



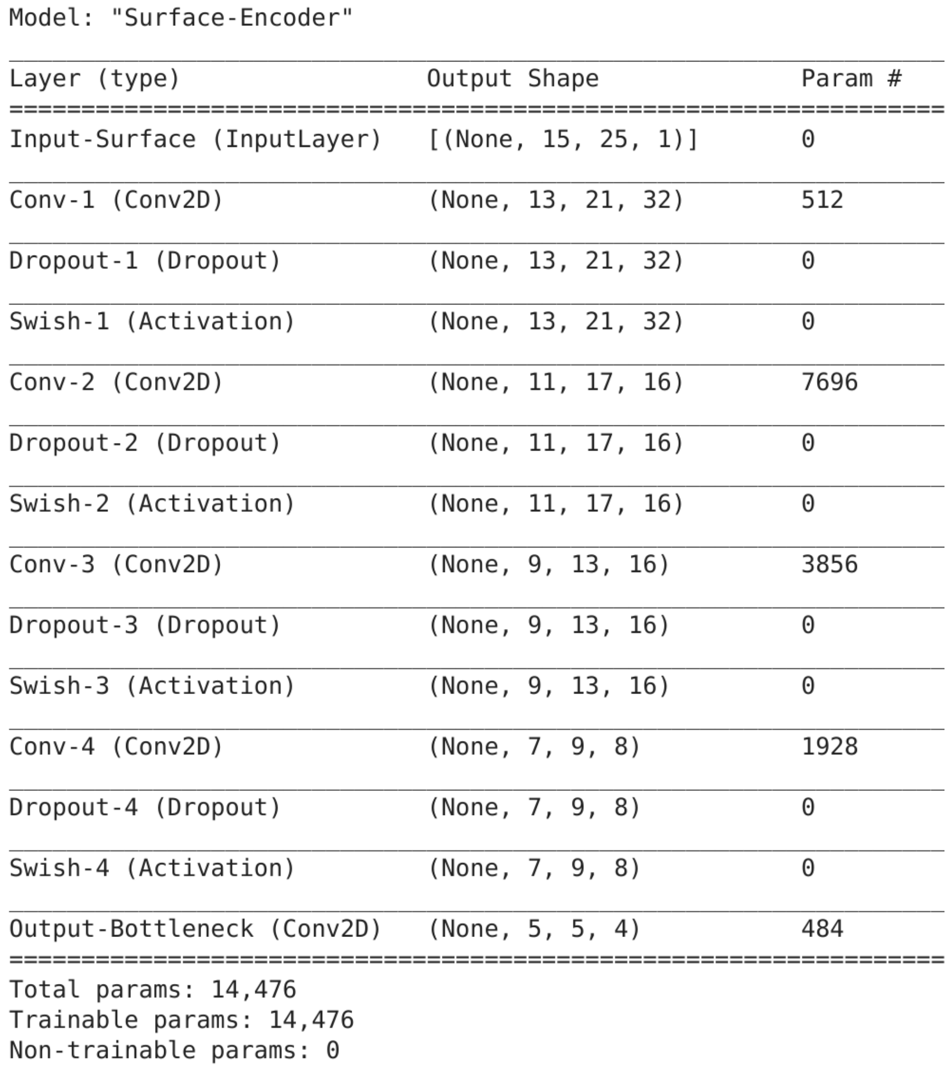
```
Model: "Helmholtz-basis-w-map"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input-Boost-basis-w (InputLa [(None, 9)]               0
_________________________________________________________________
Dense-1 (Dense)              (None, 64)                640
_________________________________________________________________
Swish-1 (Activation)         (None, 64)                0
_________________________________________________________________
Dropout-1 (Dropout)          (None, 64)                0
_________________________________________________________________
Dense-2 (Dense)              (None, 32)                2080
_________________________________________________________________
Swish-2 (Activation)         (None, 32)                0
_________________________________________________________________
Dropout-2 (Dropout)          (None, 32)                0
_________________________________________________________________
Dense-3 (Dense)              (None, 16)                528
_________________________________________________________________
Swish-3 (Activation)         (None, 16)                0
_________________________________________________________________
Output-Shape-basis-w (Dense) (None, 9)                 153
=================================================================
Total params: 3,401
Trainable params: 3,401
Non-trainable params: 0
```

Figure B.5: Summary of Helmholtz basis weight mapping network.

- Input: 9x1 vector (12 boost basis weights).
- Output: 9x1 vector (12 surface basis weights).
- Optimiser: Adam.
- Loss function: MSE of target and predicted basis weights.

The network consists of four our dense layers (including output layer), swish activation after each hidden layer and 20 percent dropout after each activation except the last one. The input takes nine scaled boost basis weights and the output is nine scaled surface basis weights.

B.5 Encoders and Decoder

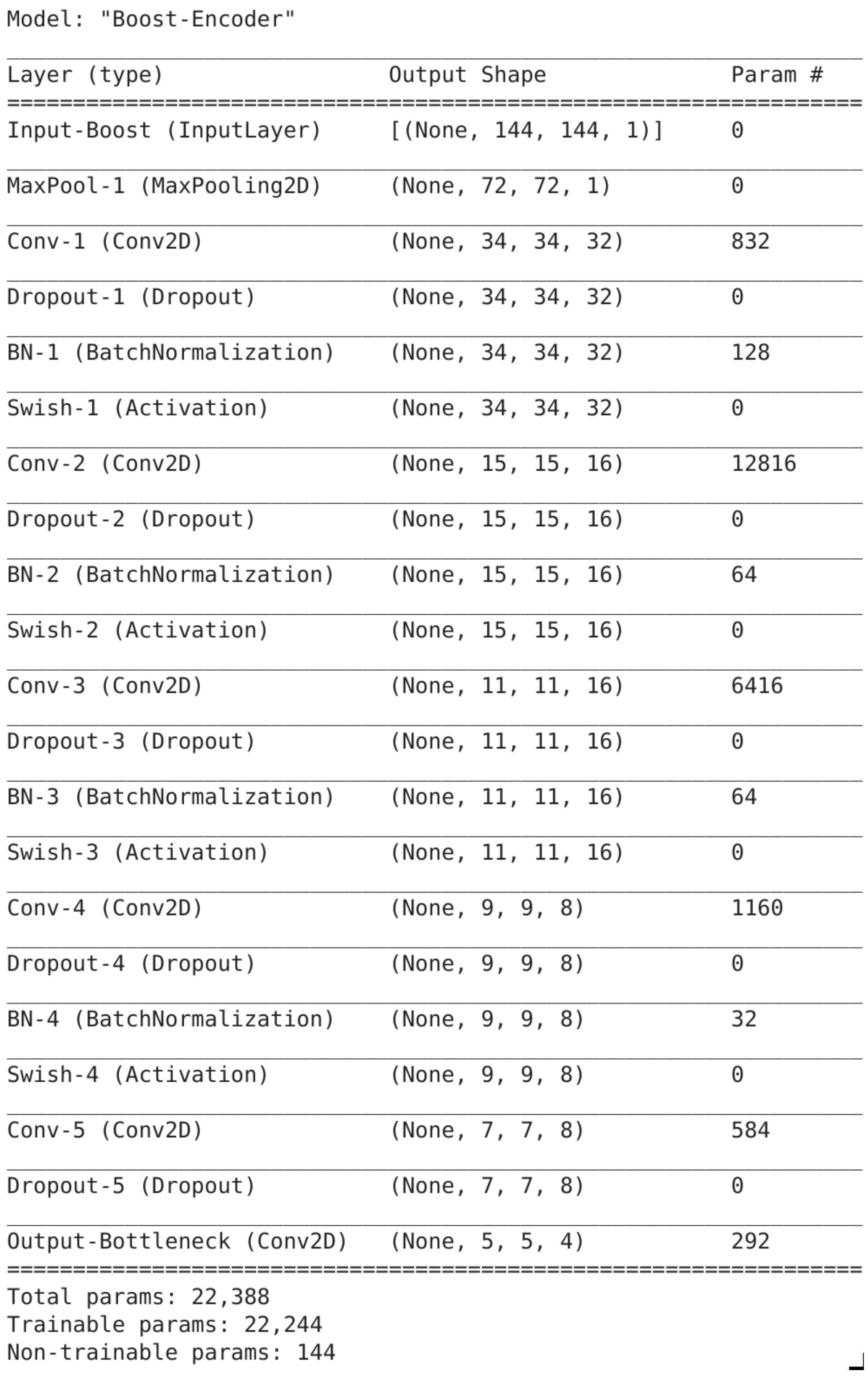


```
Model: "Surface-Encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input-Surface (InputLayer)   [(None, 15, 25, 1)]       0
_________________________________________________________________
Conv-1 (Conv2D)              (None, 13, 21, 32)        512
_________________________________________________________________
Dropout-1 (Dropout)          (None, 13, 21, 32)        0
_________________________________________________________________
Swish-1 (Activation)         (None, 13, 21, 32)        0
_________________________________________________________________
Conv-2 (Conv2D)              (None, 11, 17, 16)        7696
_________________________________________________________________
Dropout-2 (Dropout)          (None, 11, 17, 16)        0
_________________________________________________________________
Swish-2 (Activation)         (None, 11, 17, 16)        0
_________________________________________________________________
Conv-3 (Conv2D)              (None, 9, 13, 16)         3856
_________________________________________________________________
Dropout-3 (Dropout)          (None, 9, 13, 16)         0
_________________________________________________________________
Swish-3 (Activation)         (None, 9, 13, 16)         0
_________________________________________________________________
Conv-4 (Conv2D)              (None, 7, 9, 8)           1928
_________________________________________________________________
Dropout-4 (Dropout)          (None, 7, 9, 8)           0
_________________________________________________________________
Swish-4 (Activation)         (None, 7, 9, 8)           0
_________________________________________________________________
Output-Bottleneck (Conv2D)   (None, 5, 5, 4)           484
=================================================================
Total params: 14,476
Trainable params: 14,476
Non-trainable params: 0
```

Figure B.6: Summary of surface encoder architecture.

- Input: 15x25 matrix (Surface).
- Output: 5x5x4 tensor (Bottleneck).

The network consists of five convolutional layers (including output layer) with swish activation and 20 percent dropout after each hidden layer. The input takes a normalised surface and the output is a tensor containing information about the surface.



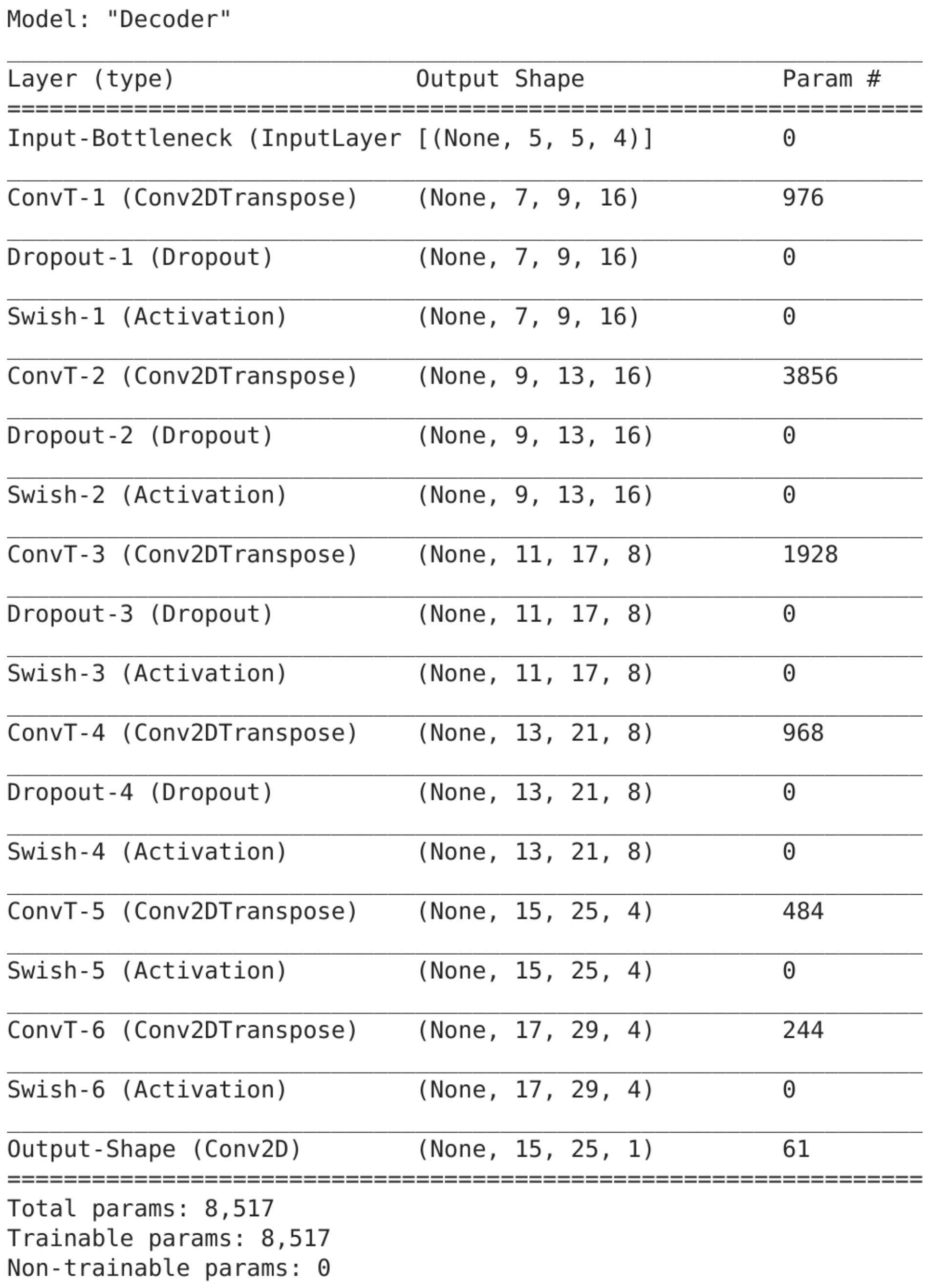
```
Model: "Boost-Encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input-Boost (InputLayer)     [(None, 144, 144, 1)]     0
_________________________________________________________________
MaxPool-1 (MaxPooling2D)     (None, 72, 72, 1)         0
_________________________________________________________________
Conv-1 (Conv2D)              (None, 34, 34, 32)        832
_________________________________________________________________
Dropout-1 (Dropout)          (None, 34, 34, 32)        0
_________________________________________________________________
BN-1 (BatchNormalization)    (None, 34, 34, 32)        128
_________________________________________________________________
Swish-1 (Activation)         (None, 34, 34, 32)        0
_________________________________________________________________
Conv-2 (Conv2D)              (None, 15, 15, 16)        12816
_________________________________________________________________
Dropout-2 (Dropout)          (None, 15, 15, 16)        0
_________________________________________________________________
BN-2 (BatchNormalization)    (None, 15, 15, 16)        64
_________________________________________________________________
Swish-2 (Activation)         (None, 15, 15, 16)        0
_________________________________________________________________
Conv-3 (Conv2D)              (None, 11, 11, 16)        6416
_________________________________________________________________
Dropout-3 (Dropout)          (None, 11, 11, 16)        0
_________________________________________________________________
BN-3 (BatchNormalization)    (None, 11, 11, 16)        64
_________________________________________________________________
Swish-3 (Activation)         (None, 11, 11, 16)        0
_________________________________________________________________
Conv-4 (Conv2D)              (None, 9, 9, 8)           1160
_________________________________________________________________
Dropout-4 (Dropout)          (None, 9, 9, 8)           0
_________________________________________________________________
BN-4 (BatchNormalization)    (None, 9, 9, 8)           32
_________________________________________________________________
Swish-4 (Activation)         (None, 9, 9, 8)           0
_________________________________________________________________
Conv-5 (Conv2D)              (None, 7, 7, 8)           584
_________________________________________________________________
Dropout-5 (Dropout)          (None, 7, 7, 8)           0
_________________________________________________________________
Output-Bottleneck (Conv2D)   (None, 5, 5, 4)           292
=================================================================
Total params: 22,388
Trainable params: 22,244
Non-trainable params: 144
```

Figure B.7: Summary of boost encoder architecture.

- Input: 144x144 matrix (Boost matrix).

- Output: 5x5x4 tensor (Bottleneck).

The network consists of a max pooling layer, six convolutional layers (including output layer) with 20 percent dropout after each hidden convolutional layer and swish activation after each hidden layer except the last one. The input takes a normalised boost matrix and the output is a tensor containing information about the boost matrix.



```
Model: "Decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input-Bottleneck (InputLayer [(None, 5, 5, 4)]         0
_________________________________________________________________
ConvT-1 (Conv2DTranspose)    (None, 7, 9, 16)          976
_________________________________________________________________
Dropout-1 (Dropout)          (None, 7, 9, 16)          0
_________________________________________________________________
Swish-1 (Activation)         (None, 7, 9, 16)          0
_________________________________________________________________
ConvT-2 (Conv2DTranspose)    (None, 9, 13, 16)         3856
_________________________________________________________________
Dropout-2 (Dropout)          (None, 9, 13, 16)         0
_________________________________________________________________
Swish-2 (Activation)         (None, 9, 13, 16)         0
_________________________________________________________________
ConvT-3 (Conv2DTranspose)    (None, 11, 17, 8)         1928
_________________________________________________________________
Dropout-3 (Dropout)          (None, 11, 17, 8)         0
_________________________________________________________________
Swish-3 (Activation)         (None, 11, 17, 8)         0
_________________________________________________________________
ConvT-4 (Conv2DTranspose)    (None, 13, 21, 8)         968
_________________________________________________________________
Dropout-4 (Dropout)          (None, 13, 21, 8)         0
_________________________________________________________________
Swish-4 (Activation)         (None, 13, 21, 8)         0
_________________________________________________________________
ConvT-5 (Conv2DTranspose)    (None, 15, 25, 4)         484
_________________________________________________________________
Swish-5 (Activation)         (None, 15, 25, 4)         0
_________________________________________________________________
ConvT-6 (Conv2DTranspose)    (None, 17, 29, 4)         244
_________________________________________________________________
Swish-6 (Activation)         (None, 17, 29, 4)         0
_________________________________________________________________
Output-Shape (Conv2D)        (None, 15, 25, 1)         61
=================================================================
Total params: 8,517
Trainable params: 8,517
Non-trainable params: 0
```

Figure B.8: Summary of decoder architecture.

- Input: 5x5x4 tensor (Bottleneck).
- Output: 15x25 tensor (Surface).

The network consists of six transposed convolutional layers which are used as up-sampling layers. Swish activation is performed after each hidden layer and 20 percent dropout is applied after each hidden transposed convolutional layer excepted for the last one. The output layer consist of a regular convolutional layer to reduce the output to the correct size. The shape is intentionally made too large at the sixth transposed convolutional layer to then be downsized again at the output layer to make the edge values more consistent. The input takes a bottleneck tensor and the output is a scaled prediction of the surface corresponding to the bottleneck tensor.

Simple Encoder-Decoder architecture

The Encoder-Decoder architecture is built by connecting the output of the boost encoder in figure B.7 to the input of the decoder in figure B.8. The model is trained by minimising the reconstruction loss, defined as the MSE between the target and predicted shape, with the default Adam optimiser.

Combined surface autoencoder and boost encoder

The Surface encoder and decoder are connected to produce an autoencoder and trained to minimise a loss consisting of the reconstruction loss, which is defined as the MSE of the target and predicted surface, as well as a weighted part of the loss used to trained the boost encoder. The boost encoder is simultaneously trained to minimise the MSE of its bottleneck output and the bottleneck output of the surface encoder. In words, it is trained to output a bottleneck that is similar to the one that the surface encoder outputs. By including the loss of the boost encoder in the loss of the autoencoder, it is incentivised to find an encoding that the boost matrix encoder is able to find as well.

Appendix C

Data Generation

The construction of the surfaces is split up into two steps. The first step is to construct intermediate surfaces using twelve basis surfaces which are displayed in figure C.1:

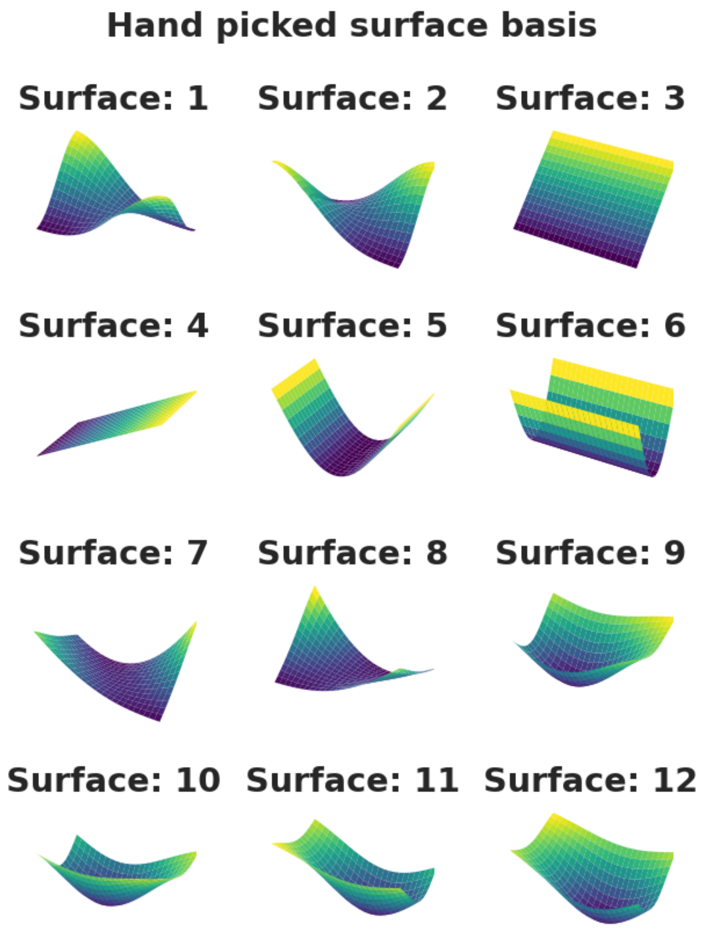


Figure C.1: Visualisation of twelve handpicked surfaces which constitutes the handpicked basis.

Random weights are sampled and the intermediate surfaces are constructed as a linear combination of the basis surfaces where the coefficients are the sampled weights. The weights are drawn from uniform distributions:

$w_1,\ w_2 \sim \mathcal{U}(-0.025, 0.475)$
$w_3,\ w_4 \sim \mathcal{U}(-0.025, 0.250)$
$w_5,\ w_6 \sim \mathcal{U}(-0.015, 0.585)$
$w_7,\ w_8 \sim \mathcal{U}(-0.050, 0.950)$
$w_9,\ w_{10},\ w_{11},\ w_{12} \sim \mathcal{U}(-0.020, 0.380)$

Table C.1: Sample distributions for the weights.

To add some extra variation, noise terms $\epsilon_i \sim \mathcal{N}(0,\ 0.1)$ are added to each weight w_i.

When a large set of these intermediate steps have been constructed, principal component analysis is performed on the set (see section 2.2.2). The dimensionality is then decreased by projecting the set onto the first eight principal components, which are displayed in figure C.2:

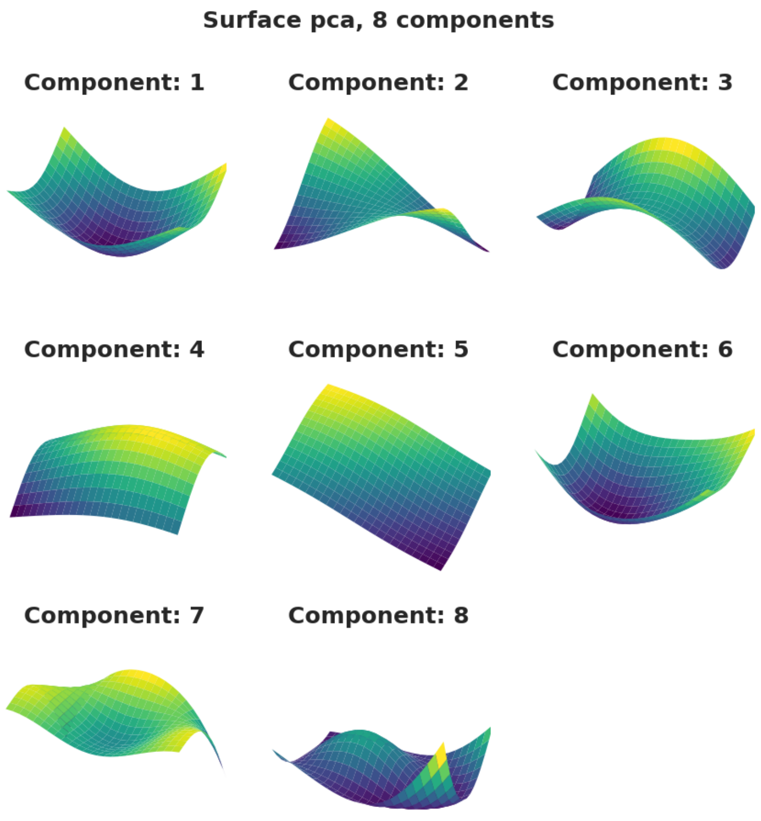


Figure C.2: Basis surfaces from the PCA.

The final surfaces are constructed as linear combinations of the principal components where the weight for each component is drawn uniformly between the smallest and largest value that was found for that component in the data set. Lastly, when these surfaces are constructed, a fourth degree polynomial surface is fitted to each shape and the boost simulation is run for each scanline for each fitted shape. Hence a method for generating reasonable surfaces with corresponding boost matrices is constructed and can be utilised to rapidly generate training data.

Appendix D

Reconstructions

Five surface reconstructions from the generated test set using the PCA basis alongside neural network mapping function:

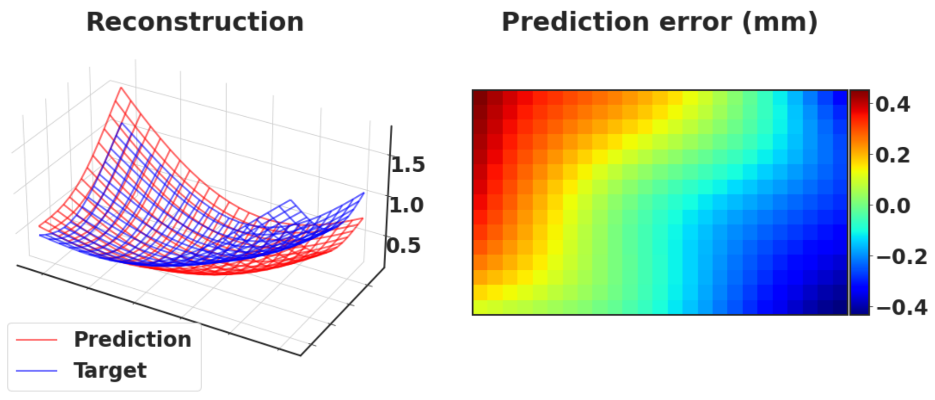


Figure D.1: Reconstruction of surface from the generated test set.

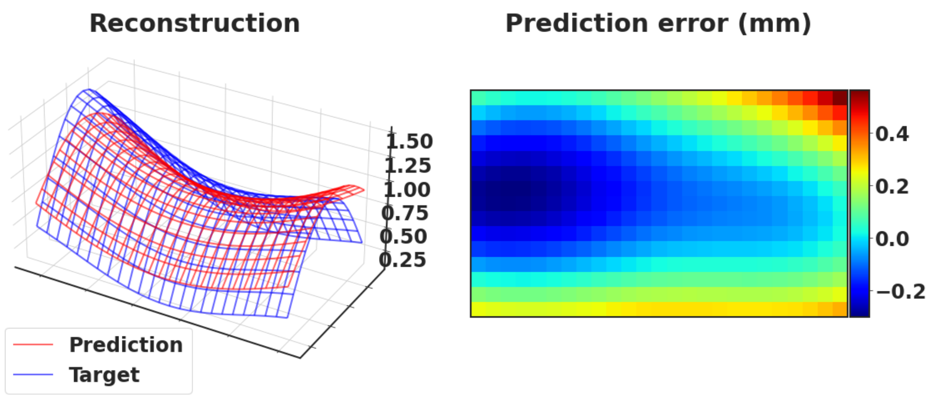


Figure D.2: Reconstruction of surface from the generated test set.

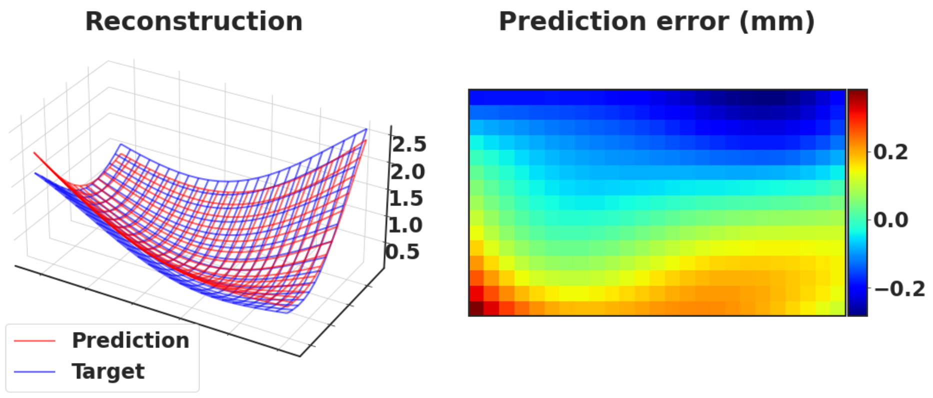


Figure D.3: Reconstruction of surface from the generated test set.

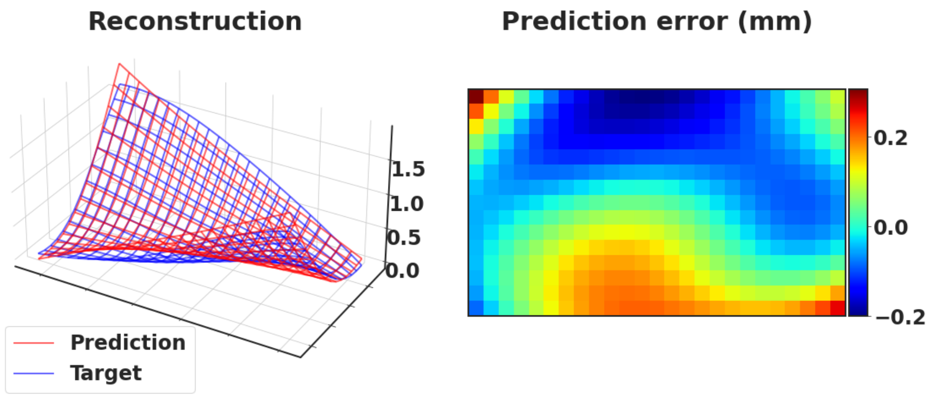


Figure D.4: Reconstruction of surface from the generated test set.

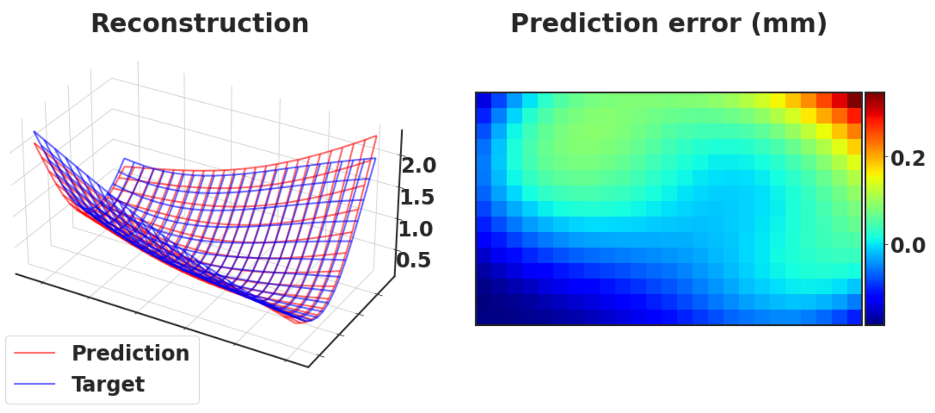


Figure D.5: Reconstruction of surface from the generated test set.

Five surface reconstructions from the real test set using the PCA basis alongside neural network mapping function:

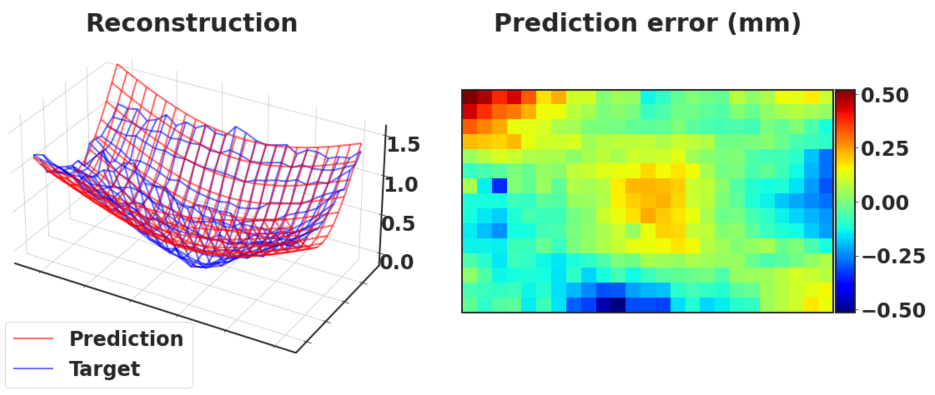


Figure D.6: Reconstruction of surface from the real test set.

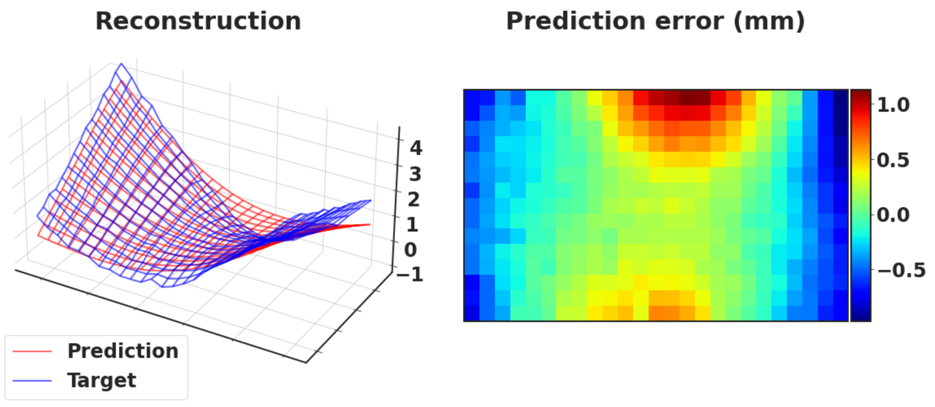


Figure D.7: Reconstruction of surface from the real test set.

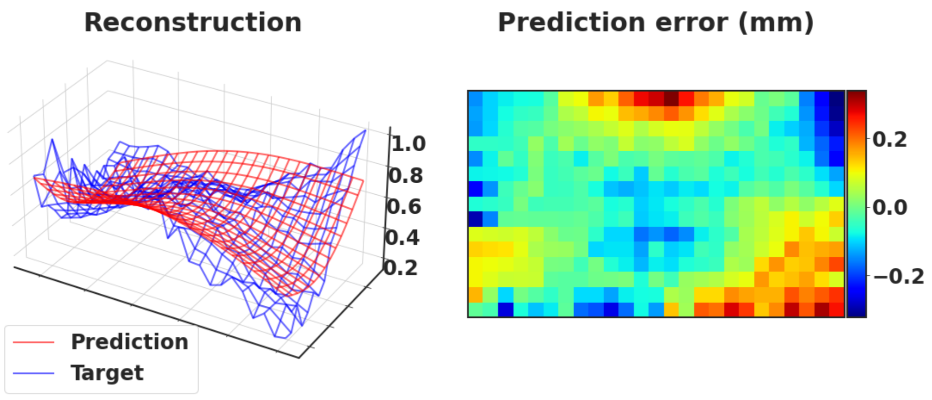


Figure D.8: Reconstruction of surface from the real test set.

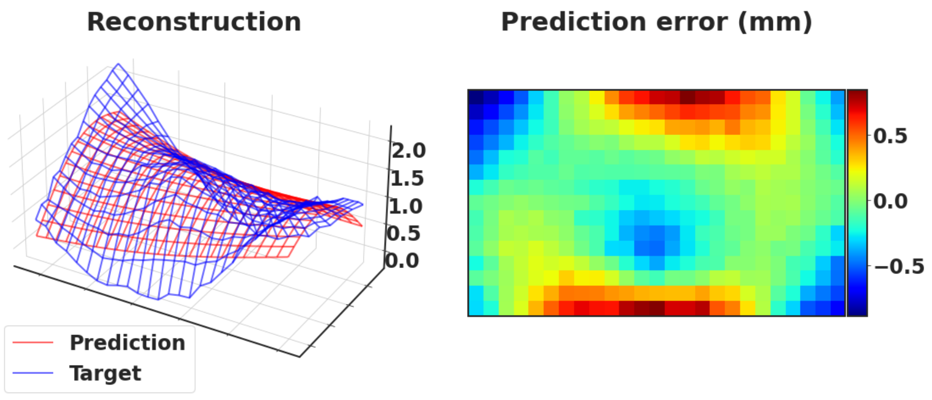


Figure D.9: Reconstruction of surface from the real test set.

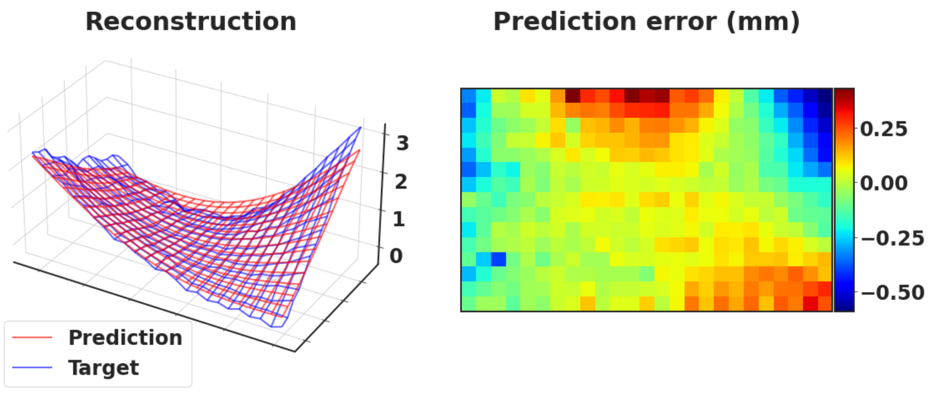


Figure D.10: Reconstruction of surface from the real test set.

Master's Theses in Mathematical Sciences 2021:E7
ISSN 1404-6342

LUTFMS-3407-2021

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/