Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Ola Alstad

Convolutional Neural Networks for Filtering Reflections in Laser Scanner Systems

Master's thesis in Mechanical Engineering
Supervisor: Olav Egeland
July 2021

NTNU
Norwegian University of
Science and Technology

Ola Alstad

Convolutional Neural Networks for Filtering Reflections in Laser Scanner Systems

Master's thesis in Mechanical Engineering
Supervisor: Olav Egeland
July 2021

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Preface

This thesis is submitted as the final work for the requirements of Master of Science in Mechanical Engineering at the Norwegian University of Science and Technology (NTNU). The work in this thesis has been carried out in the spring of 2021, during the COVID-19 pandemic.

Basic knowledge in computer vision and machine learning is beneficial for reading this thesis, however an introduction is provided in the background chapter. The main contribution of the thesis is to compare the performance of a convolutional neural network to filter reflections with different laser scanning systems. The thesis is primarily a contribution to the advancement of robotic welding systems. The convolutional neural network is trained and compared on simulated scans from systems with a single camera, stereo cameras and color encoded scan lines.

Acknowledgements

I would like to thank my supervisor Olav Egeland for an exciting subject, and the freedom while exploring it. Lars Tingelstad has given me general guidelines throughout the thesis, which has been appreciated. Discussions with Sebastian Grans have been of great help, giving me a second opinion on technical problems. The support from my family has been invaluable throughout the thesis.

Summary

Robotic welding and quality control require high accuracy 3D measurements of the workpieces. Structured light methods are widely used to capture these measurements, but for reflective metals such as aluminum, reflections can cause false measurements. This thesis explores the capability of a convolutional neural network (CNN) to distinguish the true measurements from the false, for simulated scan images from different laser scanning systems. The CNN model is trained on various simulated reflections, and the performance of each system is compared on distinct types of reflections. It was found that the CNN model was good at distinguishing blurry reflections from the true scan line across all methods, but had problems where strong reflections overlapped the true scan line. The methods using two cameras were better at predicting the validity of sharp specular reflections, than the methods using one camera. Color encoding the scan line and using a pre-processing step based on epipolar geometry with color matching to filter reflections, further improved the results on specular reflections. Based on the results from this thesis, machine learning shows great promise to be a component for filtering reflections in a real laser scanning system.

Sammendrag

Robotsveising og kvalitetskontroll krever nøyaktige 3D målinger av arbeidsstykkene. Strukturert lys er en utbredt metode for å ta disse målingene, men for reflektive materialer som aluminium, kan refleksjoner forårsake falske målinger. Oppgaven utforsker et konvolusjonelt nevralt nettverks evne til å detektere falske målinger gjennom simulerte refleksjoner, som blir sammenlignet for flere typer lasersystemer. Resultatene viste at det nevrale nettverket var god til å skille uskarpe refleksjoner fra den ekte laser linjen for alle metoder, men hadde problemer der sterke refleksjoner overlappet den ekte laserlinjen. Metodene som brukte to kameraer var bedre til å skille skarpe refleksjoner fra den ekte laser linjen. Å fargekode laserlinjen, med et forbehandlingssteg basert på epipolar geometri, gjorde resultatene bedre på skarpe refleksjoner. Basert på resultatene fra denne oppgaven, vurderes maskinlæring til å være en lovende komponent for å filtrere refleksjoner på et ekte laserskanner system.

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

The following introductory chapter will present the motivation, describe the problems to be addressed, review the most relevant literature and state the major goals of the thesis.

1.1. Motivation

Robotic welding is the use of mechanical robots to automate the welding process completely. The goal of robotic welding is to replicate the work traditionally performed by highly skilled operators while utilizing the general advantages such as productivity, cost-effectiveness and safety of using robots compared to manual labor [21]. Industries are incentivized to increase the use of robotics and automation to create value and ensure competitiveness in the future [25]. Industrial robotic welding is one of the most applied fields of robotics worldwide, extensively used in high production applications such as the automotive industry. Increasing the capabilities of the range of tasks robots can execute, is therefore of great interest.

To be able to replicate the work of a highly skilled welder, one of the inevitable prerequisites is to sense and acquire information of the welding process [4]. The majority of welding vision systems are based on structured light or range data collection. In structured light methods, also referred to as laser scanning, laser diodes are used to project a predefined pattern at a set angle. Triangulation mathematics are then used to determine 3D points on the surface of the scanned object [7]. This information can then be used for application-specific processes such as planning robotic welding trajectories and feedback control during seam tracking. Laser scanning can also be used after the welding process for quality monitoring, identifying defects such as porosity, metal spatter, irregular bead shape, and burn-through [10].

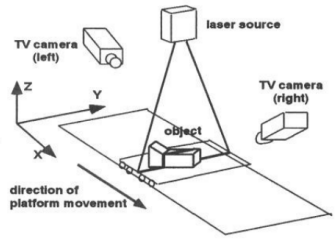


Figure 1.1.: Stereo camera laser scanner [26]

1.2. Problem description

When it comes to highly reflective materials such as aluminum, laser scanners encounter challenges when determining the geometry of the reflective surface. When the projected light from the laser hits the reflective object, the reflection causes several issues which, demands additional methods for determining the true measurement of the laser, where the true measurement is referred to as the measurement the sensor would get if the object were not reflective. The true measurement points can be lost due to weak diffuse reflection, reflections may corrupt the true measurement, and specular reflections can cause outlier measurement errors [28].

1.3. Related work

Removing unwanted reflections with laser scanning could be considered a quite specific and narrow field. However there exist a few contributions to the topic, which from this literature review, was found to date back to 1994 with E. Trucco et al. using a stereo camera setup to scan a moving platform with an object as shown in 1.1[26]. The stereo camera setup was found to be able to scan highly reflective objects, but uses a setup where the cameras are far apart and scanning the object from vastly different angles.

In [28], a study on the outlier formation caused by specular reflections is conducted using an integrated commercial scanner with two sensors and a laser scanner. The two sensors are used for less occlusion of the scan line, and not to validate measurements. The paper shows how outlier planes appear in the resulting point cloud of the scanner, and proposes two models for determining these planes. The models are directly related to the geometry of the scanned object and are not a generic outlier removal filter.

Sebastian Grans showed how a simulated laser scanner in Blender could be used for neural-network training data [9]. The paper shows that virtually generated laser scan images are promising for transferring knowledge to the real domain,

and Blender proves sufficient for generating synthetic data.

1.4. Goal

An approach to removing the measurements of the reflections is to use conventional laser scanners, and apply a post-processing step to identify the true geometry of the part. However, this thesis aims to identify how extra information can be incorporated in the scanning and image processing stage, to then be used as input to a machine learning model. Four different methods are used to produce the input images for the machine learning model. The three methods to be compared are

- traditional laser scanning with one camera and a laser line,
- geometric consistency through stereo cameras and a laser line,
- epipolar consistency with a colored laser line,
- and combined geometric and epipolar consistency with a colored laser line.

Chapter 2.

Preliminaries

2.1. Points, lines and planes

The following section summarizes key elements on geometry in 2D and 3D from [6] useful for computer vision.

2.1.1. Points and lines in 2D geometry

Points in 2D

A point $\mathbf{p}$ on the 2D Euclidean plane can be represented with 2 coordinates with

$$\boldsymbol{p} = \begin{bmatrix} x & y \end{bmatrix}^T \tag{2.1}$$

It is useful in vision algorithms to describe the point in terms of a homogeneous representation $\boldsymbol{x}$ with

$$\boldsymbol{x} = x_3 \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix} \tag{2.2}$$

where all non-zero values of x_3 represent the same Euclidean point $\boldsymbol{p}$.

Lines in 2D

A common approach to representing lines in the 2D Euclidean plane is with the following expression

$$y = Ax + B \tag{2.3}$$

However lines parallel to the y-axis is not defined, and a more general representation is

$$ax + by + c = 0 \tag{2.4}$$

The line is described in terms of homogeneous coordinates with

$$l = \begin{bmatrix} a & b & c \end{bmatrix}^T \tag{2.5}$$

2.1.2. Points, lines and planes in 3D geometry

Points in 3D

A point $\boldsymbol{p}$ in 3D Euclidean space can be represented by 3 coordinates with

$$\boldsymbol{p} = \begin{bmatrix} x & y & z \end{bmatrix}^T \tag{2.6}$$

A homogeneous representation of the point can be made with

$$\boldsymbol{x} = x_4 \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix} \tag{2.7}$$

where all non-zero values of x_4 represents the same Euclidean point $\boldsymbol{p}$.

Lines in 3D

Lines in 3D can be described in terms of the 6 parameter representation of Plücker coordinates. Plücker coordinates have a geometric interpretation consisting of a direction vector $\boldsymbol{a}$ and a moment $\boldsymbol{m}$. Given 2 Euclidean points $\boldsymbol{x}$ and $\boldsymbol{y}$, the Plücker line is calcuclated as follows

$$(\boldsymbol{l}, \boldsymbol{l}') = (\boldsymbol{y} - \boldsymbol{x}, \boldsymbol{x} \times \boldsymbol{y}) = (\boldsymbol{a}, \boldsymbol{m}) \tag{2.8}$$

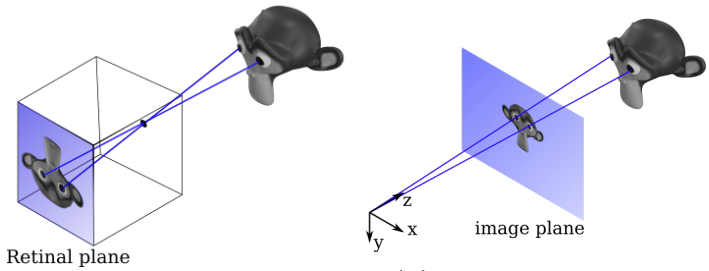
Planes in 3D

Planes in 3D can be described by 4 coordinates with

$$\boldsymbol{\pi} = \begin{bmatrix} a & b & c & d \end{bmatrix}^T \tag{2.9}$$

where $\boldsymbol{n} = \begin{bmatrix} a & b & c \end{bmatrix}^T$ is the normal vector of the plane and $-d/|\boldsymbol{n}|$ is the distance from the origin to the plane in the direction of $\boldsymbol{n}$. This implies that a plane can be constructed with a normal vector $\boldsymbol{n}$ and a point $\boldsymbol{p}$ on the plane as

$$\boldsymbol{\pi} = \begin{bmatrix} \boldsymbol{n} \\ -\boldsymbol{n} \cdot \boldsymbol{p} \end{bmatrix} \tag{2.10}$$



(a) Pinhole camera **(b)** Pinhole camera model

Figure 2.1.: Pinhole camera and model

Point as intersection of line and plane

The intersection of a line and a plane can be derived from the dual of the Plücker coordinates of a plane, and can be found in [6]. The derivation is cumbersome and the result, the homogeneous coordinate as the intersection of a line and a plane, is directly stated here as

$$(\boldsymbol{x}, x_4) = (-u_4\boldsymbol{l} + \boldsymbol{u} \times \boldsymbol{l}', \boldsymbol{u} \cdot \boldsymbol{l}) \tag{2.11}$$

2.2. Computer vision fundamentals

The following subchapter will present excerpts surrounding traditional computer vision, mainly from [6].

2.2.1. Pinhole camera model

The most common camera model in computer vision is the pinhole camera, which projects 3D Euclidean points to the image plane. In the pinhole camera model, light rays into the camera passes through a single point called the optical center. This ideal pinhole camera model has no lenses used to focus light. An illustration of a pinhole camera where light rays hit the retinal plane is shown in 2.1a. The mathematical relationship of the pinhole camera model is simplified by using a virtual image plane in front of the camera as shown in 2.1b[6].

It is common to introduce another plane called the normalized image coordinates, where the z-value of the image plane is normalized to 1. The mapping from normalized image coordinates to pixel coordinates 2.12 is done using the camera parameter matrix 2.13.

$$\boldsymbol{p} = \boldsymbol{K}\boldsymbol{s} \tag{2.12}$$

$$\boldsymbol{K} = \begin{bmatrix} \frac{f}{\rho_w} & 0 & u_0 \\ 0 & \frac{f}{\rho_h} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

The inverse relationship is then given by the inverse camera matrix $\boldsymbol{K}^{-1}$, which maps pixel coordinates to normalized image coordinates as given in 2.14.

$$\boldsymbol{s} = \boldsymbol{K}^{-1}\boldsymbol{p} \tag{2.14}$$

The camera matrix can also be used to describe other optical devices such as projectors, with the only difference that the projector projects light instead of measure incoming light.

2.2.2. Epipolar geometry

Two optical devices described by the camera matrix, viewing a scene from two distinct positions, have a geometrical relationship between the image points of each devices described by epipolar geometry. Consider the two optical devices in 2.2. The vector $\boldsymbol{r}_1$ goes through the normalized image coordinate $\boldsymbol{s}_1$ hitting the point P in the scene. From the view of the camera, the direction of $\boldsymbol{r}_1$ is known as it intersects $\boldsymbol{s}_1$, but the length of the vector is unknown. Consider a range of possible points $\bar{\boldsymbol{P}}$ at the end point of $\boldsymbol{r}_1$. The vector $\boldsymbol{r}_2$ to these possible points, intersects the image plane of camera 2, creating a line of possible points. This line of possible points is called the epipolar line and is denoted $\boldsymbol{l}_2$ in frame 2. The possible points form a line because $\boldsymbol{r}_1$, $\boldsymbol{r}_2$ and $\boldsymbol{t}_{21}$ lie in the same plane, which means that the triple scalar product of the three vectors is 0.

$$\boldsymbol{r}_2 \cdot (\boldsymbol{t}_{21} \times \boldsymbol{r}_1) = 0 \tag{2.15}$$

In frame 2, the coordinate form is

$$(\boldsymbol{r}_2^2)^T (\boldsymbol{t}_{21}^2)^\times \boldsymbol{R}_1^2 \boldsymbol{r}_1^1 = 0 \tag{2.16}$$

The constraint between $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ is usually defined in terms of the epipolar matrix

$$\boldsymbol{E} = (\boldsymbol{t}_{21}^2)^\times \boldsymbol{R}_1^2 \tag{2.17}$$

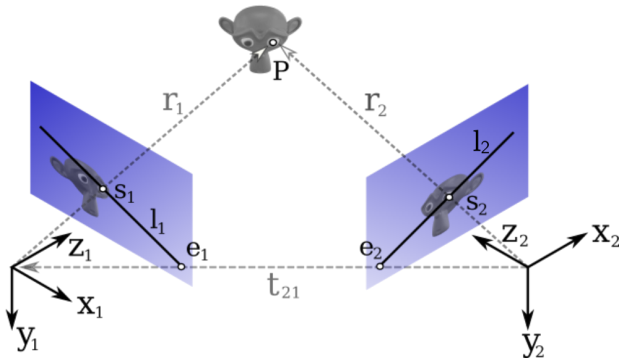


Figure 2.2.: Stereo camera/projector setup

Substituting in the expressions for the normalized image coordinates, $\boldsymbol{r}_1^1 = \lambda_1 \boldsymbol{s}_1$ and $\boldsymbol{r}_1^2 = \lambda_2 \boldsymbol{s}_2$, gives

$$\lambda_2 \boldsymbol{s}_2^T \boldsymbol{E} \lambda_1 \boldsymbol{s}_1 = 0 \tag{2.18}$$

The essential matrix is independent of scaling such that the constraint can be simplified to

$$\boldsymbol{s}_2^T \boldsymbol{E} \boldsymbol{s}_1 = 0 \tag{2.19}$$

It can be shown that the essential matrix can be used to calculate the epipolar lines in frame 2, directly from the normalized image coordinate of frame 1 as [6]

$$\boldsymbol{\ell_1} = \boldsymbol{E} \boldsymbol{s}_1 \tag{2.20}$$

and in the other direction as

$$\boldsymbol{\ell_2} = \boldsymbol{E}^T \boldsymbol{s}_2 \tag{2.21}$$

It is possible to make the same epipolar constraint in pixel coordinates by substituting $\boldsymbol{s}_1 = \boldsymbol{K}_1^{-1} \boldsymbol{p}_1$ and $\boldsymbol{s}_2 = \boldsymbol{K}_2^{-1} \boldsymbol{p}_2$. The constraint between the pixel coordinates is then

$$\boldsymbol{p}_2^T \boldsymbol{F} \boldsymbol{p}_1 = 0 \tag{2.22}$$

where $\boldsymbol{F}$ is now named the fundamental matrix, and is given from the essential matrix as

$$\boldsymbol{F} = \boldsymbol{K}_2^{-T} \boldsymbol{E} \boldsymbol{K}_2^{-1} \tag{2.23}$$

As with epipolar lines in normalized image coordinates, epipolar lines in pixel coordinates can be determined similarly as

$$\boldsymbol{\ell}_1 = \boldsymbol{F}^T \boldsymbol{p}_2, \quad \boldsymbol{\ell}_2 = \boldsymbol{F} \boldsymbol{p}_1 \tag{2.24}$$

2.2.3. Homographies

A homography in three dimensional space is an invertible transformation from a point $\boldsymbol{x}$ to $\boldsymbol{x}'$ given by[6]

$$\boldsymbol{x}' = \boldsymbol{H}\boldsymbol{x} \tag{2.25}$$

and its inverse transformation as

$$\boldsymbol{x} = \boldsymbol{H}^{-1}\boldsymbol{x}' \tag{2.26}$$

A general homography consists of 9 elements and is only equivalent under scaling, resulting in 8 independent elements. There will always exist a scaling factor μ such that the homography can be written in its normalized form

$$\mu\boldsymbol{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \tag{2.27}$$

with the bottom right element as 1.

Planar homography

The following section is based on [6] and [15]. Consider two cameras viewing the same scene from two distinct positions. Now suppose that the same point $\boldsymbol{X}$ are given in frame of camera 1 and camera 2 as $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$. Given the geometrical relationship between the camera frames the following relationship can be made

$$\boldsymbol{X}_2 = \boldsymbol{R}\boldsymbol{X}_1 + \boldsymbol{t} \tag{2.28}$$

Lets consider that all points of interest lie on a plane in the scene with normal vector $\boldsymbol{N}$. The distance from the plane to the optical centre of camera 1 is denoted as d. For the arbitrary point $\boldsymbol{X}_1$, the distance is calculated using the dot product between a point and a vector as

$$d = \boldsymbol{N} \cdot \boldsymbol{X}_1 = \boldsymbol{N}^T\boldsymbol{X}_1 \longleftrightarrow \frac{1}{d}\boldsymbol{N}^T\boldsymbol{X}_1 = 1 \tag{2.29}$$

Substituting this in to 2.28 gives

$$\boldsymbol{X}_2 = \boldsymbol{R}\boldsymbol{X}_1 + \boldsymbol{t}\frac{1}{d}\boldsymbol{N}^T\boldsymbol{X}_1 = (\boldsymbol{R} + \frac{1}{d}\boldsymbol{t}\boldsymbol{N}^T)\boldsymbol{X}_1 \tag{2.30}$$

which gives the homography

$$\boldsymbol{H} = \boldsymbol{R} + \frac{1}{d}\boldsymbol{t}\boldsymbol{N}^T \tag{2.31}$$

Denoting the normalized image coordinates of $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ as

$$\boldsymbol{x}_1 = \lambda_1 \boldsymbol{X}_1, \quad \boldsymbol{x}_2 = \lambda_2 \boldsymbol{X}_2 \tag{2.32}$$

we get that

$$\boldsymbol{X}_2 = \boldsymbol{H}\boldsymbol{X}_1 \longleftrightarrow \boldsymbol{x}_2 = \boldsymbol{H}'\boldsymbol{x}_1 \tag{2.33}$$

where $\boldsymbol{H}$ and $\boldsymbol{H}'$ are equivalent homography matrices since homographies are equivalent under scaling. The scaling is expressed as

$$\boldsymbol{H}' = \frac{\lambda_2}{\lambda_1}\boldsymbol{H} \tag{2.34}$$

Given the camera matrices for camera 1 and 2, the mapping between the normalized image coordinates and pixel coordinates can be made as

$$\boldsymbol{p}_1 = \boldsymbol{K}_1 \boldsymbol{x}_1, \quad \boldsymbol{p}_2 = \boldsymbol{K}_2 \boldsymbol{x}_2 \tag{2.35}$$

Substituting in for 2.33 we get

$$\boldsymbol{p}_2 = \boldsymbol{K}_2 \boldsymbol{H} \boldsymbol{K}_1^{-1} \boldsymbol{p}_1 = \bar{\boldsymbol{H}} \boldsymbol{p}_1 \tag{2.36}$$

resulting in that the homographic mapping between pixel coordinates for two cameras with known geometrical relationship, viewing points on a plane, is

$$\bar{\boldsymbol{H}} = \boldsymbol{K}_2 \boldsymbol{H} \boldsymbol{K}_1^{-1} = \boldsymbol{K}_2 (\boldsymbol{R} + \frac{1}{d}\boldsymbol{t}\boldsymbol{N}^T) \boldsymbol{K}_1^{-1} \tag{2.37}$$

2.3. Laser triangulation

In computer vision, triangulation is the process of determining the spatial dimension of a point or object, such that the given points and solution forms a triangle.

2.3.1. 2D to 3D mapping

Given a laser scanning setup with a camera and a laser as shown in 2.3, where the goal is the get an accurate 3D point cloud of the object which is scanned. The geometrical relationship between the camera and laser is constant, while the

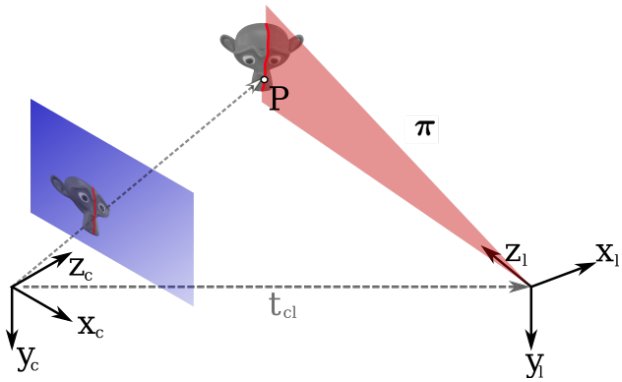


Figure 2.3.: Laser scanning geometry

object to be scanned has a relative motion to the camera-laser system, such that the laser line is swept along the object.

To calculate the 3D points on the surface of the scanned object, denote the measured pixel $\boldsymbol{p}$ in the 2D image plane and the known laser plane $\tilde{\boldsymbol{u}}$. The normalized image coordinate $\boldsymbol{s}$ of pixel coordinate $\boldsymbol{p}$, is found by using 2.14. The line through the optical centre of the camera and the normalized image coordinate, in the frame of the camera, is

$$\boldsymbol{\ell} = (\boldsymbol{l}, \boldsymbol{l}') = (\boldsymbol{s}, \boldsymbol{0}) \tag{2.38}$$

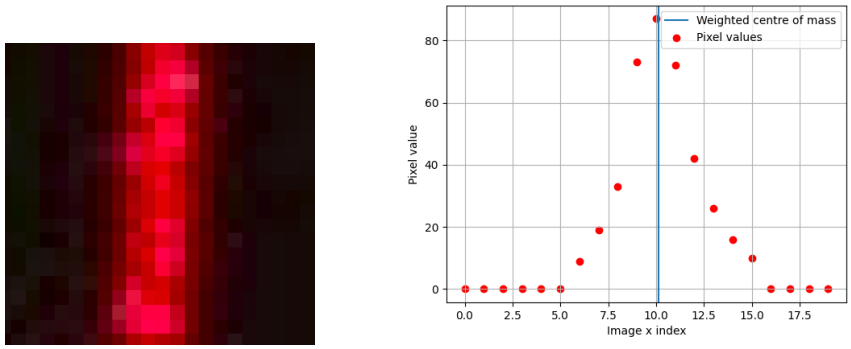
The line has direction vector is $\boldsymbol{s}$, and the moment is 0 since the distance from the line to the optical centre is 0. Calculating the intersection of a line and a plane we get [6]

$$\boldsymbol{x} = -\frac{u_4 \boldsymbol{s}}{\boldsymbol{u} \cdot \boldsymbol{s}} \tag{2.39}$$

2.3.2. Subpixel accuracy

Consider the close view of the laser line as shown in 2.4a. To extract an accurate 2D coordinate for the measurement of the laser plane, a method for determining the subpixel accuracy is needed. One method for determining subpixel accuracy is the *weighted centre of mass* [17], which is calculated for each row in the image as follows

$$x_{ic} = \frac{\sum_{j=s}^{e} j \bar{I}(j)^2}{\sum_{j=s}^{e} \bar{I}(j)^2} \tag{2.40}$$



(a) Close view of laser line **(b)** Weighted centre of mass

Figure 2.4.: Subpixel accuracy of laser line

where j is the row-index, $\bar{I}$ is the row normalized pixel intensity, s is the start of each row and e is the end of each row. The laser intensity profile $I(x)$ of each row in the image is unity-based normalized with

$$\bar{I} = \frac{I(x) - \min I(x)}{\max I(x) - \min I(x)} \tag{2.41}$$

The weighted centre of mass for one of the rows in 2.4a is shown in 2.4b.

2.4. Convolutional neural networks

The following section will cover the necessary aspects of fully convolutional neural networks in the context of 2D visual imagery for the work following this chapter. It is important to note that convolutional neural networks (CNN) have applications in fields other than 2D visual imagery, but the following section will only consider it in the context with 2D images as input. Convolutional neural networks are a subset of a machine learning used for optimizing successive filters given a dataset. The following sections summarizes excerpts from the *Deep Learning* book [8], where most material are from the *Convolutional Networks* chapter.

2.4.1. Convolution operation

The input for a convolutional neural network is often a multidimensional array, referred to as a tensor. The input tensor to a CNN can be an image, given by a height, width and number of channels. For a two dimensional input and kernel

with indices i and j, the convolution operation is denoted

$$F(i,j) = (K * I)(i,j) \tag{2.42}$$

where the output F is refereed to as a feature map, I is the input and the argument K is the kernel[1]. A kernel can also be referred to as a filter, due to its practical applications of filtering the image for features such as lines. An example of a convolution operation is shown in 2.5a. Each output can $f_{i,j}$ can be calculated with [8].

$$F(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.43}$$

As an example, the calculation of $f_{1,1}$ is

$$f_{1,1} = t_{1,1}k_{1,1} + t_{1,2}k_{1,2} + t_{2,1}k_{2,1} + t_{2,2}k_{2,2} \tag{2.44}$$

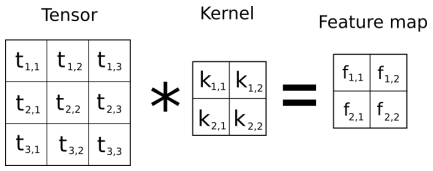
A numeric example with a convolution with a four-by-four input and a three-by-three kernel is shown in 2.5b, with the same method of calculating the output feature map. Convolutions can also be calculated on three-dimensional inputs as shown in 2.5c. The depth dimension is denoted d, and must match on both the input tensor and kernel. Using multiple filters on the same input gives a three-dimensional feature map as shown in 2.5d. Each three-dimensional kernel is independently convolved with the input, and the output of each filter is stacked in the output. In the example, four kernels are used to create four feature maps in the output. The equation for the three-dimensional convolution with multiple filters is

$$f_{i,j,d} = \sum_{l,m,n} I_{l,j+m-1,d+n-1} K_{d,l,m,n} \tag{2.45}$$

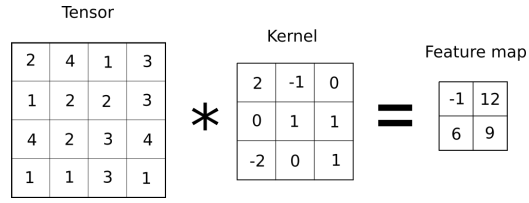
2.4.2. Pooling layer and downsampling

Pooling is an operation in which the output is a summary statistic of the nearby inputs. The most common pooling layer is max pooling, which outputs the maximum value within a neighborhood of the input. Pooling layers are a computationally effective way to downsample feature maps, by summarizing the presence of features in patches of the previous feature map. Max pooling is a good choice for downsampling feature maps because it keeps the highest activations, which is interpreted to be the most important aspects of each channel. A tensor undergoing a two-by-two maxpool kernel with stride 2 is shown in 2.6. The stride is the horizontal or vertical steps the kernel is moved before a new value is calculated over

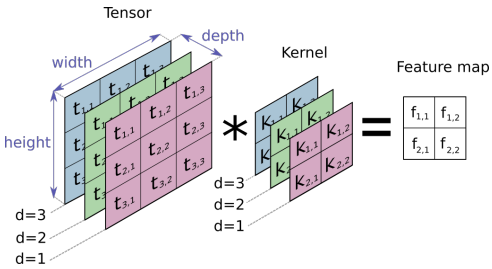
[1] The correct mathematical term for the given equation is cross-correlation, but the term convolution is more widely used in the context of CNN. In a mathematical convolution the indexes i and j are flipped in the input and output arguments.



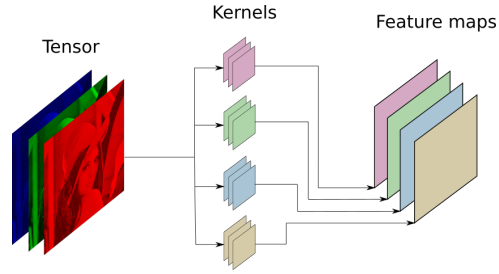
(a) Symbolic 2 dimensional convolution



(b) 2 dimensional convolution example



(c) 3 dimensional convolution example



(d) Multiple filters

Figure 2.5.: Convolution examples

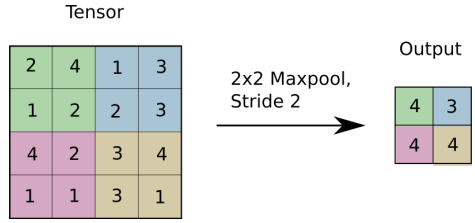
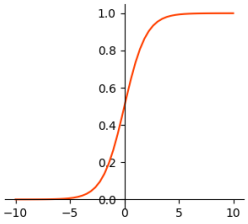


Figure 2.6.: Maxpool example

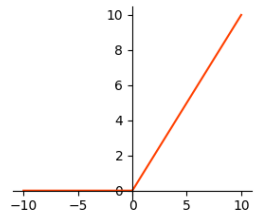
the input tensor. An alternative to pooling for downsampling, is to use a filter with stride bigger than one, such that the weights of the filter are tuned optimally. The interpretation is then that the filter learns the optimal way to downsample an image. Note that the pooling operation has no learnable parameters, while using a filter, the weights will be tuned during optimization. A pooling layer is therefore more computationally efficient, without a significant drop in performance.

2.4.3. Padding

The convolutions which have been mentioned so far have been *valid* convolutions, in contrast to a *same* convolution. For a same convolution, the input is padded such that the spatial dimensions of the output is the same as the input. To pad the input is to artificially increase the spatial resolution, by appending numeric values to the boundaries of the tensor. The most common approach is to pad zeros to the



(a) Sigmoid



(b) ReLU

Figure 2.7.: Examples of non-linear activations

boundaries, although there exist other rules for padding. If the spatial dimensions of a tensor in layer l is $n_l \times n_l$ the spatial dimensions of the subsequent layer is

$$n_{l+1} = \frac{n_l + 2p - k}{s} + 1 \tag{2.46}$$

where p is the padding, k is the kernel size and s is the stride. For a same convolution we have $n_{l+1} = n_l$ and $s = 1$ such that the required padding is solved for as

$$p = \frac{k - 1}{2} \tag{2.47}$$

2.4.4. Non-linear activation

To build a network of convolutions, several convolution operations are applied step-wise to the input. However, the convolution operation is just a linear operator. Step wise applying only linear operations to the input, would make the output be linearly dependent on the input. The whole network could then be reduced to a single convolution. To make the network to be able to learn non-linear relationships between the input and output, a non-linear function f is applied to the outputs of the convolutions in the network as such

$$\bar{F}(i, j) = f(F(i, j)) = f(K * I)(i, j)) \tag{2.48}$$

The function f is a non-linear function that is usually computationally efficient to calculate the derivative of, and is referred to as an activation function. Two of the most common activation functions are the sigmoid as shown in 2.7a, and the rectified linear unit (ReLU) as shown in 2.7b.

2.4.5. CNN architecture

Typical CNNs have a similarity in the sequence of layers used. The typical pattern of layers is shown in 2.8, which consists of a convolution followed by a non-linear activation function and optionally, a pooling layer. Traditional networks which was an essential part of the emergence of CNNs, such as VGG[24] and AlexNet[13], used this pattern in a single path network. When referring to a convolution layer, it can have two meanings, either it means only the convolution operation layer, or it can refer to the whole convolution block in 2.8 depending on the context. When making simple adjustments to existing architectures, or making a new one, it is generally a good choice to follow the discussed pattern.

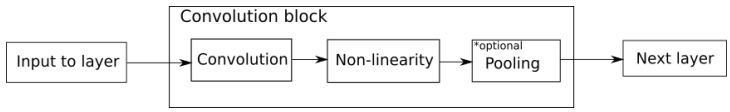


Figure 2.8.: Typical arrangement of layers in CNN

2.4.6. Receptive field

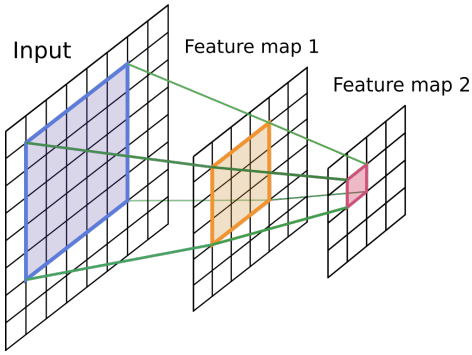
Since convolutions are locally connected in a network, each part of the output may only be a function of a certain region of the input. The spatial region of the input that a certain spatial position of the output is dependent on, is called the receptive field. Consider a three-layer network with kernel size 3×3 as in 2.9. Each of the pixels in the last feature map is a function of a larger region of the input. The receptive field r_{l-1} in the previous layer in a network is given as

$$r_{l-1} = s_l r_l + (k_l - s_l) \tag{2.49}$$

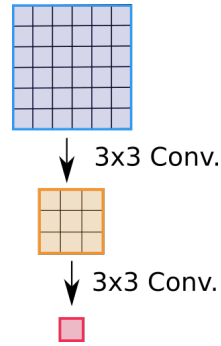
where k_l is the kernel size and s_l is the stride of layer l. Solving the recursive equation for a whole single path network then works out to be[2]

$$r_0 = \sum_{l=1}^{L}((k_l - 1)\Pi_{i=1}^{l-1} s_i) + 1 \tag{2.50}$$

For a single-path network with equal stride and kernel size for all layers, three parameters can be changed to increase the receptive field, the kernel size, the stride, and the number of layers. Changing the stride is the most effective approach to increase the receptive field, since it is a multiplicative term in the equation, compared to the additive term of the kernel size. Increasing the number of layers L will also increase the receptive field.



(a) Receptive field for three layers with 3 by 3 kernel



(b) Convolutions increase receptive field

Figure 2.9.: Receptive field

2.4.7. Optimization and loss function

So far it has been discussed how to set up a model, but not what problem it is solving. When the network is initially set up, the filters of the network only contain random weights and are not capable of solving any meaningful task. What the network will eventually achieve is dependent on the dataset it is given. In the context of 2D visual imagery the dataset is a large number of images, which usually range from 500 to over 100 000 in numbers. Each of the images in the dataset must contain some associated ground truth, how this ground truth is defined, vary from task to task. A machine learning model is usually implemented in a framework, the most popular being Tensorflow[16] and Pytorch[19]. The main feature of these machine learning frameworks is the automatic calculation of gradients for a model. The cornerstone of being able to efficiently calculate the gradients for a model which could contain millions of tuneable parameters, is the simple chain rule for derivatives. Consider 2.10 which shows a series of functions f, g and h, applied to the input a to produce the output d. We have $b = f(a)$, $c = g(b)$ and $d = h(c)$. In a machine learning context, we are introduced in calculating the gradient of each of the parameters with respect to the output. Now consider that we want to find the partial derivative of d with respect to each other variable. The partial derivative of d with respect to a can be written as

$$\frac{\partial d}{\partial a} = \frac{\partial d}{\partial c}\frac{\partial c}{\partial b}\frac{\partial b}{\partial a} = f'(c)g'(b)h'(a) \tag{2.51}$$

which can be written as string of derivatives. Now if we want to calculate $\frac{\partial d}{\partial b}$, no new derivatives have to be calculated, as the previous calculated derivatives can

be used with

$$\frac{\partial d}{\partial b} = f'(c)g'(b) \tag{2.52}$$

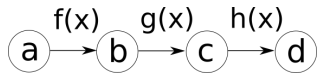


Figure 2.10.: A small computational graph

Loss function

Let the parameters, usually the weights and biases, of a network be given by $\boldsymbol{\theta}$. The optimization problem in deep learning, is to tune the parameters $\boldsymbol{\theta}$ of a network, to reduce a loss function $L(\boldsymbol{\theta})$. The loss function summarizes the error between the predictions and the ground truth of a neural network. A simple loss function is to average the least square error between the predictions and the ground truths. For problems where the network is assigned to predict distinct classes, a loss function based on probabilities is commonly used, the cross entropy loss. Before calculating the cross entropy loss, the outputs of the neural network are converted to probabilities through the softmax function. Let the output of the neural network predicting a class k, be given by z_k. The output of the softmax function $\hat{y}$ is interpreted as the probability of belonging to the class k. For K classes the softmax function for a prediction of class k is

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{k'}^{K} e^{z_{k'}}} \tag{2.53}$$

Given the output of the softmax function $\hat{y}_k$, and denoting the ground truth as y_k, the cross entropy loss function C for a single example n is

$$C_n = -\sum_{k=1}^{K} y_k \log(\hat{y}_k) \tag{2.54}$$

The cross entropy loss may be weighted for specific classes, which is useful if the training set is unbalanced. An unbalanced dataset have large deviation of number of examples for each class. The weighted cross entropy introduces a specific weight for each class w_k, which is multiplied with the loss for the class as

$$C_n = -\sum_{k=1}^{K} w_k y_k \log(\hat{y}_k) \tag{2.55}$$

The total cross entropy loss C for multiple examples n is simply averaging the individual losses C_n as

$$C = \frac{1}{N} \sum_{n=1}^{N} C_n \tag{2.56}$$

Gradient descent and its variants

Given a loss function that summarizes a defined error, it is desirable to minimize this function such that the errors are minimized. The optimization method that has proven to be most efficient for neural networks, is gradient descent and related methods. To update the weights of a neural network using gradient descent, the partial derivative $\frac{\partial C}{\partial w}$ of the loss function C with respect to each weight w must be found. These calculations are effective because of the chain rule and partial derivative calculations as previously discussed. We want to adjust the weights such that the loss function is minimized. The loss function is minimized, by adjusting each weight w, with the following rule for gradient descent

$$w := w - \alpha \frac{\partial C}{\partial w} \tag{2.57}$$

where α is the learning rate. A popular variation of gradient descent is Adam[12], short for adaptive moments. Adam calculates an adaptive learning rate, based on derivatives in the current and previous steps. Adam has proven to reduce the training time and provides robustness to the choice of hyperparameters. More details about Adam compared with other adaptive optimization algorithms can be found in [8].

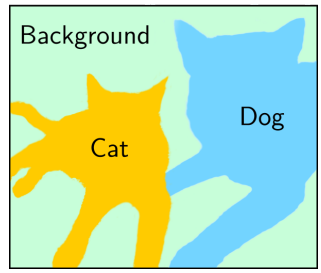
2.4.8. Semantic Segmentation

Semantic segmentation is a process where given an input image, each pixel is predicted to belong to a class[14]. An example is given in 2.11a showing a cat and a dog. For a semantic segmentation problem classifying cats and dogs, the desired output would look like in 2.11b. Note that the number of classes for the example problem is three, since the background is counted as a class in addition to the cats and dogs.

Dice score

For the segmentation problem, a common metric for determining the overlap between the ground truth segmentation and the predicted segmentation is the Sørensen-Dice coefficient[29], also referred to as Dice score. The Sørensen-Dice

(a) Semantic segmentation input



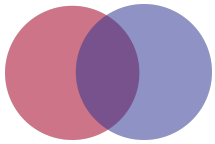
(b) Semantic segmentation output

Figure 2.11.: Semantic segmentation

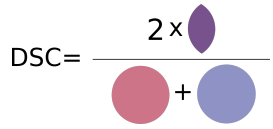
coefficient determines the spatial overlap for two sets A and B as

$$DSC = \frac{2 \times |A \cap B|}{|A| + |B|} \tag{2.58}$$

Two circles with spatial overlap is shown in 2.12a, a visual representation of the calculation is shown in 2.12b.



(a) Intersecting circles



(b) Visual dice score

Figure 2.12.: Dice score

2.4.9. U-Net

U-net is a fully convolutional network initially developed for biomedical image segmentation[22], but has later been widely used in many types of segmentation tasks. The idea behind U-Net is to have a wide receptive field for each spatial location in the output, while maintaining high-resolution information from the input.

Motivation

Consider only stacking convolutional blocks, without pooling, after each other as in 2.9a. The width and height of the receptive field of the output compared to

the input, increases by two for each convolution block. If we have an input image that has a width and height of 1024, from 2.50, there must be a large number of convolution blocks to produce a considerable receptive field. The receptive field can be enlarged by pooling layers or convolutions with stride $s > 1$, but then the resolution of the output decreases according to 2.46, such that finer grained spatial information is lost. U-Net solves these issues by having two paths, one that decreases the resolution for large receptive field, and one that increases the resolution and concatenating finer grained spatial information.

U-Net architecture

The original U-Net architecture is shown in 2.13. The network architecture consists of a contracting path on the left side, and an expansive path on the right side. To pass on finer grained spatial information, skip-connections are made from the contracting path to the expansive path. The contracting path of the network consists of repeating blocks of

- 3×3 convolution,
- ReLU,
- 3×3 convolution,
- ReLU,
- 2×2 maxpool,

which is a typical architecture for CNNs. The expansive path is similar, but instead of downsampling with maxpool, a convolutional up-sampling which also halves the number of feature channels. At the start of each block in the expansive path, the correspondingly cropped feature maps are concatenated onto the up-sampled feature map of the expansive path. Due to the good performance of U-Net in segmentation tasks, several variants have been made. There exists several variants which differ in the depth of the contracting path, and more advanced inner workings. The similarities is the idea behind a contracting and expansive path with information flow in between the paths.

2.5. Physically Based Rendering

The following subchapter is mainly based on [20]. Rendering is the process of generating an image given the description of a 3D scene, used extensively in computer games, movies and simulations. Different rendering techniques exist due to different demands in computational complexity versus realism. The rendering

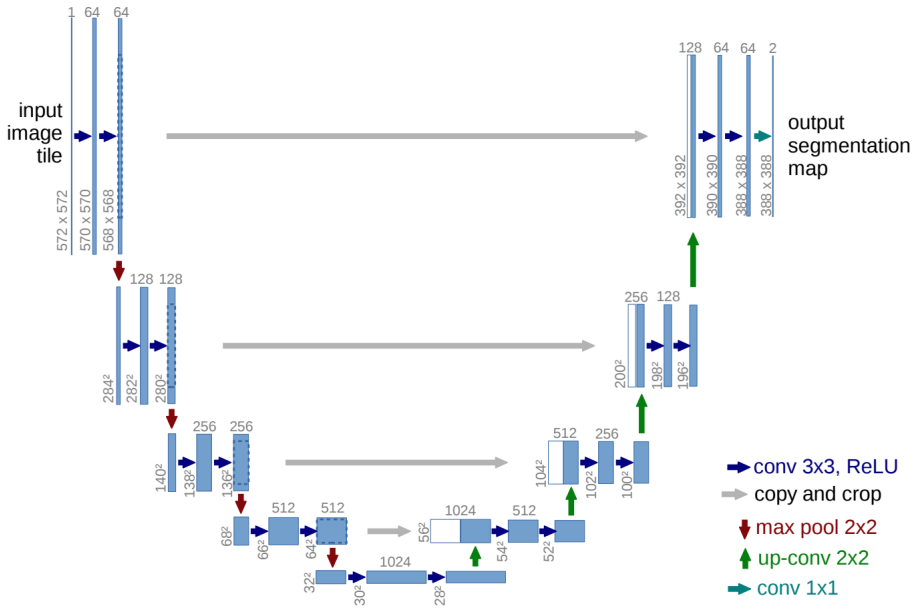


Figure 2.13.: Original U-Net architecture[22]

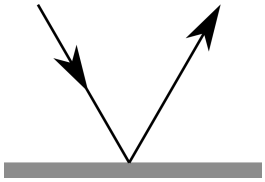
technique that focuses most on realism is called physically based rendering, which is an attempt to simulate reality.

2.5.1. Basic models for reflections

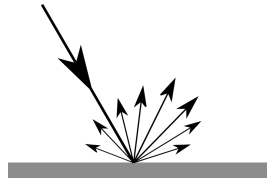
Rendering an image is basically choosing the color and intensity of each pixel in the image. The intensity and color is dependent of the objects, materials and light sources in the scene. The most important factor to create a realistic image from a scene, is accurate calculation of light and how it interacts with the materials and surfaces in the scene. The following section will go through the 3 basic reflection models.

Specular

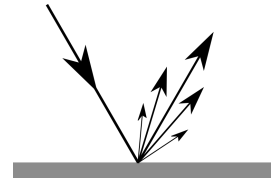
Specular reflection is used to model surfaces such as smooth metal mirrors. For specular reflections there are two basic principles, the law of reflection and the fresnel equation. The law of reflection states that the angle of incident is the same as the angle of reflection, where the incident direction, surface normal and direction of reflection is co-planar. The fresnel equation describes the fraction of light which is reflected and by complement, the fraction which is absorbed[23].



(a) Specular reflection on surface

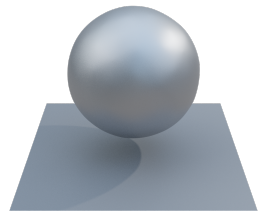


(b) Diffuse reflection on surface

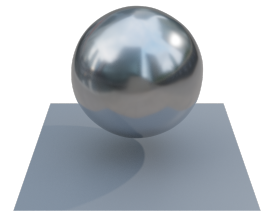


(c) Spread reflection on surface

(d) Sphere with specular surface



(e) Sphere with diffuse surface



(f) Sphere with spread surface

Figure 2.14.: Basic reflections

Diffuse

A surface that reflects light equally in all directions regardless of the incident angle is called an idealized diffuse surface or Lambertian surface. A perfectly diffuse surface does mathematically conserve energy, but does not exist in nature. Diffuse surfaces that reflect light unequally, but in all directions exist, and make up most surfaces we encounter daily.

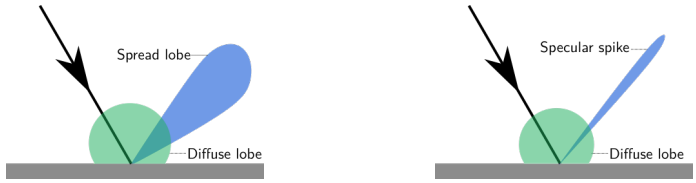
Spread

Although metals have perfect specular reflection for a single light ray, the irregularities in the surface cause the reflections at a larger scale to appear blurry. A spread reflection, also referred to as glossy or imperfect specular reflection, models this behavior.

Combined reflection models

Most real surfaces have reflections that are a mixture of the specular, diffuse and spread reflection models. The combination of reflections that are of interest for this thesis is a weak diffuse reflection combined with a strong spread or specular reflection. The combined spread-diffuse and specular-diffuse models are shown respectively in 2.15a and 2.15b. The spread-diffuse model consists of a diffuse

and spread lobe, while the specular-diffuse model consists of a diffuse lobe with a specular spike. When referring to specular and spread reflections of a material in the following chapters, it is implied that there is also a diffuse lobe in the reflection.



(a) Spread-diffuse reflection

(b) Specular-diffuse reflection

Figure 2.15.: Combined reflection models

2.5.2. Ray tracing

A majority of photorealistic renderers are based on tracing the path of light. This technique is called ray tracing. The ray tracing algorithm follows a path of a ray of light through the scene as it interacts with the objects in it. As the goal of the ray tracing algorithm is to make a realistic 2D image, only the light that makes the 2D image seem realistic is necessary to simulate. Certain simplifications can therefore be made, the computation of light that is certain to not hit the camera can be discarded.

Forward ray tracing

Forward ray tracing calculates how light from a source, bounces around in the scene, and possibly hit the camera. Although the method simulates how light behaves in nature, it is very computationally inefficient. As we are interested in the light that hits the camera, most light rays calculated with forward ray tracing do not.

Backward ray tracing

Backward path tracing reverses the process of forward ray tracing. The light paths are calculated from the camera, as it interacts with the scene objects in the form of reflections, and eventually hit a light source. Compared to forward ray tracing, this method is more computationally feasible as it only calculates light paths that contribute to the image on the camera. All optical systems are reversible, and the backward ray tracing method can theoretically produce the same result as forward ray tracing.

Hybrid ray tracing

As backward ray tracing is more computationally feasible than forward ray tracing, it seems we can discard the forward ray tracing method. In fact, several render engines only use backward ray tracing. However, backward ray tracing has a downfall when it comes to caustics. Caustics are the light that goes through a specular reflection, then hit a diffuse, before it hits the camera. Without diving into how light is sampled in the scene using Monte Carlo sampling, consider how one would trace back the light from a diffuse surface. The light reflected from a diffuse surface could have any incident angle. Tracing this back to a light source is easy since we know the position of the light. If the light comes from a concentrated specular reflectio, it is much harder, since we do not know where these specular reflections occur in the first place. If one trace from the light source these specular reflections can be easily accounted for since they are known in advance, before they hit the diffuse surface. Hybrid ray tracing solves the issue of caustics with forward ray tracing, and the computational feasibility of backward ray tracing by combining the two methods.

Chapter 3.

Method

In this chapter, the method in which the results were acquired is presented. The experiments were conducted using simulation, contrary to real life experiments. As the experiments required simulation of light in interaction with materials and objects, a physically based render engine was chosen. Several physically based rendering softwares exist, which among others include Cycles, LuxCoreRender, PBRT and Mitsuba. While these engines all offer different capabilities when it comes to physically based rendering, some did not include a graphical user interface (GUI) for development. A graphical user interface makes the development easier, because one can visually confirm that certain processes have been successful, without setting up the whole pipeline for a rendered image. These processes include loading of 3D models, initializing the optical devices and their geometrical relationship. Cycles and LuxCoreRender[3] can both be used with the 3D creation suite called Blender, which offers both a GUI and a python API. Blender was therefore chosen as the development platform for this thesis.

3.1. Blender

Blender is an open-source multi purpose 3D computer graphics program, with the main use cases being for creating animated films, visual art and modelling[5]. As these workflows depend on computer vision and rendering, it is suitable for testing computer vision algorithms.

3.1.1. Scene

In computer graphics, a scene can be a complex collection of objects. The scene includes all objects and parameters, that can potentially affect the final rendered image. In Blender, these objects in the scene include cameras, meshes, lights

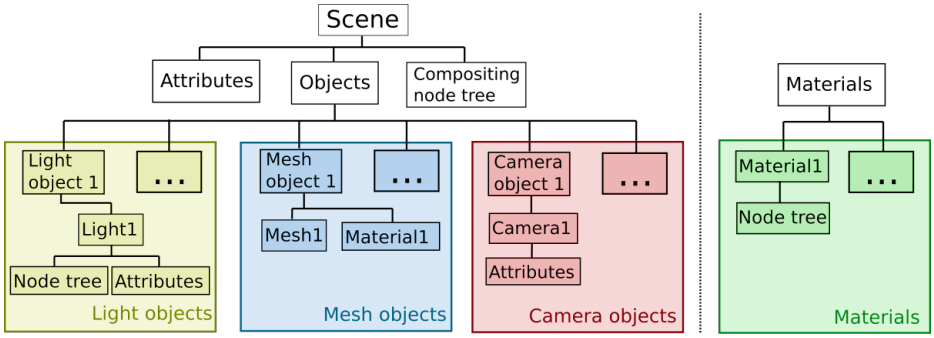


Figure 3.1.: Blender scene structure

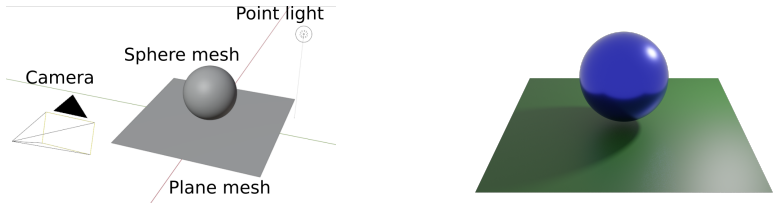
and more, as shown in 3.1. An object in Blender is a meta-class for storing common properties such as their geometrical relationship to the world origin or other objects. Objects also have specific properties, depending on which type of object it is. Light objects have a type of light attached to them, which could be a spot light, point light, etc. Mesh objects have an associated mesh and are linked to a material. A mesh is a collection of vertices, edges and faces which defines the shape of an object. Each materials properties is defined by a data-processing pipeline called a node tree. Camera objects have an attached camera, with attributes such as focal length, resolution and sensor width. The scene has attributes such as specifying which camera is the active camera used for rendering, which render engine to use and more. The compositing node tree specifies post processing steps of the final render, it also specifies which file format the rendered image should be saved to.

Rendering an image

An example scene is shown in 3.2a, which contains a sphere, plain, light and camera. The scene contains objects, lights and at least one camera, such that it is possible to produce a rendered image. Blender passes on the scene information to the render engine, which produces an image by tracing the light paths coming into the chosen camera. The result of rendering is an image with intensities of each color, however the image still needs post-processing to be realistic. The rendered image of the example scene is shown in 3.2b.

Scene graph

The objects' geometrical relationships in the scene are defined using a parent-child relationship, which creates a hierarchical tree structure. A child object can only have one parent object, while a parent object can have multiple child objects.

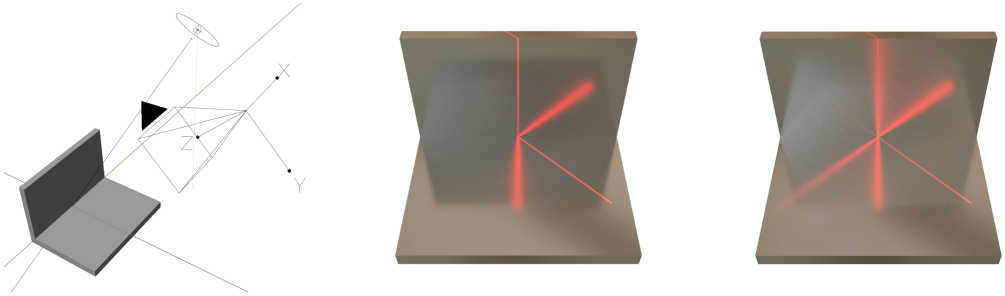


(a) Simple scene **(b)** Rendered image

Figure 3.2.: Simple scene with associated render

3.1.2. Ray tracing engine

The default ray tracer engine in Blender is Cycles. Cycles is a fast physically based renderer, but only supports backward ray tracing. As discussed in 2.5.2, backward ray tracing struggles when it comes to calculating caustics. LuxCoreRender is another engine that is available as an external plugin to Blender, and supports hybrid ray tracing. A scene with a corner mesh, camera and line laser is shown in 3.3a. The same scene is rendered with Cycles and LuxCore as shown in 3.3b and 3.3c respectively. The comparison shows the shortcomings of Cycles being unable to calculate the caustics in the scene, which makes up a significant portion of the reflections. To test the proposed methods in this thesis, LuxCore was chosen to get the most realistic reflections.



(a) Scene **(b)** Cycles render **(c)** LuxCore render

Figure 3.3.: Cycles and LuxCore comparison

3.2. 3D vision systems

The different methods proposed in this thesis, consists of various optical systems of cameras, projectors and lasers. This section explains the overall composition of the systems. How the components of the system were implemented in Blender

and LuxCore are attached in A.2. Each component and system was implemented as a class in python, as it made the most sense to use an object-oriented approach.

3.2.1. Components

Camera

The pinhole camera is already implemented in Blender and LuxCore, however a wrapper class was implemented through the python API, to add additional functionality and simplify the initialization of the camera. The intrinsics of of the camera wrapper class are fully specified by the sensor width, focal length and image resolution in both directions. Additional functionality such as calculating the camera matrix and rendering functions were added.

Projector

LuxCore does not directly have a projector implemented, though it is possible to implement one through a LuxCore spot light with a projected image texture. However, calculating the camera matrix of this spot light projector is convoluted. The calculation of the camera matrix for a spot light projector in LuxCore is added as a section in the appendix in A.1. Additional functionality was added such as loading images to the projector, which were converted to Blender's internal method of storing images.

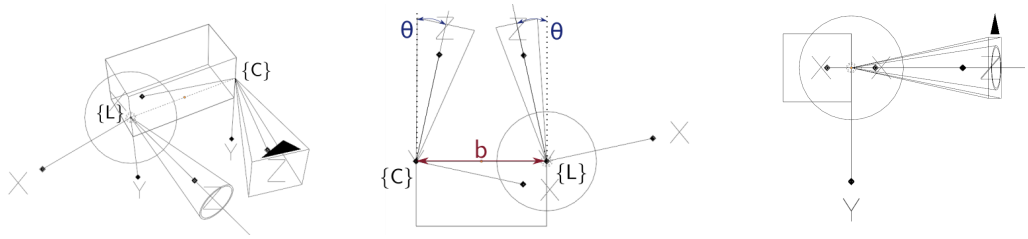
Laser

The laser component was implemented by inheriting the functionality of the projector. The laser line was emulated by projecting an image with a predefined line through the centre, and black pixels elsewhere. Using an image to project the laser line made it easy to control the width, color and appearance of the line.

3.2.2. Scanning systems

Laser scanner

The laser scanner consists of the camera component and the laser component as shown in 3.4. Apart from the intrinsics of each optical device, the laser scanner is defined by the baseline between the optical centres b, and the angles θ as shown in 3.4b.

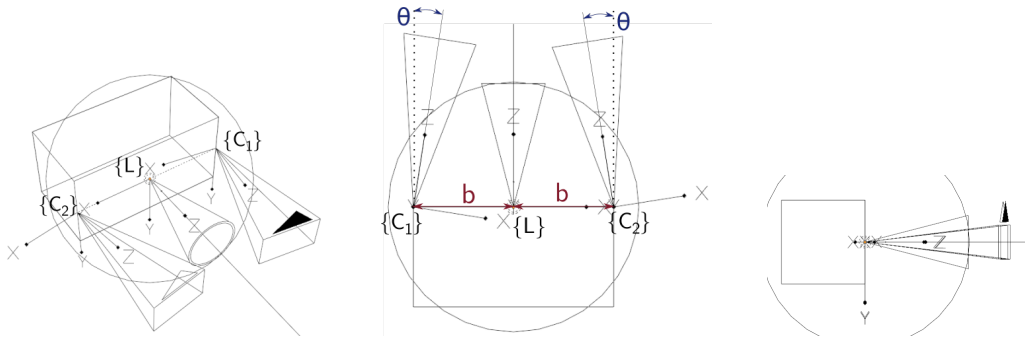


(a) Perspective view **(b)** Top view **(c)** Side view

Figure 3.4.: Laser scanner

Planar stereo laser scanner

The planar stereo laser scanner shown in 3.5, has an extra camera compared to the standard laser scanner. The scanner is symmetrical from the top view, and is defined by the distance from each camera to the laser b, and the angle between the laser and the camera θ as shown in 3.5b.

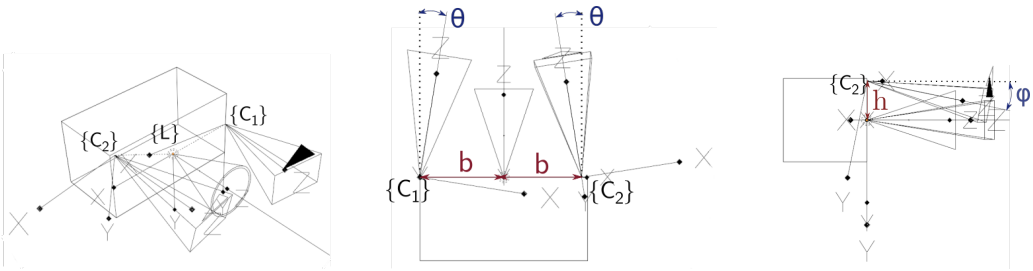


(a) Perspective view **(b)** Top view **(c)** Side view

Figure 3.5.: Planar stereo laser scanner

Non-planar stereo laser scanner

The non-planar stereo laser scanner is shown in 3.6, and consists of the same component as the planar stereo stereo laser scanner. As with the previous scanner, the baseline b determines the distance from the laser to each camera in the $x - z$ plane. The scanner has two additional parameters shown in 3.6c, the distance from the $x - z$ plane of the laser h to the second camera, and the angle between the z-axis of the camera and the laser ϕ.



(a) Perspective view **(b)** Top view **(c)** Side view

Figure 3.6.: Laser scanner

3.3. Dataset generation

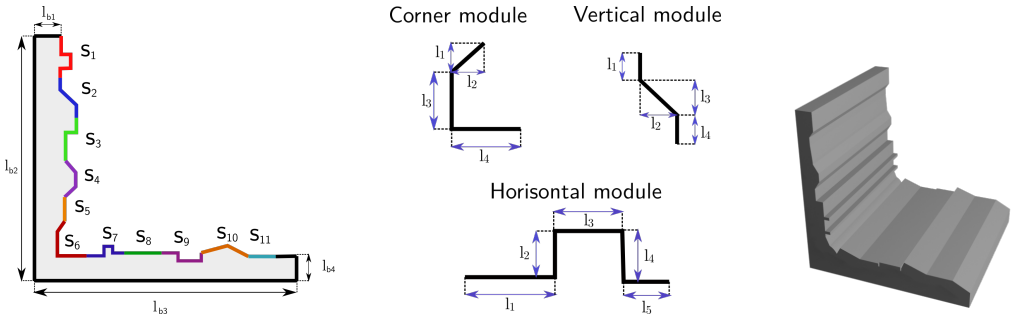
Several datasets were generated for use in training the CNN networks, and for comparing results of the different methods. For a machine learning dataset, it is desirable to generate over 1000 unique images. Manually making different parts, and rendering the images were not feasible. The reflection a render will get is dependent on the geometry of the part, the material of the part and the position of the 3D scanner device.

3.3.1. Mesh dataset

To generate unique parts that will give different reflections, a randomized generation of parts was desirable. With the parametric 3D CAD package CadQuery[27], it is possible to generate 3D models through python. The parts were generated by defining a set of 2D modules which were assembled to generate a corner cross-section as shown in 3.7a. The cross-section consists of a base defined by the constant lengths $\{l_{b1}, l_{b2}, l_{b3}, l_{b4}\}$, and a set of 11 sections $\{s_1, s_2, ..., s_{11}\}$. At each section s_n, a random module is chosen. Each module m is defined by vertical, horizontal, diagonal or curved lines with randomized lengths. Modules were defined for the vertical sections, corner section and horizontal sections respectively. 9 vertical modules, 4 corner modules and 4 horizontal modules were made. Three example modules of each type is shown in 3.7b, where the lengths $\{l_1, l_2, ..., l_n\}$ are randomized in a uniform interval. Once the cross-section was defined by the set of random modules, it was extruded a constant length to get a 3D part as shown in 3.7c.

Material generation

The materials used were generated in 2 different ways. One of the methods used Luxcore presets, along with Blenders node system to generate materials with dif-

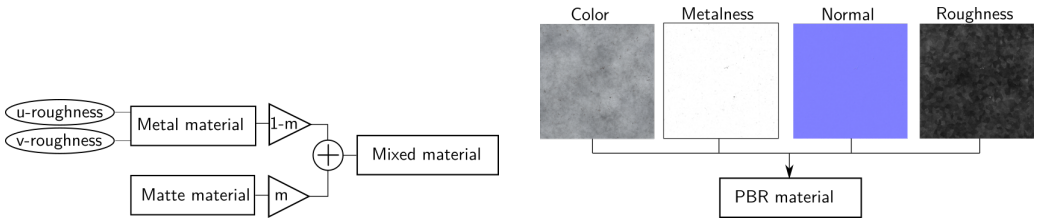


(a) Cross section generation **(b)** Example modules **(c)** Extruded part

Figure 3.7.: Randomized mesh generation

ferent reflections. Blenders node system is a simplified and visual way of assigning outputs of one function to the input of the next. The node system used is shown in 3.8a, and is a mix of the *Matte* material preset and the *Metal* material preset. The matte preset has a diffuse surface, while the metal material has a spread or specular reflection depending on the roughness parameters. The roughness parameters are split up into 2 perpendicular directions, u and v. Low u and v values creates a specular surface while high u and $v$values creates a spread surface. If u and v values differ, the material is anisotropic, meaning it is rough in one direction and smooth in the other. This mix material consists of 3 parameters, u-roughness, v-roughness and the mix factor m.

The other type of materials were assigned from PBR texture images. PBR materials are made of a series of images defining color, roughness, normals and metalness as shown in 3.8b. These materials are typically generated in advanced softwares for material generation. The PBR materials used in this thesis consist of 40 PBR textures downloaded from [1].



(a) Mixed metal material generation **(b)** PBR material generation

Figure 3.8.: Material generation

3.3.2. Render datasets pipeline

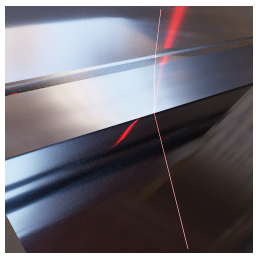
Now that a method to developed semi-random geometries is defined, the final steps to get unique images of different reflections, is to define the image capturing positions of the scanners and the materials of the object. The rendering pipeline starts out by loading a unique mesh in the origin of the scene. To increase variation, each mesh is only used for one render. The next steps in the rendering pipeline are described in the following sections.

Scanner positioning

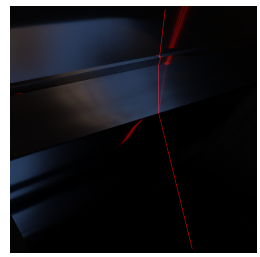
The following step in the rendering pipeline is to load one of the scanner systems described in 3.2.2. The scanner is positioned using spherical coordinates, randomly varying the angles and radial distance from the scene origin, within a predefined range. The scanner is then oriented such that it faces the origin of the scene, where the corner of the mesh is.

Exposure

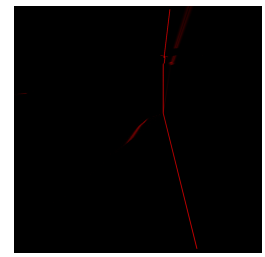
In the context of cameras, exposure refers to the amount of light hitting the sensor of the camera. It is possible to control the exposure of a camera in different ways. The time it takes for a camera to take an image is called shutter speed. Lowering the shutter speed limits the duration light passes into to camera, thus reducing exposure. Another method to limit the light hitting the sensor of the camera is to attach a dark filter in front of the camera. In Blender the exposure can be set directly, such that only stronger light is captured in the image. The advantage of lowering the exposure is that the background is removed, such that only the laser line is visible and can more easily be processed in further stages. Different exposure settings in Blender are shown in 3.9a, 3.9b and 3.9c where lowering the exposure of the camera gradually removes the weaker light of the background.



(a) Exposure 0



(b) Exposure -5



(c) Exposure -8

Figure 3.9.: Adjusting exposure

Ground truth

The ground truth image is defined as the image of the laser line where there are no second order reflections. The ground truth image without reflections can easily be obtained, by disabling second order reflections in the LuxCore API. To get the ground truth subpixel accuracy, the weighted centre of mass, described in 2.3.2, was applied to the ground truth image.

Datasets

A model selection process in machine learning typically includes three datasets, a training dataset, a validation dataset and a test set. The training set is the dataset in which the model is trained through backpropogation. The model is not trained on the validation set, but the accuracy is checked against the validation set throughout the training. A good score on the validation set means the model is more likely to generalize to the problem, and has not overfitted the training set. In this thesis, the time point in which the model had the highest accuracy on the validation set was chosen. Since we have chosen a model based on the validation score, the model may have some bias against the examples in the validation set. The final accuracy is checked once against a test set, to minimize the bias against the specific examples. The results from the test set are used to compare the different methods in the thesis. The number of images in each dataset is summarized in 3.1. The training set consists of 3600 images for each method, the validation set 400 and the test set is 300. The test set was split into three, depending on the type of material used. The PBR images were generated by assigning the image textures as described in 3.3.1. A blurry and specular test set were generated with the mixed material described in 3.3.1. The blurry dataset consists of materials that cause blurry reflections, and the specular test set has more sharp specular reflections. The parameters for the mixed material to generate the blurry and specular datasets are shown in 3.2. Half of the images in the training and validation set were assigned PBR materials, and the rest were assigned a mixed material with semi-random parameters. The values were randomized in $\log_{10}$ based on the minimum and maximum values.

3.4. Accuracy metrics

To quantify the accuracy of the different methods, three metrics for comparison are defined. The metrics are dice score, mean subpixel accuracy and outlier fraction. Dice score measures the overlap between the prediction and ground truth in the segmentation problem, the mean subpixel error measures each methods ability to find the centre of the scan line and the outlier fraction counts the number of

Dataset	Dataset length
Training set	3600
Validation set	400
Test set PBR	100
Test set specular	100
Test set blurry	100

Table 3.1.: Datasets

Parameter	Train/validation		Specular		Blurry	
	min.	max.	min.	max.	min.	max.
m	0.001	0.003	0.0015	0.002	0.0015	0.002
u-roughness	0.001	0.2	0.001	0.008	0.05	0.2
v-roughness	0.001	0.2	0.001	0.008	0.05	0.2

Table 3.2.: Mixed material parameters

outliers outside a given threshold.

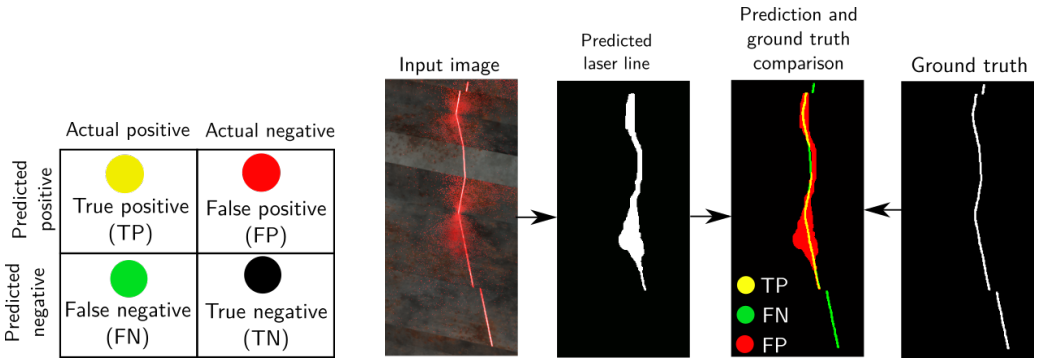
3.4.1. Dice score

The dice score was chosen to evaluate the pixel-wise overlap between the prediction and the ground truth. The dice score measures the overlap between two sets A and B as described in 2.4.8. However, the dice score does not consider how far away or close to the actual scan line a wrongly classified pixel is. In the segmentation problem, the overlap between the ground truth and prediction is of interest. The dice score is calculated based on the confusion matrix shown in 3.10a with

$$\frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \tag{3.1}$$

A *true positive*(TP) is where the prediction and ground truth have a positive value, meaning the prediction is correct. A *false positive* (FP) is a incorrect prediction of a positive output, while a *false negative* is the opposite, incorrectly predicting a negative output. A *true negative*(TN) is where the prediction and ground truth both have a negative output, which is a correct prediction. An example scanning process is shown in 3.10b, where there is an input image that has a semantic segmentation prediction of the scan line. The predicted laser line is compared against the ground truth. The overlap of the prediction and ground truth is shown in yellow, being true positives. The pixel positions where the prediction

has predicted the laser line, but the ground truth is negative, is shown in red, being false positives. The pixel positions where the ground truth is positive, while the prediction does not predict the laser line is shown in green, being false negatives.

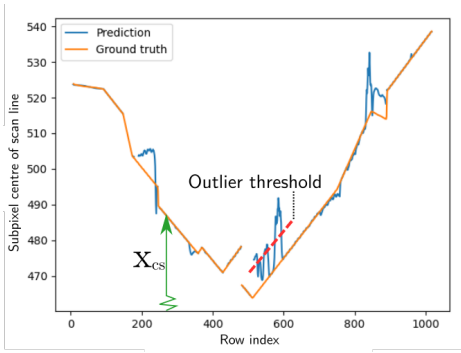


(a) Confusion matrix **(b)** Dice score from prediction and ground truth

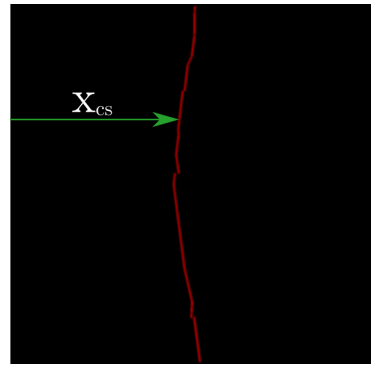
Figure 3.10.: Confusion matrix and dice score

3.4.2. Mean subpixel accuracy and outlier fraction

Given an example scan line as shown in 3.11b, the column index of the centre of the scan line x_{cs}, is plotted both for the prediction and ground truth in 3.11a. The mean subpixel accuracy is calculated by taking the mean absolute distance between the prediction and ground truth, within a given threshold. The subpixel accuracy is only calculated for predictions that are below this threshold, and the rest if classified as outliers. The outlier fraction is the percentage of the measurements outside the outlier threshold shown in 3.11a. The outlier fraction is counted when there exists a row with a ground truth, and there is a prediction in the corresponding row. The outlier threshold chosen was 5 pixels from the centre of the scan line.



(a) Subpixel accuracy and outlier threshold



(b) Ground truth scan

Figure 3.11.: Determining subpixel accuracy and outlier fraction

Chapter 4.

End-to-end CNN

The following chapter explores the capability of U-Net to eliminate reflections using the standard laser scanner system described in 3.2.2.

4.1. Implementation

4.1.1. Datasets

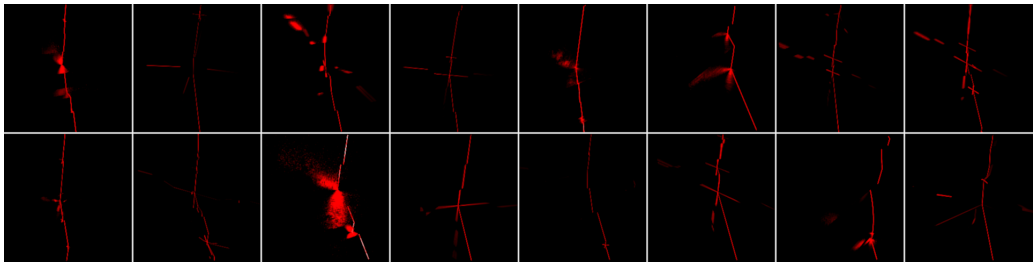
The datasets were generated with 4300 different meshes, and rendering 4300 images as described in 3.3. Rendering all the datasets took approximately 120 hours with a RTX 3090. The resolution was chosen to be 1024×1024 pixels.

4.1.2. U-Net model

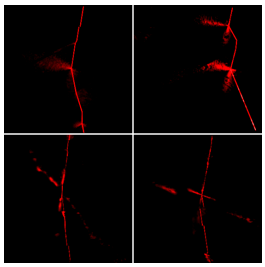
The U-Net model was implemented in Pytorch according to the figure in 2.13, with the exception that same convolutions were used instead of valid convolutions in the convolutions layers and the input images have resolution 1024 by 1024. The contracting path has a total of ten 3×3 convolutional and four 2×2 maxpool layers with stride 2. Using 2.50, gives a receptive field of 140. Each output pixel is therefore a function of the 140 surrounding pixels of the input, in the contracting path. This implies that the model can not evaluate the whole image when determining if a specific pixel belongs to the true scan line, but only a local patch around each predicted pixel.

4.1.3. Training and hyper parameters

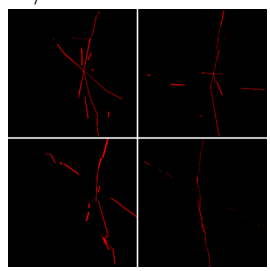
The loss function used was a weighted cross entropy loss, with weighing the laser scan line pixel predictions at 9 times the loss of classifying a pixel as the background class. Weighing the laser scan line predictions considerably higher than



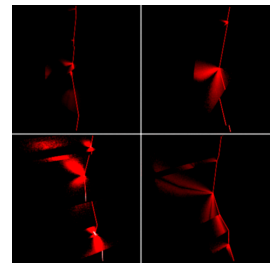
(a) Train/validation dataset samples



(b) PBR test set samples



(c) Specular test set samples



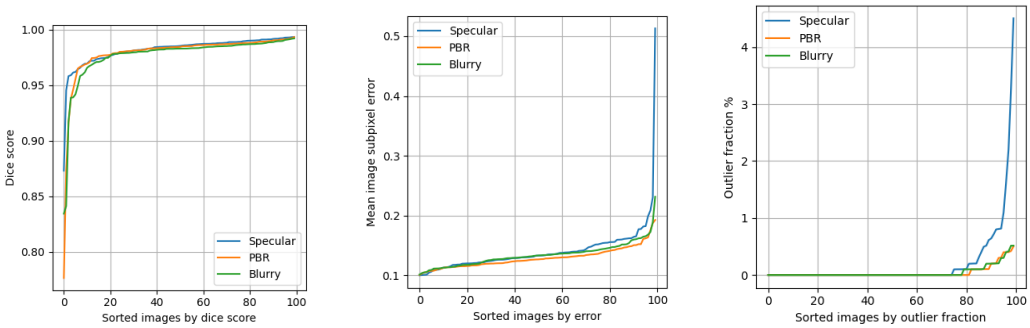
(d) Blurry test set samples

Figure 4.1.: Dataset samples

Hyperparameter	Value(s)	Note
Learning rate	0.001	
Adam betas	0.99, 0.999	Pytorch default
Batch size	2	
C.E. weight background	0.1	Cross entropy loss
C.E. weight laser	0.9	Cross entropy loss

Table 4.1.: Hyperparameters

the background class was an important step for stable training. Weighing the two classes equally resulted in the U-Net model hitting a local minimum of only predicting the background class for the whole image. A low batch size of 2 since it was the limit the GPU could store in its memory alongside the model. The U-Net model was trained for 10 epochs, which had a total duration of about 5 hours on a RTX3090 GPU. The model with the highest dice score across the 10 epochs was chosen as the final model to be compared on the test sets.



(a) Dice score **(b)** Mean subpixel error **(c)** Outlier fraction

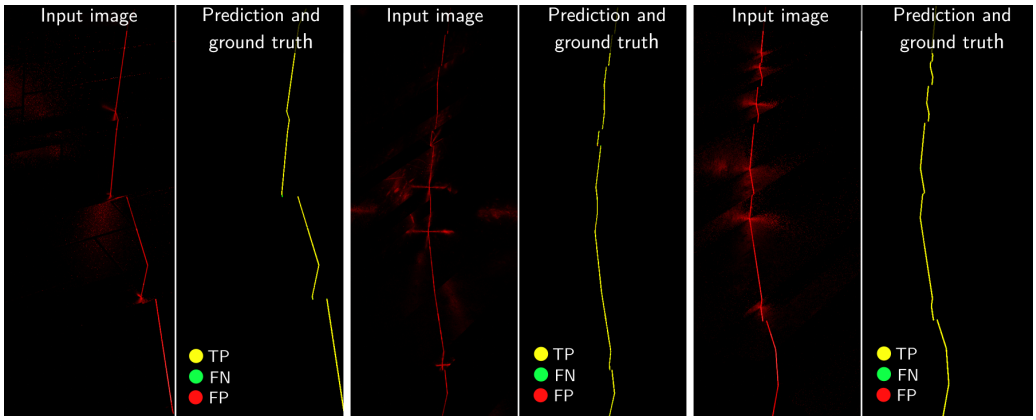
Figure 4.2.: End-to-end machine learning test set results

Test set	Dice score	Mean subpixel error	Outlier fraction
PBR	0.979	0.13 pixels	0.042%
Specular	0.982	0.14 pixels	0.20%
Blurry	0.979	0.13 pixels	0.046 %

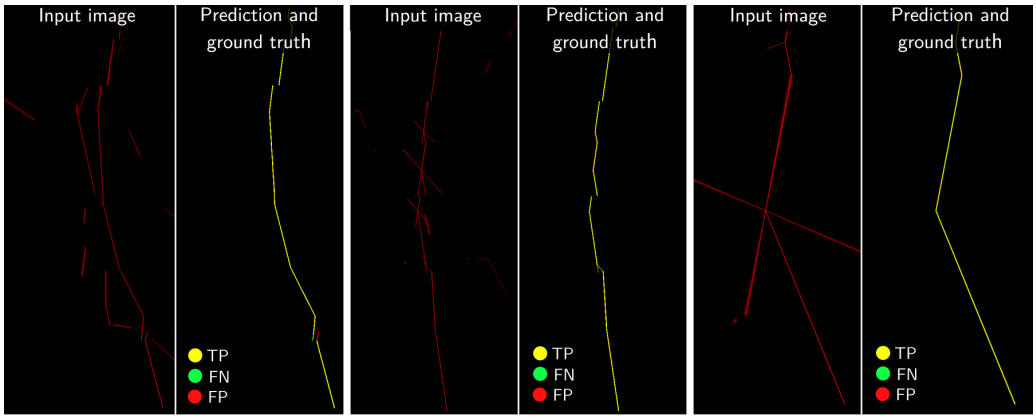
Table 4.2.: End-to-end machine learning averaged results

4.2. Results

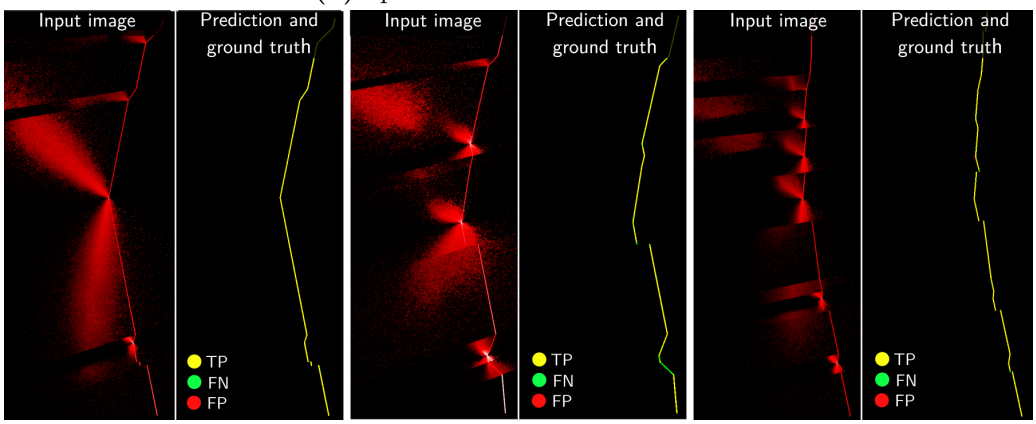
The final model was at last evaluated on the 100 images from each of the test sets. Three examples where the input image is compared with prediction and ground truth, from each of the test sets are shown in 4.3. Additional prediction images are attached in the appendix in B.2. The scores from each image in the test sets are shown in the graphs in 4.2. The dice score graph in 4.2a is sorted on increasing dice score, and shows that the dice score was similar across the test sets. The mean subpixel error graph in 4.2c is sorted on increasing error, and also performed similarly across the test sets. The plot shows that there is a large mean subpixel error for a few predictions in the specular test set. The outlier fraction graph in 4.2c, shows that most predictions had no outliers across all the test sets. For approximately a quarter of the predictions that had outliers, the specular test set had considerably higher percentage than the blurry and PBR test set. The average scores against the 100 images of each test set are summarized in 4.2



(a) PBR test set U-Net results



(b) Specular test set U-Net results



(c) Blurry test set U-Net sampled results

Figure 4.3.: U-Net prediction examples from end-to-end machine learning

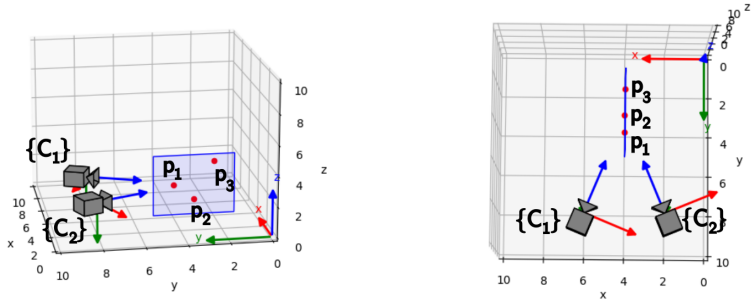
Chapter 5.

Geometric Consistency

The following chapter shows how geometric consistency can be used to remove a significant amount of reflections. The system described in 3.2.2 is used, which is the system with two cameras and one laser. The idea behind geometric consistency is to check which measurements from both cameras are consistent with each other. Geometric consistency is similar to what [26] refers to as data consistency, but with the cameras viewing the workpiece from roughly the same angle and on the same device.

5.1. Consistency from two views

Consider the scene with two cameras, a plane, and three points shown in 5.1, where the geometric relationship between the cameras and plane is known. All points lie on the plane as shown in 5.1b. The three points are given to be visible when each camera is capturing an image. The triangulation equation from 5.1 is



(a) Side view **(b)** Top view

Figure 5.1.: Scene with cameras, plane and points

given as

$$\boldsymbol{p}_n = -\frac{u_4 \boldsymbol{s}_n}{\boldsymbol{u} \cdot \boldsymbol{s}_n} \tag{5.1}$$

which implies that each camera could reconstruct the points independently of the other camera, when the geometric relationship to the plane is known. Denote the points $\boldsymbol{p}_n$ for $n = 1, 2, 3$ viewed in each camera frame, as normalized image coordinates as $\boldsymbol{z}_n$ in camera 1, and $\boldsymbol{q}_n$ in camera 2. Given the transformation matrix between the world frame and camera frames, the following relationship must hold for points visible from both cameras

$$-T_{W,C1}\frac{u_4 \boldsymbol{z}_n}{\boldsymbol{u}_{C1} \cdot \boldsymbol{z}_n} = -T_{W,C2}\frac{u_4 \boldsymbol{q}_n}{\boldsymbol{u}_{C2} \cdot \boldsymbol{q}_n} \tag{5.2}$$

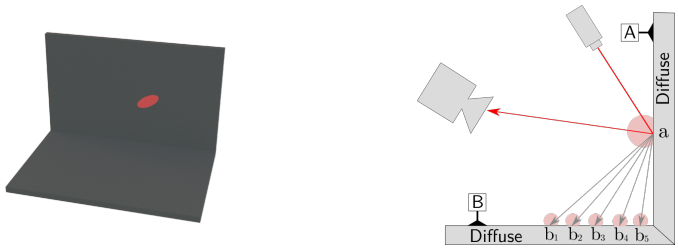
since each point can be reconstructed from one of the cameras. The question then remains, what can be concluded if a point is not consistent from both views? One possibility is that the view to the point is occluded from one camera. However the focus of this chapter is to explore how reflections from a laser are geometric consistent from two cameras, and exploit this to filter them.

5.2. Reflections and consistency

Before discussing geometric consistency of reflections, a definition of the different type of reflections the camera is needed. The light arriving at the camera either comes directly from the light source, or interacts with one or more surfaces before reflecting into the sensor of the camera.

5.2.1. First order reflections

First order reflections are the result of one intermediate reflection between the light source and the camera. An object which does not reflect light at all will appear black to the camera, and light that reflects of objects is the reason the camera can see them at all. A corner with a laser and a camera in a scene shown in 5.2. The laser projects a circular shape against the vertical surface A, at point a. The surfaces A and B of the part in 5.2b is diffuse such that there are no reflections at surface B visible to the camera. The diffuse to diffuse reflections, hitting the points $b_1, b_2, ..., b_5$, spreads out the light such that there is no concentrated intensity of light at the surface B.



(a) Render **(b)** Diffuse reflection

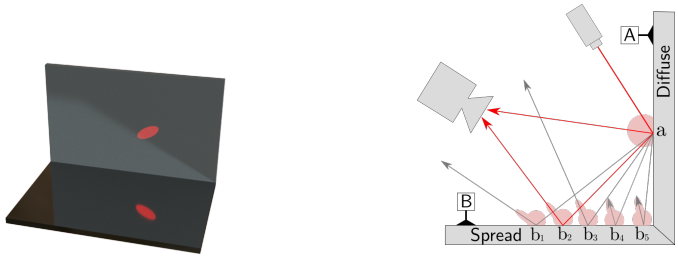
Figure 5.2.: First order diffuse reflection

5.2.2. Second order reflections

A second order reflection has one more intermediate reflection, between the light source and the camera, than the first order reflection. Tracing second order reflections backwards from the camera, the light path interacts with surfaces two times before the path is traced back to a light source.

Diffuse to spread

The same corner mesh as in 5.2.1 is shown in 5.3a, with the same laser shape projected at the vertical surface A. The vertical surface A has a diffuse material while the horizontal surface has a spread reflection. Two red ellipses are shown in the render in 5.3a, the ellipse at the vertical surface stems from a first order reflection while the other at the horizontal surface is a second order reflection. The reflection at the horizontal surface is a spread reflection, which traces back from the diffuse surface as shown in 5.3b at point b_2. The gray arrows in 5.3b causes specular reflections that miss the camera, and are therefore not visible in the captured image. The spread reflection which arises from the diffuse reflection, will be referred to as a *diffuse to spread* reflection.

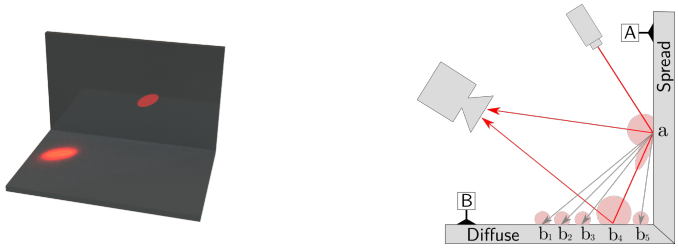


(a) Render **(b)** Diffuse to spread reflection

Figure 5.3.: Diffuse to spread

Specular to diffuse

The same mesh and laser projection as in the previous paragraphs are shown in 5.4. Now the vertical surface A has a material causing spread reflection, and the horizontal surface B is diffuse. Tracing the light from the light source shown in 5.4b, the light hits the vertical surface where most of the energy of the light follows a spread reflection. The concentrated light from the spread reflection hits the horizontal surface which causes a diffuse reflection. The diffuse reflection has enough energy to be visible in the image of the camera as shown in 5.4a

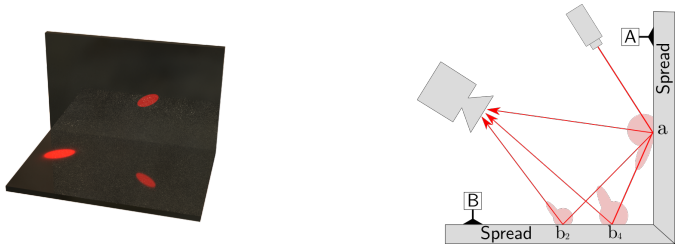


(a) Render **(b)** Top view

Figure 5.4.: Scene with cameras, plane and points

Simultaneous second order reflections

With a material causing spread reflections at both surfaces, both types of second order reflections previously discussed are apparent in the rendered image in 5.5a. The reflection at b_2 stems from the diffuse lobe at the reflection at a, and is a diffuse to specular reflection. The diffuse lobe of the reflection at b_4 is visible, since it stems from the specular lobe of the reflection at a, and is a *specular to diffuse* reflection.



(a) Render **(b)** Top view

Figure 5.5.: Scene with cameras, plane and points

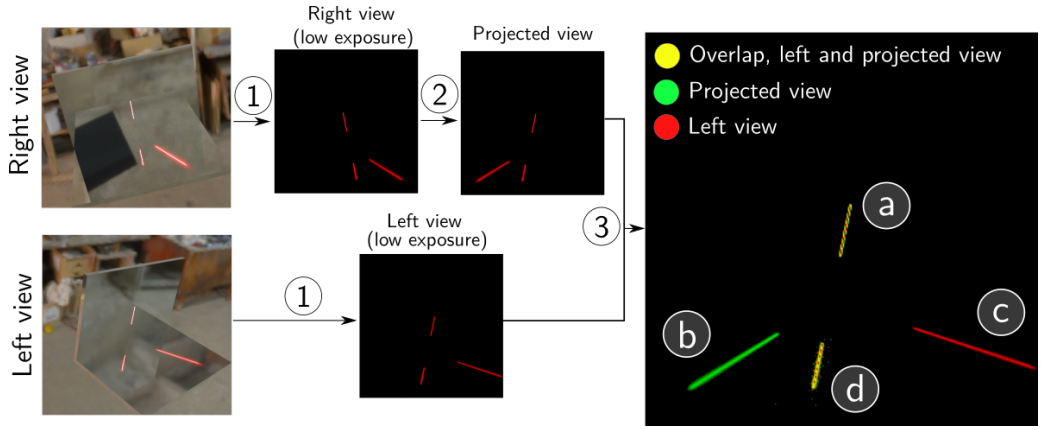
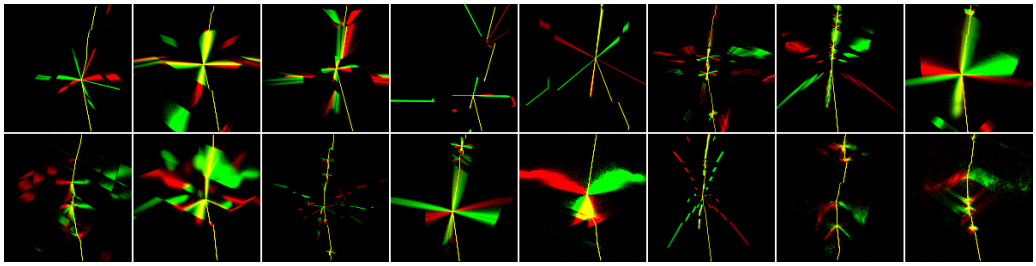


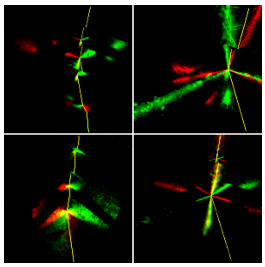
Figure 5.6.: Geometric consistency of reflections

5.2.3. Geometric consistency of reflections

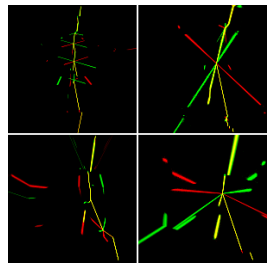
The left and right view in 5.6 shows a similar scene as in 5.5. A small laser line is projected at the vertical surface of the part and the same type of reflections as discussed previously occur. The left and right view in 5.6 are processed in several steps. The step labeled 1 is simply capturing an image with low exposure, such that the background is filtered and the laser line is remaining. In step 2, a homography is applied to the image. The homography is calculated from 2.37, given the geometrical relationship of the laser and right camera. Applying this homography to the image, produces an image that views the laser line taken from the right camera, in the view of the left camera, which will be referred to as the *projected image*. Combining the projected image from the right view, and the image from the left view in step 3, produces the final image in 5.6. The images are combined by producing a three channel RGB image, in which the left view is assigned to the red channel, and the projected image to the green channel. The line b in green, is the spread to diffuse reflections originating from the right view. The red line labeled a, originates from the left view and is the same diffuse to specular reflection as b. The yellow lines a and d overlaps from the left and right view, where a is the real scan line and d is a diffuse to spread reflection. Only the lines in yellow are geometric consistent since the lines are consistent from both views, with the same arguments as discussed in 5.1. It can thus be concluded that lines b and c are reflections since they are not geometrically consistent.



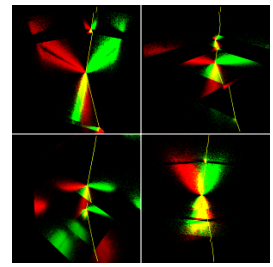
(a) Train/validation dataset samples



(b) PBR test set samples



(c) Specular test set samples



(d) Blurry test set samples

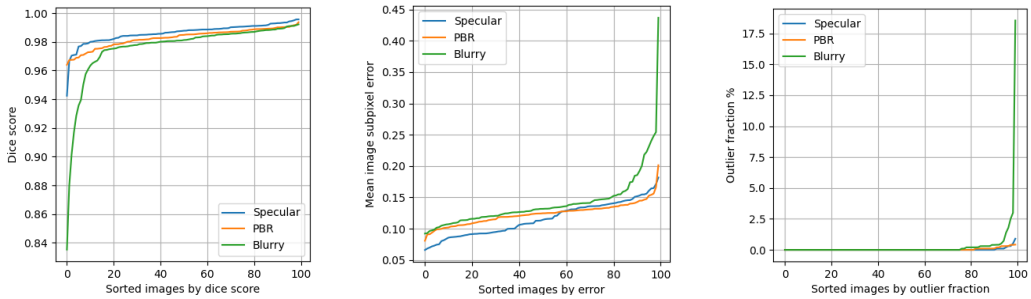
Figure 5.7.: Dataset samples

5.3. Implementations and results

Using the same U-Net model as in 4, with the same training procedure and hyper parameters, the model was trained on a training set of 3600 images with overlapping left and projected views. The model was then tested on each of the PBR, specular and blurry test sets. Example images from each of the training, validation and test sets are shown in 5.7, where the left view is assigned the red channel, and the projected view the green channel of a RGB image. The raw images were used as input to the U-Net model with no pre-processing, as it was assumed the model would find the optimal approach to use the given information in the raw images during training.

The results for each of the images in the test sets are shown in 5.8, and a summarized table of averaged results across the test sets in 5.1. The dice score graph in 5.8a shows that a majority of predictions had a dice score 0.98 or better. The worst predictions were from the blurry test set, scoring as low as 0.84, and having generally worse predictions than the two other test sets. The blurry test set performed worse in the mean subpixel accuracy in 5.8b and outlier fraction in 5.8c. The outlier fraction graph very few outliers across all images for the PBR and specular test set. Three examples where the input image is compared with prediction and ground truth, from each of the test sets are shown in 5.9. Additional

prediction images are attached in the appendix in B.3.

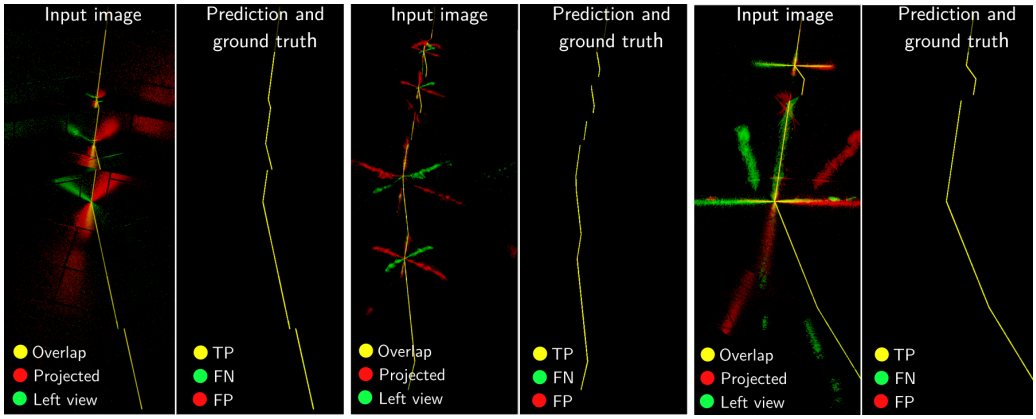


(a) Dice score **(b)** Mean subpixel error **(c)** Outlier fraction

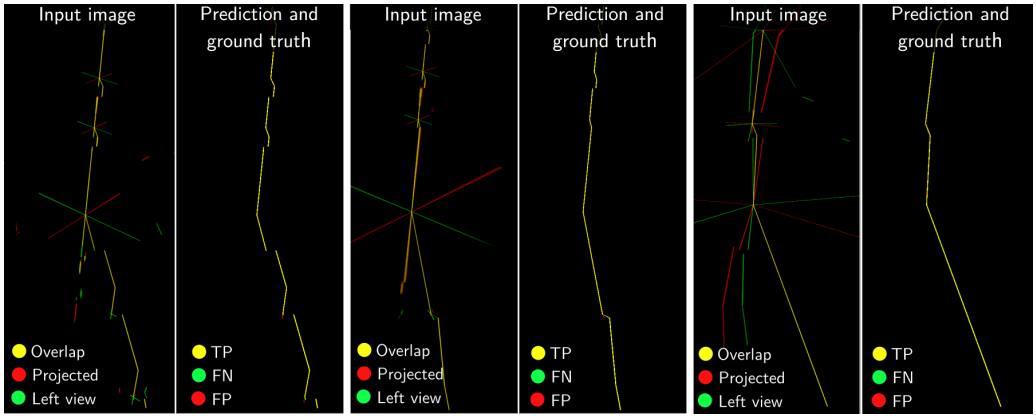
Figure 5.8.: Geometric consistency test set results

Test set	Dice score	Mean subpixel accuracy	Outlier fraction
PBR	0.983	0.12 pixels	0.037%
Specular	0.986	0.12 pixels	0.026%
Blurry	0.976	0.14 pixels	0.33 %

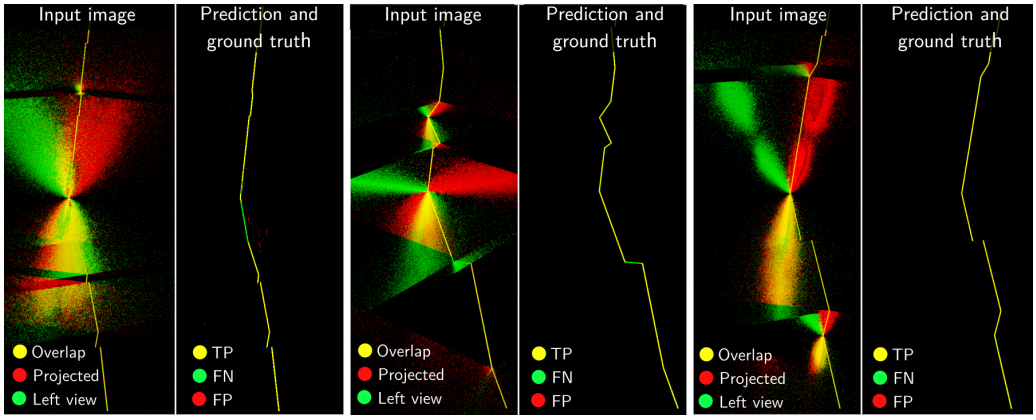
Table 5.1.: Geometric consistency averaged results



(a) PBR test set U-Net results



(b) Specular test set U-Net results



(c) Blurry test set U-Net sampled results

Figure 5.9.: U-Net prediction examples from geometric consistency method

Chapter 6.

Epipolar Consistency

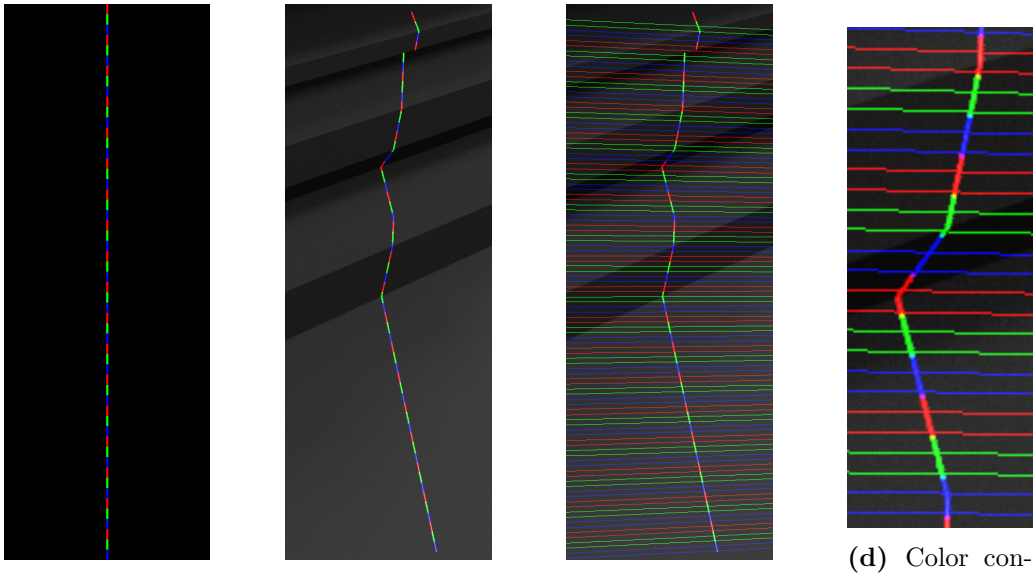
The last chapter explored verification of the scan line through the geometric consistency of an additional camera. The following chapter explores incorporating information and verification through epipolar geometry. If there exists a laser line, such that a camera can distinguish the line continuously or separate sections of it, the following chapter explores how this additional information can be used to filter reflections. The method to distinguish segments of the line in this thesis, will be by color.

6.1. Color encoded consistency

To map different sections of the scanned image to the scan line through epipolar geometry, there must exist some encoded information in the scan line. The line chosen for this thesis is shown in 6.1a, with a periodic pattern of red, green and blue. The line projected at an example mesh is shown in 6.1b, where there are no second order reflections. Using the fundamental matrix, which is calculated from the geometry of the camera and laser as shown in 2.2.2, the relationship from 2.24 can be used. The laser is denoted in frame 2 while the camera is denoted in frame 1, we then have

$$\boldsymbol{\ell}_n = \boldsymbol{F}^T \boldsymbol{p}_n \tag{6.1}$$

for each pixel $\boldsymbol{p}_n$ in the laser scan image. Coloring some lines $\boldsymbol{\ell}_n$ with the color the line originates from in pixel $\boldsymbol{p}_n$ we get the image shown in 6.1c. A zoomed region of the image is shown in 6.2c, where it is shown that the colored epipolar lines match color with the projected laser line in the scan image.



(a) Laser line **(b)** Scan image **(c)** Epipolar lines **(d)** Color consistency

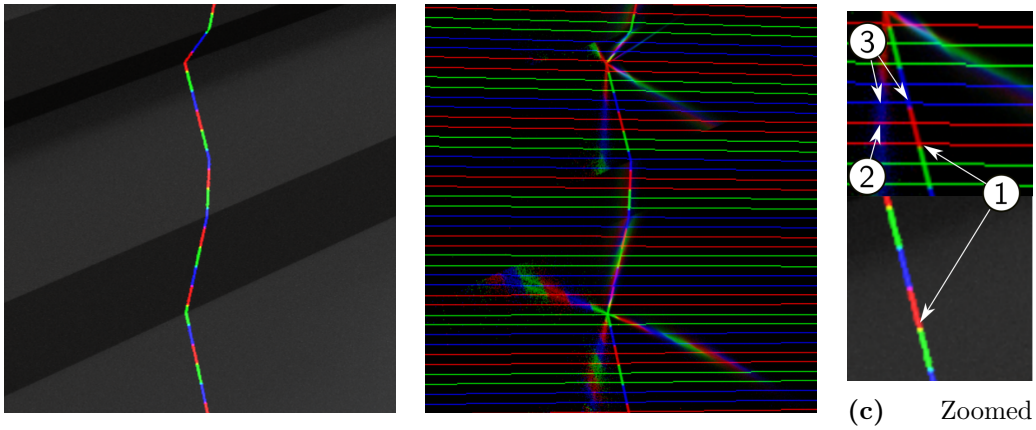
Figure 6.1.: Color encoded laser scan without second order reflections

6.2. Color encoded consistency and reflections

A zoomed in region of the the scan image in 6.1b is shown in 6.2a, with the same projected colored laser line. Assigning a reflective material and rendering reflections results in the scan image in 6.2b, where colored epipolar lines are drawn, at a set interval, with the same process as previously discussed. The images in 6.2c compares the same region of the images, with and without reflections. Three observations of interest are labeled in image 6.2c. The first observation (*1*) shows that the epipolar line is consistent with the real scan line, where the arrows point to the same pixel in the scan line in both images where the color match with the epipolar line. The second observation (*2*) points to a pixel where the reflection is blue, and the epipolar line is red. Since the color of the measurement does not match with the color of the epipolar line, it can be concluded it is not the true scan line. For observation (*3*) the epipolar line matches color with both the true scan line and the reflection, which shows that all reflection can not be filtered with epipolar consistency.

6.2.1. Epipolar filtering process

Using the color encoded consistency discussed in the last chapter, a scan image can be filtered where the colored epipolar lines do not match up. A distance metric in hue, saturation and range could possibly be checked against the epipolar lines



(a) No reflection scan **(b)** Epipolar lines and reflections **(c)** Zoomed view

Figure 6.2.: Determining reflections

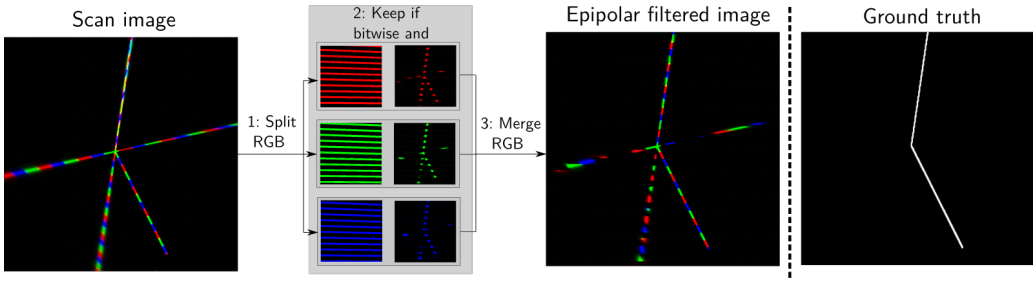


Figure 6.3.: Epipolar consistency filtering

and the scan image, but there are cases where the reflections and the real scan line blend colors. For example, if the real scan line is red at a certain point, and a blue reflection overlaps the real scan line, we get a purple color where the real scan line is. This can be seen in the scan image in 6.3, where the upper part of the scan line is both purple, yellow and cyan. It was therefore assumed that checking the red, green and blue channel against the colors of the epipolar lines was a more consistent approach. Consider the filtering of the colors in the second step named *2: keep if bitwise and* in 6.3. An image of the epipolar lines is generated with their respective colors, and checked against the corresponding color in the scan image. If both images have a value at a given pixel position, the pixel value of the scan line is kept, or otherwise set to zero. At step *3: Merge RGB*, the red, green and blue channels are merged to get the epipolar filtered image. Comparing the filtered image with the ground truth, we see that none of the pixels of the real scan line is filtered, while some pixels of the specular reflections are filtered.

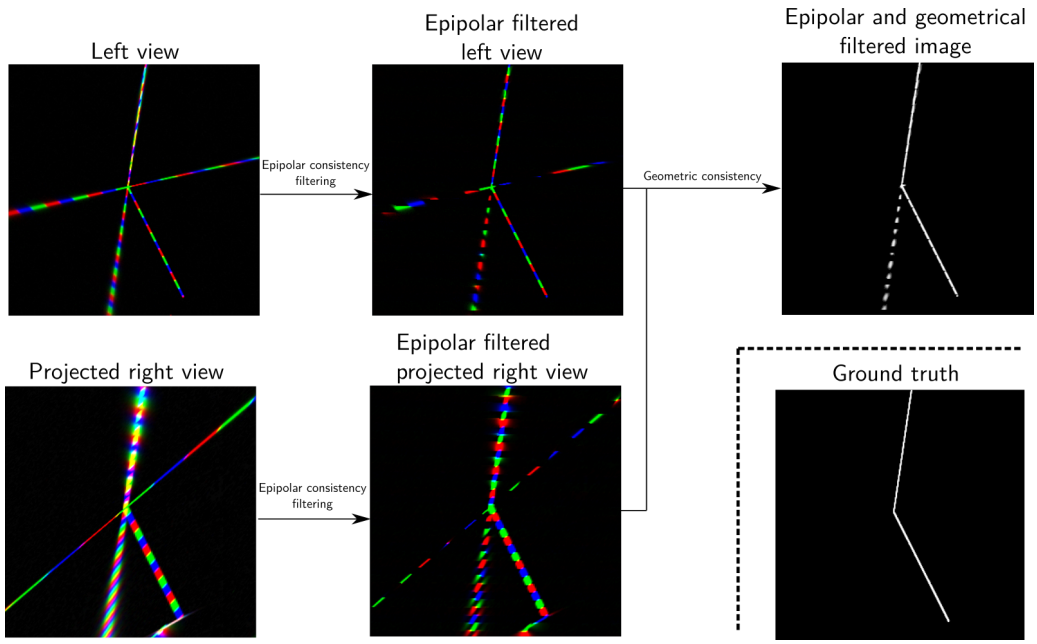
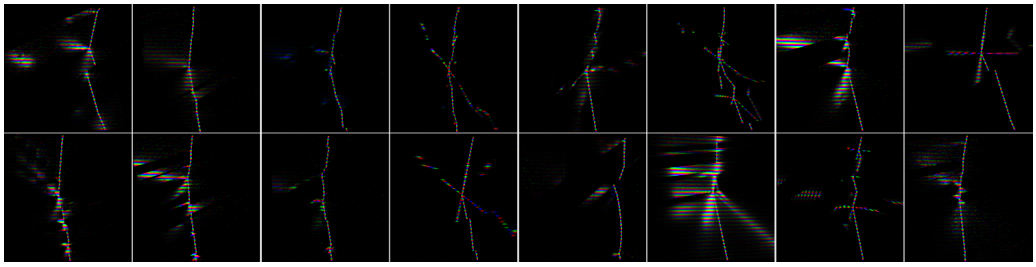


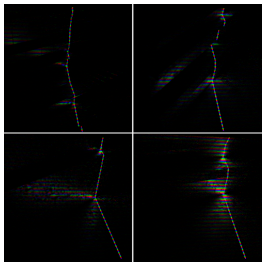
Figure 6.4.: Epipolar and geoemtric consistency filtering

6.3. Epipolar and geometric consistency

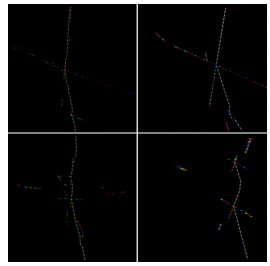
Combining the filtering methods discussed in this chapter and the previous chapter on geometric consistency is possible. To combine the methods, there must be a system using two cameras and a color encoded laser line. The process is shown in 6.4, which starts with each camera capturing a low exposure image of the laser line. The right image is then projected with a homography as discussed in 5.2.3, to get the projected right view. Each image undergoes the epipolar filtering process described in 6.3, to get the epipolar filtered right and left view. The last image labeled *epipolar and geometrical filtered image* is given as an example, where the pixel values of the epipolar filtered images are normalized in the range $[0, 1]$ and multiplied together. Comparing the epipolar and geometrical filtered image with the ground truth, it is shown that most reflections are filtered for the given example. The only reflections that are left is the dotted line which originates from the diffuse to specular reflection.



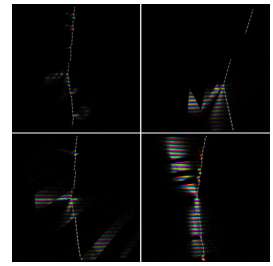
(a) Train/validation dataset samples



(b) PBR test set samples



(c) Specular test set samples



(d) Blurry test set samples

Figure 6.5.: Epipolar consistency dataset samples

6.4. Implementations and results

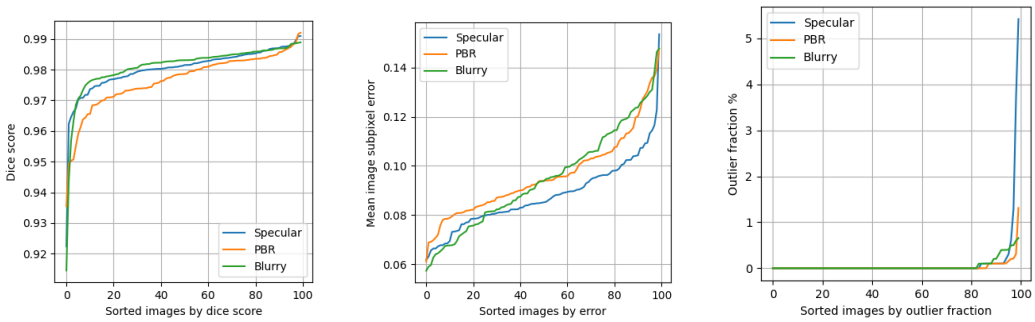
6.4.1. Epipolar consistency

The epipolar filtering method was used as a pre-processing step before given as an input to the same U-Net model and hyper parameters as described in 4. Images were rendered in Blender with the laser scanner projecting a periodical color line with the single view laser scanner system described in 3.4. Training samples from each of the datasets are shown in 6.5, using 3600 images for the training, 400 images for validation and 100 images for each of the test sets.

The result for each of the images in the test sets is shown in 6.6. The dice score graph in 6.6a and mean subpixel error graph in 6.6b shows that predictions from each of the test set had similar results, with a majority of the prediction scoring a dice score above 0.97. The outlier fraction graph in 6.6c shows that approximately 80 predictions from all of the test sets had no outliers, and a few predictions from the specular test set scoring considerably worse than the others. Three examples where the input image is compared with prediction and ground truth, from each of the test sets are shown in 6.9. Additional prediction images are attached in the appendix in B.4.

Test set	Dice score	Mean subpixel error	Outlier fraction
PBR	0.977	0.096 pixels	0.030%
Specular	0.980	0.088 pixels	0.13%
Blurry	0.981	0.095 pixels	0.052%

Table 6.1.: Epipolar consistency average results



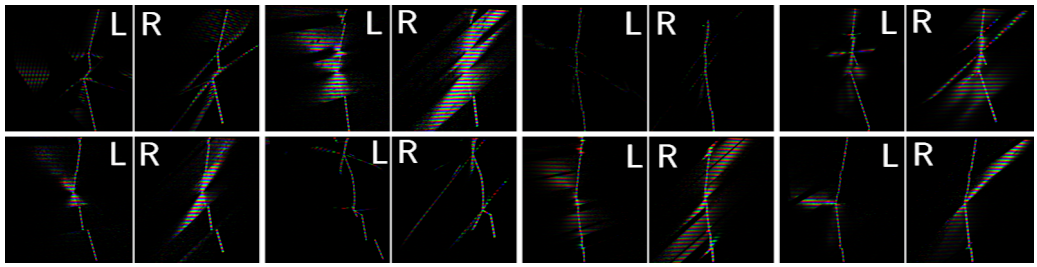
(a) Dice score **(b)** Mean subpixel error **(c)** Outlier fraction

Figure 6.6.: Epipolar consistency results

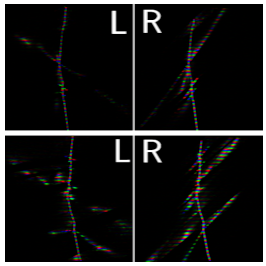
6.4.2. Epipolar and geometric consistency

The training, test and validation set for the epipolar and geometric consistency method were generated in Blender using the scanner in 3.2.2, with samples from each test set shown in 6.7. The epipolar filtered left view and projected right view from 6.4 were used as the input to the U-Net model. The stacked left and right view with 3 color channels each, made up 6 channels together. Apart from using 6 input channels instead of 3, the same U-Net model, and hyper parameters as described in 4 was used during training.

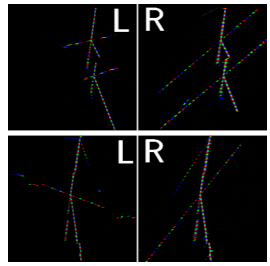
The results for each of the images in the test sets are shown in 6.2, and the summarized averaged results in 6.2. The dice score graph in 6.8a shows that each test set got similar scores, except a few bad predictions in the blurry test set scoring as low as 0.87. The mean subpixel error in 6.8b had similar results across the test sets. Approximately 80 images in each test sets had no outliers as shown in 6.8c, while the last 20 images had a higher percentage of outliers in the blurry test set. Two examples where the input image is compared with prediction and ground truth, from each of the test sets are shown in 6.10. Additional prediction images are attached in B.5.



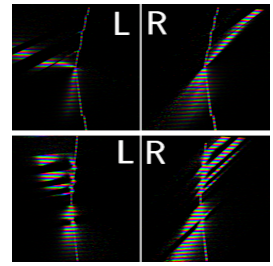
(a) Train/validation dataset samples



(b) PBR test set samples



(c) Specular test set samples

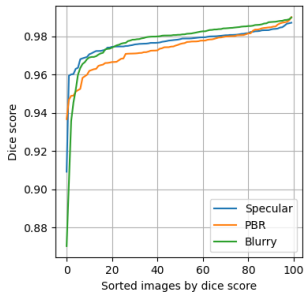


(d) Blurry test set samples

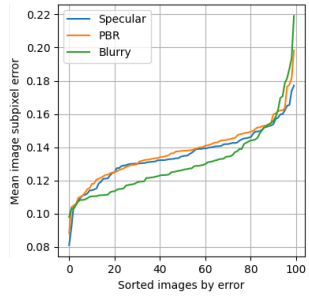
Figure 6.7.: Epipolar and geometric consistency dataset samples. Images labeled L and R are for left and right view respectively

Test set	Dice score	Mean subpixel error	Outlier fraction
PBR	0.974	0.14 pixels	0.0091%
Specular	0.977	0.14 pixels	0.021%
Blurry	0.977	0.13 pixels	0.10%

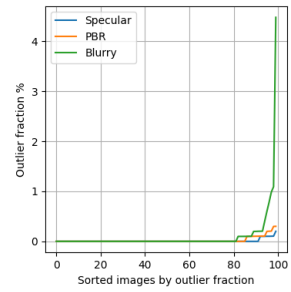
Table 6.2.: Epipolar and geometric consistency average results



(a) Dice score

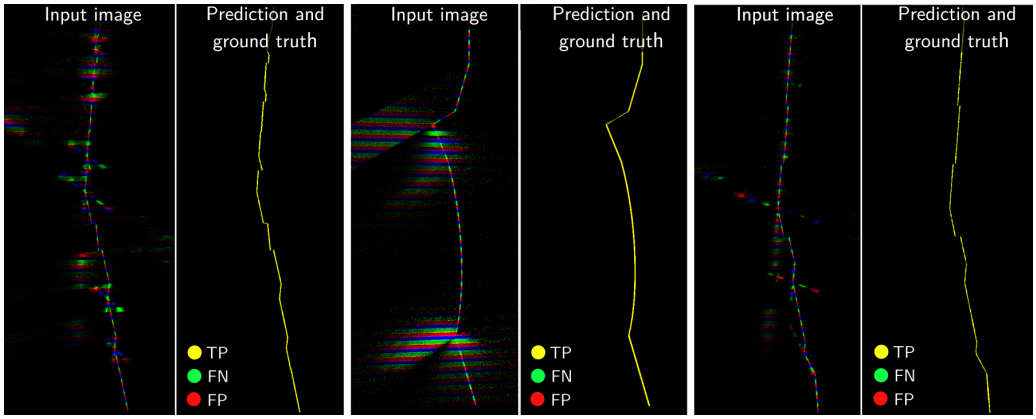


(b) Mean subpixel error

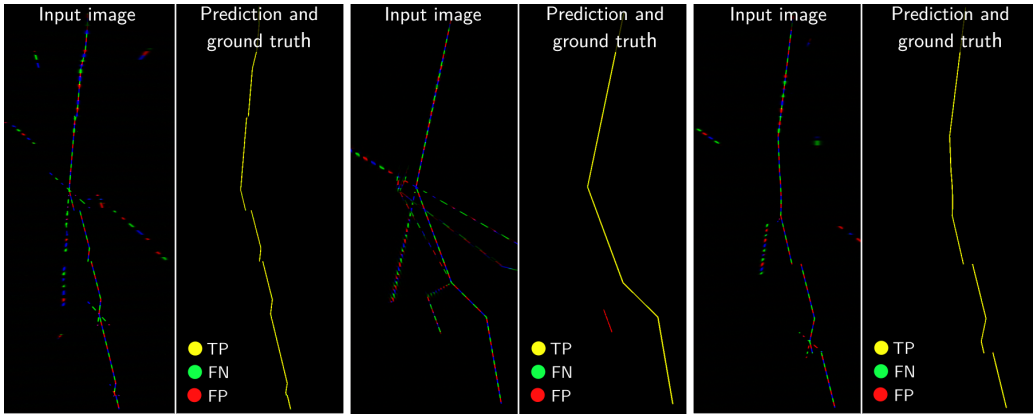


(c) Outlier fraction

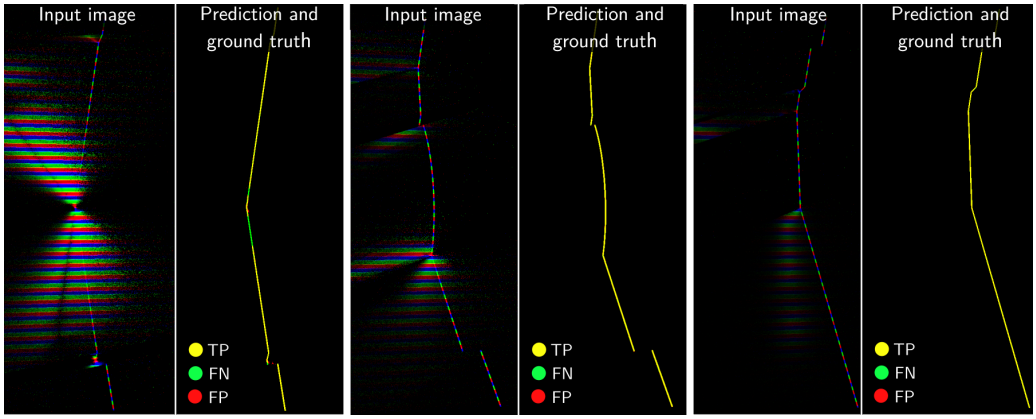
Figure 6.8.: Epipolar and geometric results



(a) PBR test set U-Net results

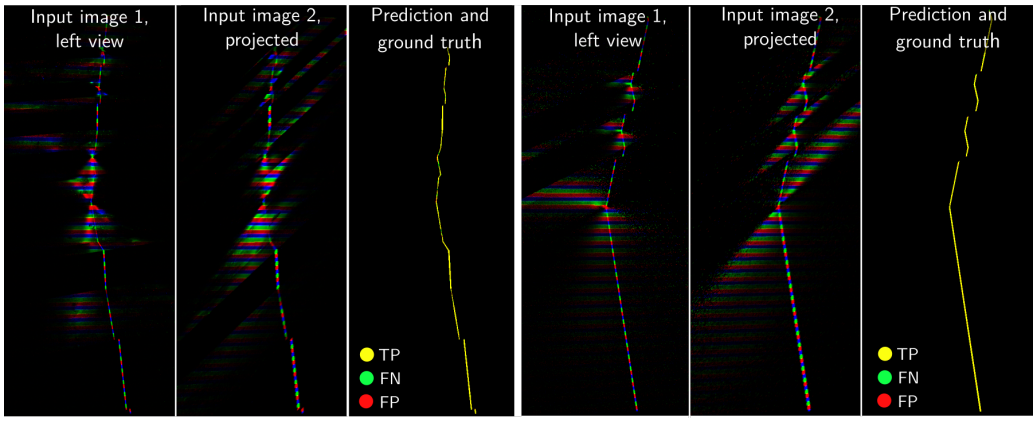


(b) Specular test set U-Net results

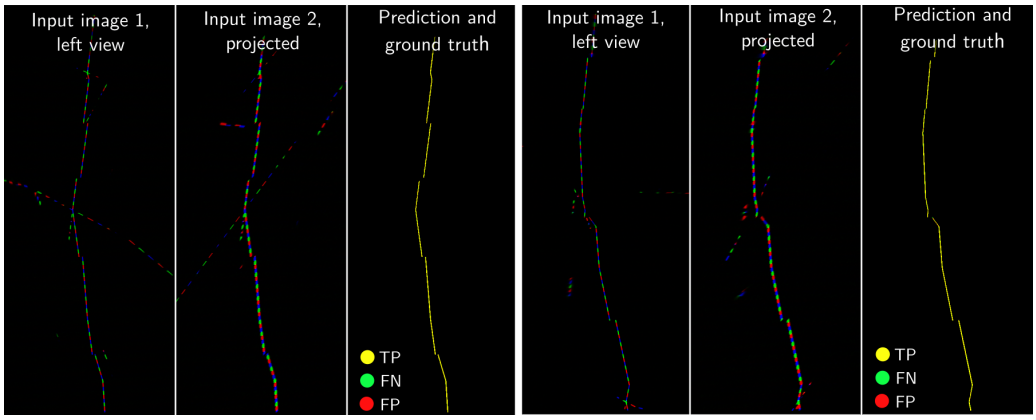


(c) Blurry test set U-Net sampled results

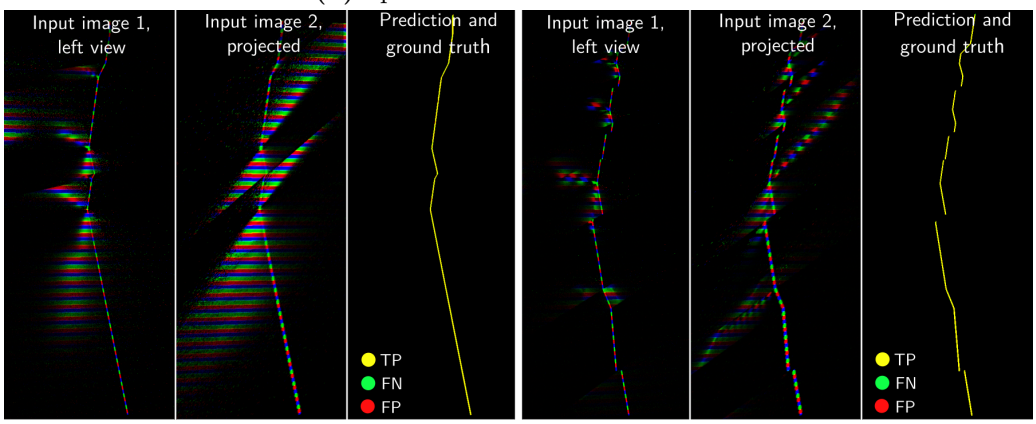
Figure 6.9.: U-Net prediction examples from epipolar consistency method



(a) PBR test set U-Net results



(b) Specular test set U-Net results



(c) Blurry test set U-Net sampled results

Figure 6.10.: U-Net prediction examples from epipolar and geoemetric consistency method

Chapter 7.

Discussion

7.1. Comparing results

The following section will compare the numeric results given in the earlier chapters of each method. The differences in the numeric results are highlighted before the sources of errors are discussed.

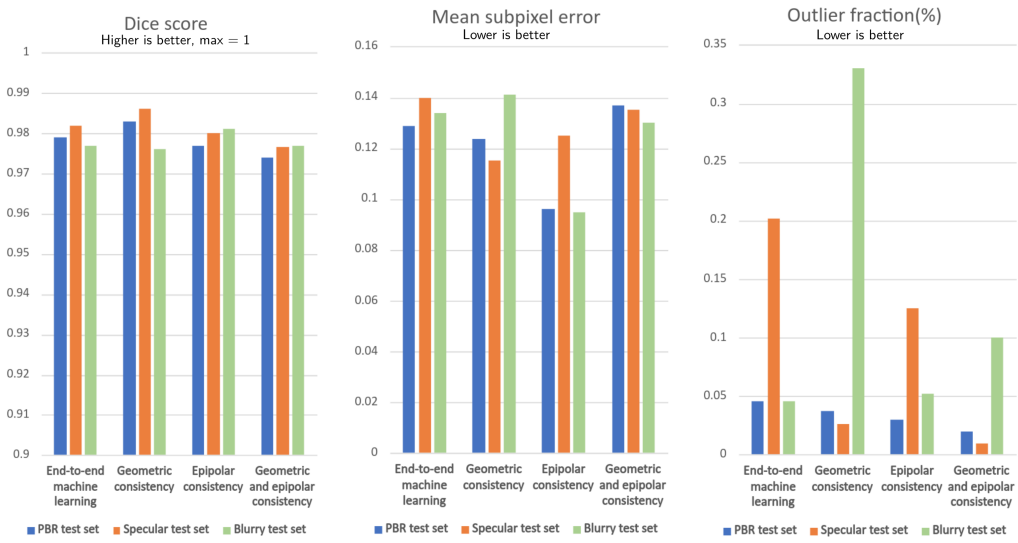
7.1.1. Numeric results

Dice score

The numeric results given in the result sections of earlier chapters are summarized in the bar plots in 7.1. The dice scores, shown in 7.1a, were similar across all methods. All methods scored a dice score of approximately 0.97 across all test sets, which means there is an average of 97% overlap between the prediction and ground truth compared to the union of these sets. It was expected to be a bigger difference between the dice scores across the methods. The end-to-end machine learning approach was expected to perform considerably worse than the others in terms of the dice score, since it only used a single scan image with no further encoded information.

Mean subpixel error

The mean subpixel error, compared in 7.1b, was also similar across the different methods. This indicates that each of the methods was equally capable of detecting the centre of scan line for each row it made a prediction, when ignoring large deviations of outlier measurements.



(a) Dice score **(b)** Mean subpixel error **(c)** Outlier fraction

Figure 7.1.: Numeric results comparison

Outlier fraction

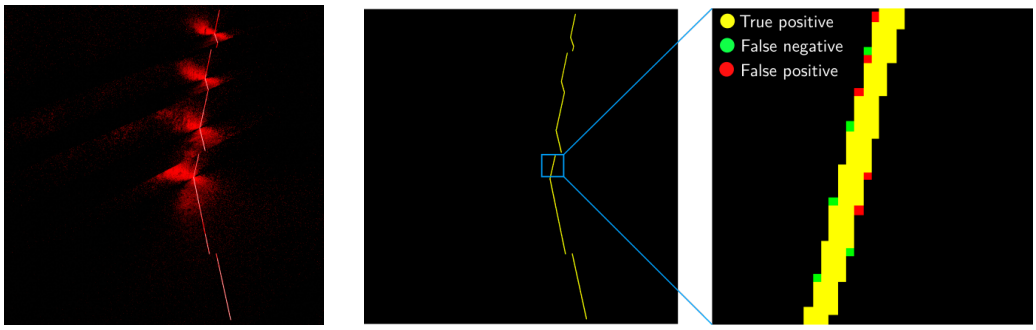
The outlier measurements, counted as an outlier fraction across the rows in the image, is compared in 7.1c. Compared to the dice score and mean subpixel error, the outlier fraction comparison offered a greater variation across the results. The end-to-end machine learning method had a high outlier fraction on the specular test set compared to the PBR and blurry test set. Since the PBR test set contains mostly blurry reflections, the low outlier fraction on the PBR and blurry test set indicates that only using machine learning is sufficient for filtering blurry reflections. Specular reflections caused large outliers when they are mistaken for the real scan line, since they can be far away from the actual scan line in the image. The result indicates that the end-to-end machine learning method had errors classifying specular reflections from the real reflection. The geometric consistency method performed well on the PBR and specular reflection test set, while considerably worse on the blurry test set.

7.1.2. Sources of error

None of the proposed methods were able to correctly determine all the pixels of the real scan line. The next sections will present the sources of errors from the methods.

Imperfect edge detection of scan line

The predictions that upon first glance seemed to have a perfect pixel-wise prediction, achieved a dice score between 0.98 and 0.99. Upon close inspection of the prediction images stacked on top of the ground truth images, it was found that all of the predictions had small errors on the edge of the line. An example from the end-to-end machine learning approach is shown in 7.2. The scan image from the PBR test set is shown in 7.2a, and the prediction stacked upon the ground truth in 7.2b. In the zoomed view of the scan line, the majority of the scan line is correctly predicted, shown in yellow as the true positive predictions. However at the edges of the line, there are isolated pixels that are both false negative and false positive. Small errors like this were present for all methods, and are estimated to make up 1-2% of the error in the dice score across all the test sets. Considering the small variation in the dice score results, the metric could not measure significant differences across the methods since the imperfect edge detection made up a large percentage of the error. Although the errors affected the dice score to a small degree, it is not expected that the imperfect edge detection made a big difference to the mean subpixel accuracy and outlier fraction results. The imperfect edge detection caused wrongly classifying pixels with low values. Pixels with low values do not affect the subpixel accuracy much, since the pixel values are squared when calculated the mean subpixel accuracy. The imperfect edge detection was inside the 5 pixels from the centre of the scan line, which is considered within the outlier threshold.



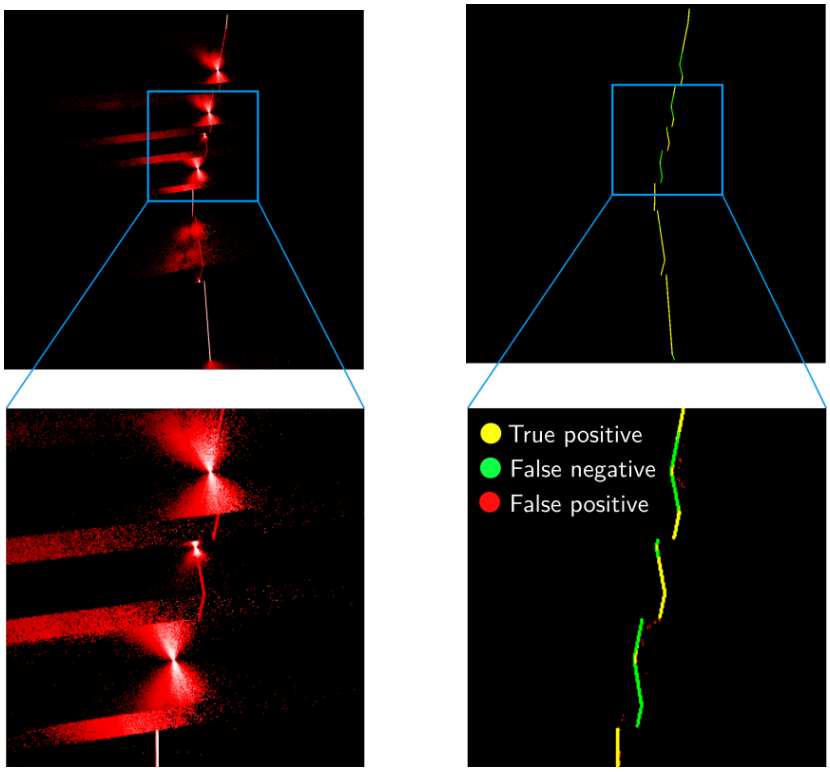
(a) Scan image **(b)** Imperfect pixel prediction

Figure 7.2.: Imperfect detection of scan line

Scan line corrupted by reflection

The main prerequisites for predicting the true scan line with the proposed methods, is the true scan line being distinct and visible in the image. For many of the scans with blurry reflections, the reflections were on top of the true scan line.

This was not a problem if the reflections had weak intensity and the real scan line being stronger. It was however a problem if the reflections had enough intensity such that the true scan line was not visible, such that the real scan line got corrupted. One such case with from the blurry test set is shown in 7.3, with the scan image in 7.3a, and ground truth and prediction image in 7.3b. The problem was prevalent with all the methods on the blurry test set, and to some degree on the PBR test set. The great difference with the outlier fraction result between the end-to-end machine learning and geometric consistency approach is related to this problem. The end-to-end machine learning approach did not make any predictions when the scan line was corrupted, however the geometric consistency method made many false predictions in the area of the false measurement such that the outlier fraction score got worse. The issue may be solved by further reducing the exposure of the camera, since the true laser line may be visible among the reflections when sampling less light.



(a) Scan image **(b)** Prediction

Figure 7.3.: Corrupted scan line by reflection

Specular reflection ambiguity

Visually inspecting images with sharp specular reflections, makes it hard to distinguish the true scan line from the reflections. It was assumed the single view machine learning methods would struggle to some degree with differentiating this as well, but the results were better than expected. The single view machine learning methods made a few minor mistakes. An example of such a mistake is shown in 7.4, where a single view scanner and a stereo view scanner are capturing the same part. In the end-to-end machine learning method, a small specular reflection is mistaken for the real scan line in the zoomed region of the image. With the stereo view scanner and geometric consistency method, the specular reflection is easily distinguished from the real scan line since it does not overlap from both views in the input image. Therefore, it is not geometric consistent.

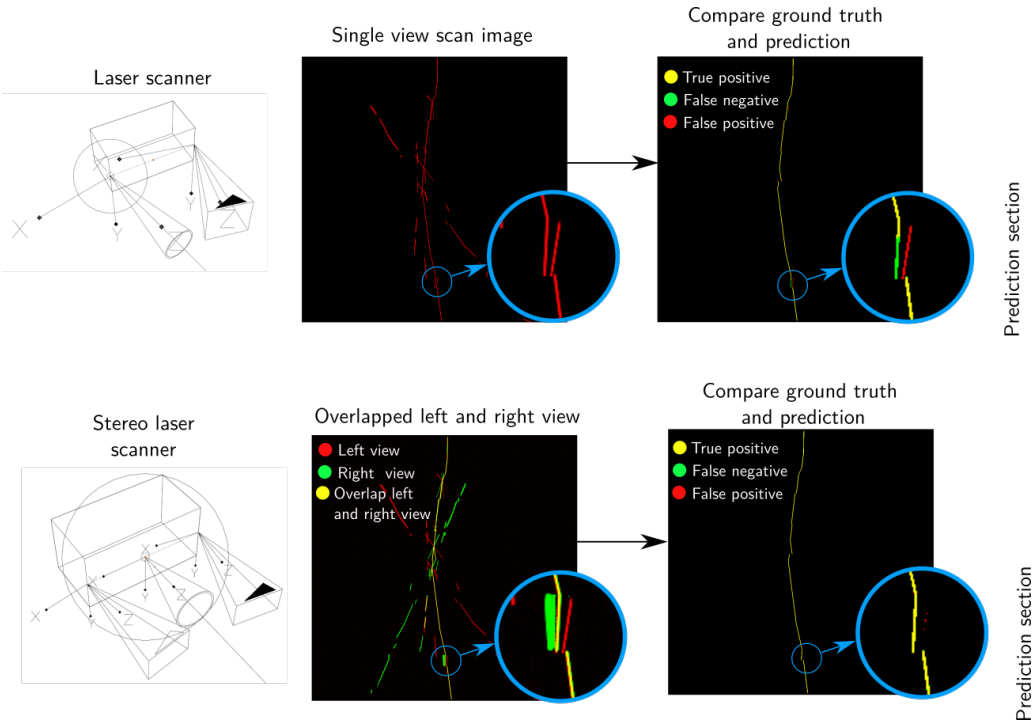
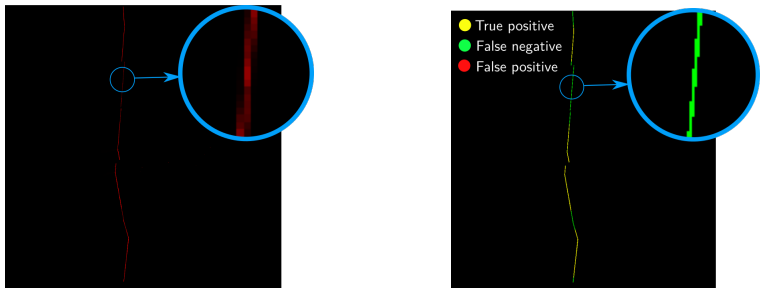


Figure 7.4.: Specular reflection, single and stereo view comparison

Dataset outlier

When training a machine learning network, the weights of the network is tuned by backpropogating the predictions it made from images of the training dataset. If some features from the dataset only exist in a very small portion of the images,

the network may not be tuned to correctly predict these features, since it may not benefit the overall cost function it minimizes. A few cases were found where the thickness of the real laser line was only 1 pixel. Since the laser line was usually from 3-6 pixels wide across most of the images in the training dataset, the network failed to recognize parts of the line that were 1 pixel wide. An example from the end-to-end machine learning PBR test set is shown in 7.5. The image should be easy to predict since there are almost no reflections. However since the laser line is 1 pixel wide, the prediction and ground truth in 7.5b have a large section of false negatives.



(a) Scan image **(b)** Prediction

Figure 7.5.: Dataset outlier

7.2. Machine Learning

The trained U-Net models successfully filtered a majority of the reflections, since the average dice score was over 0.97 for all test sets with all methods. The question is then how the model was able to distinguish the reflections from the real scan line in so many cases. The behavior and interpretability of machine learning models have generally been difficult to understand, and are an ongoing field of research today. Some tools exist for visualizing kernels and feature maps for convolutional neural networks, however no in-depth analysis was carried out. The following sections will discuss features the model may have used to distinguish reflections from the real scan line. In the context of machine learning tasks, human level performance is often mentioned as a metric to compare the performance of a neural network. Therefore, it is reasonable to discuss the features a human can use to distinguish a reflection from the real scan line, since it is likely the neural network uses the same information.

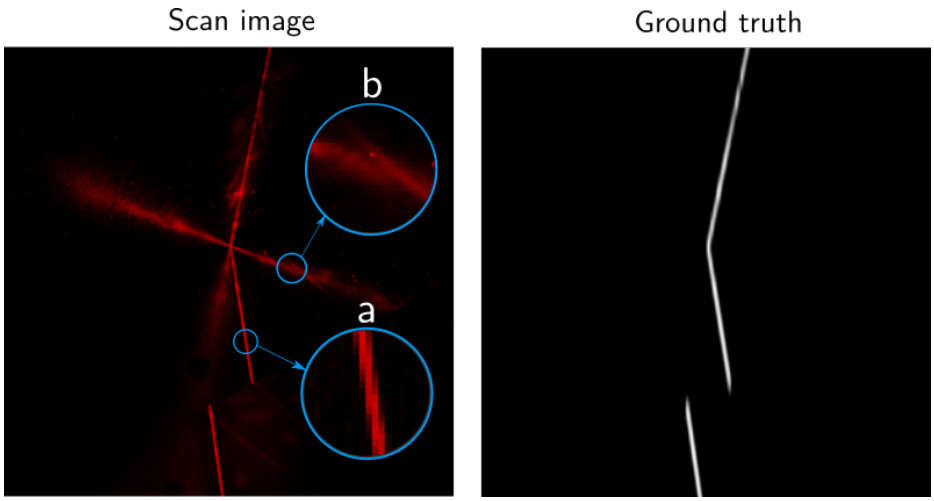


Figure 7.6.: Real scan line compared with reflected scan line

7.2.1. Scan line width and appearance

A feature that is believed to be the dominant method of the network of distinguishing reflections from the real scan line, is the overall size and appearance of the scan line against the reflections. Consider the scan image in 7.6, where a zoomed region of the actual scan line labeled a, and of a reflection b. The real scan line in a is brighter, and is consistently 2-3 pixels wide. The blurred reflection in b is wider and has less intensity. These local differences are well suited for being filtered with convolutions, as it is a function of the surrounding pixels. Upon inspection of the results, the trained U-Net models successfully detected these differences for all methods.

7.2.2. Stereo view overlap

For the methods using two cameras, an overlapped image of the left view and the projected view was used as the input image for the U-Net model. An alternative to sending in both views as the input, could have been to pass on the overlap between the views, which could have resulted in the same performance. However it was believed that the model could figure out the overlapping function itself, and potentially use information about the non-overlapping reflections to guide the search for the real scan line. From inspection of the results, the U-Net model only made predictions of the real scan line, where the views overlapped. In some cases, a diffuse to specular reflection overlapped in both views. The geometric consistency methods made some wrong predictions, but were generally able to filter these reflections, which is believed to be of the same reasoning as in 7.2.1.

For the test set with specular reflections, most reflections did not overlap in both views, and the system using a stereo camera setup proved more robust to filtering reflections than with a single camera.

7.2.3. Continuous scan lines

Color encoding the scan lines, made it possible to filter reflections where color similarity through epipolar lines did not match. The color similarity requirement was used as a pre-processing step to the machine learning process, since it is not a local feature. Another reason for the pre-processing step is that a normal convolutional neural network is translational invariant, such that encoding the positional encoded information would require some change to the network or input. The true scan line was never filtered due to the color similarity requirement, but some sections of the reflections were. The reflections were found to match colors at certain intervals, such that the lines of reflections were seldom continuous. It is believed the U-Net model was able to use this information, since the results on the epipolar consistency methods improved on the specular reflection test set when comparing methods with the same number of cameras.

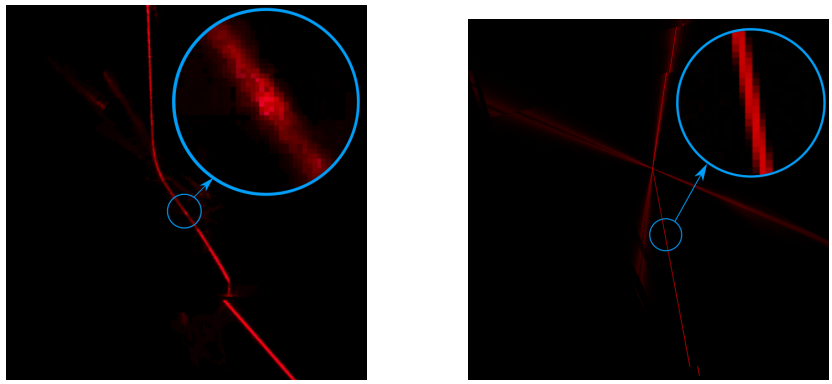
7.3. Feasibility for real implementation

This thesis explored a machine learning approach to filter reflection in simulated scan lines on reflective parts. A good result on a simulation has no real value in itself, however the results may be used as a basis for a real implementation.

7.3.1. Differences in real and simulated data

In most cases, data from a simulation is never fully equivalent to data obtained from the real world. A simulation uses data that is based on models that approximate reality, which is true for the simulation performed in this thesis. A prerequisite for the machine learning approach in this thesis, is laser scan images with associated ground truth. Since real-world data might be different from the simulated data in which the machine learning model is trained on, there is no guarantee that the model will get the same results on real data. Consider the laser scan images in 7.7, where 7.7a is an image of a real scan line, and 7.7b is an image from a simulated scan line. Using a machine learning model trained on the simulated line, on the real scan line would likely cause problems, because there are clear differences between the appearance of the lines. The real scan line may be pre-processed and made more similar to the simulation data, such as down-sampling the image in this case, but the optimal approach in machine learning is

to train the model on the same distribution of data it will be used on in inference. However acquiring ground truth data, is easier in simulation than with real data, since all information from a simulation can be found or controlled.



(a) Real scan line **(b)** Simulated scan line

Figure 7.7.: Real and simulated scan line

Reducing difference between real and simulated data

The data from the simulation may become sufficiently close to the data obtained from a real laser line, by tuning the materials, laser line and applying post-processing. However this may be a tedious process when adapting to many different materials. Another approach may be to use a generative adversarial network (GAN), such as StyleGan2 [11], as a post processing step. A GAN can take images from 2 different distributions, let's say distribution A and B, and attempts to convert images from distribution A to appear as they were in distribution B. This may help the simulated laser line to appear more like a real laser line, but will not influence the underlying properties of the reflections, such as the spread and angle of the reflections.

Transfer learning

If a small labeled dataset of real images with ground truth can be attained, it is possible to train the network on the simulated images first, then fine tune the network on the real data [18]. This approach is called transfer learning. The advantage of transfer learning is that the real dataset can be considerably smaller, since it has learnt many features of the problem, from training on the simulated dataset.

7.3.2. Real implementation

There might be small differences between simulated and real data as previously discussed, however the methods proposed in this thesis are still assumed to hold true for real scans. Reflections in real scans still have a visual difference between reflections and the true scan line, such as in 7.8. If it is feasible to gather ground truth labels for real scans, or make the simulated scans resemble the real scans, a machine learning process could be of value based on the results in this thesis. The geometric consistency method is still applicable on real scans, since it only requires an extra camera, and the method of overlapping the projected view with the camera view is the same. To the author's knowledge, there does not exist a small laser system, that can project periodically patterns of different colors. A projector might be a viable option, but a projector is larger and the required intensity of light could be insufficient. An example of a practical implementation using single color laser that utilizes epipolar consistency is shown in 7.9a. The 3 lasers will then have a projected pattern as shown in 7.9b.

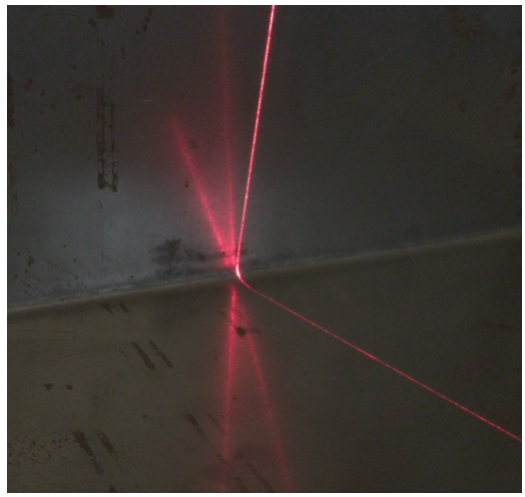


Figure 7.8.: Real scan line example

7.4. Future work

Testing the proposed machine learning workflows on a real laser scanner is an obvious next step for future work. A real implementation faces certain challenges as discussed in 7.3. As a first step, the transfer learning approach discussed in 7.3.1, with manually labeling ground truth images might be the easiest method to verify that the machine learning approach is feasible on a real implementation.

For robotic welding, the 3D scans need to be of a minimum accuracy and the

(a) Laser setup **(b)** Projected patter

Figure 7.9.: Single color laser setup using epipolar consistency

methods used to capture them must be robust for the welding to be reliable. The methods in this thesis do not fundamentally solve the problem with reflections, as there are errors in certain cases. Whether the machine learning methods meet the required accuracy and robustness with real scans is left for future work. However, an approach to add more redundancy or more incorporated information in the scans may be needed. An easy way to add redundancy to the process is to do multiple scans over the same surfaces from different angles, however this approach is more time consuming. An alternative to color encoding the epipolar lines, is implementing a time-multiplexing projected light, where certain parts of the scan line are turned on or off. The section where the scan line in on or off must be known, such that reflections can be discarded through the epipolar lines where the scan line is off. A requirement for time-multiplexing is that the projected light is not constant and is controllable, which traditional lasers are not. A post-processing approach to solve the reflections, could be to do ray tracing on the final mesh, and solve for the observed reflections during the scan.

Chapter 8.

Conclusion

In this thesis, a machine learning approach to filter reflection has been presented on multiple laser scanner systems. Simulated laser scan images with ground truth, are used to train an U-Net convolution neural network to predict the pixel-wise position of the true scan line through semantic segmentation. The methods were compared against three test datasets with different types of reflections. The three test datasets consist of materials with rough surfaces producing blurry reflections, smooth materials producing sharp specular reflections and physically based rendering materials(PBR) producing various reflections.

The first method uses a traditional laser scanner system with one laser and camera. The system is used as a baseline for the other methods, to test how incorporating additional information can benefit the machine learning process. The U-Net model was trained on images generated with the traditional laser scanner, and filtered a majority of all reflections across the test sets, while performing better at the blurry and PBR test set than the specular test set.

The second method used an additional camera compared to the standard laser scanner. With stereo cameras, it is possible to project the laser scan image from one view to the other through planar homography. Both these images were used as the input of the U-Net model, which improved the results on the specular test set, but made more false predictions on the blurry test set compared to the method with the traditional laser scanner.

The third method incorporated information through periodically color encoding the scan line. This enabled filtering reflections' color were not consistent with the epipolar lines of the projected scan line. This filtering process was used as a preprocessing step to the U-Net model, which improved the result on the specular reflection test set compared to the baseline method.

The fourth method used two cameras in addition to a color encoded scan line.

The method filtered reflections through matching colors with the epipolar lines for each camera, and one of the views was projected to the other through planar homography. Both the views were used as input to the U-Net model, which enabled the network to correctly predict all specular reflections with minor errors while making false predictions on strong blurry reflections.

Overall the machine learning approach proved effective in filtering reflections. Inspecting the predictions, it is believed that the U-Net model could distinguish the true scan line from reflections by the appearance of the line. Adding another camera or color encoding the scan line further improved the capability of U-Net to distinguish specular reflections from the true scan line. The work done in this thesis shows that machine learning is a promising component of a real laser scan system to filter reflections, however there are several challenges for a real implementation.

References

[1] *AmbientCG*. https://ambientcg.com/. Accessed: 2021-03-01.

[2] André Araujo, Wade Norris, and Jack Sim. "Computing receptive fields of convolutional neural networks". In: *Distill* 4.11 (2019), e21.

[3] David Bucciarelli, Simon Wendsche, Michael Klemm, Peter Sandbacka, Charles Nandeya Ehouman, Alessandro Castagnini, Gregor Quade, and Omid Ghotbi. *LuxCoreRender*. 2020. URL: https://luxcorerender.org/.

[4] Shan-Ben Chen. "On intelligentized welding manufacturing". In: *International Conference on Robotic Welding, Intelligence and Automation*. Springer. 2014, pp. 3–34.

[5] Blender Online Community. *Blender: a 3D modelling and rendering package*. 2020. URL: htto://www.blender.org.

[6] Olav Egeland. *Robot Vision*. NTNU, 2020.

[7] ZX Gan, H Zhang, and JJ Wang. "Behavior-based intelligent robotic technologies in industrial applications". In: *Robotic Welding, Intelligence and Automation*. Springer, 2007, pp. 1–12.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[9] Sebastian Grans and Lars Tingelstad. "Blazer: Laser Scanning Simulation using Physically Based Rendering". In: *arXiv preprint arXiv:2104.05430* (2021).

[10] Paul Kah, Manish Shrestha, Esa Hiltunen, and Jukka Martikainen. "Robotic arc welding sensors and programming in industrial applications". In: *International Journal of Mechanical and Materials Engineering* 10.1 (2015), pp. 1–16.

[11] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. "Analyzing and improving the image quality of stylegan". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8110–8119.

[12] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[14] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[15] Yi Ma, Stefano Soatto, Jana Kosecka, and S Shankar Sastry. *An invitation to 3-d vision: from images to geometric models*. Vol. 26. Springer Science & Business Media, 2012.

[16] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[17] Mirjam Mattsson. *Laser line extraction with sub-pixel accuracy for 3D measurements*. 2020.

[18] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[20] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[21] J Norberto Pires, Altino Loureiro, Tiago Godinho, Pedro Ferreira, Bruno Fernando, and Joel Morgado. "Welding robots". In: *IEEE robotics & automation magazine* 10.2 (2003), pp. 45–55.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[23] Peter Shirley and R Keith Morley. *Realistic ray tracing*. AK Peters, Ltd., 2008.

[24] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[25] M Teulieres, J Tilley, Lea Bolz, P Ludwig-Dehm, and S Wagner. "Industrial robotics, insights into the sector's future growth dynamics". In: (2019).

[26] Emanuele Trucco, Robert B Fisher, and Andrew W Fitzgibbon. *Direct calibration and data consistency in 3-D laser scanning*. Department of Artificial Intelligence, University of Edinburgh, 1994.

[27] Adam Urbanczyk, Jeremy Wright, Dave Cowden, Innovations Technology Solutions, Hasan Yavuz ÖZDERYA, Marcus Boyd, Bruno Agostini, Michael Greminger, Justin Buchanan, cactrot, huskier, Miguel Sánchez de León Peque, Peter Boin, Wink Saville, Bryan Weissinger, Ruben, nopria, Chris Osterwood, moeb, Asuki Kono, HLevering, Wes Turner, Adam Trhon, Greyson Christoforo, just-georgeb, Aaron Peterson, Artem Grunichev, Austin Gregg-Smith, Bernhard, and Derek Anderson. *CadQuery/cadquery: CadQuery 2.1*. Version 2.1. Feb. 2021. DOI: 10.5281/zenodo.4498634. URL: https://doi.org/10.5281/zenodo.4498634.

[28] Yutao Wang and Hsi-Yung Feng. "Modeling outlier formation in scanning reflective surfaces using a laser stripe scanner". In: *Measurement* 57 (2014), pp. 108–121.

[29] Kelly H Zou, Simon K Warfield, Aditya Bharatha, Clare MC Tempany, Michael R Kaus, Steven J Haker, William M Wells III, Ferenc A Jolesz, and Ron Kikinis. "Statistical validation of image segmentation quality based on a spatial overlap index1: scientific reports". In: *Academic radiology* 11.2 (2004), pp. 178–189.

Appendix A.

Blender Implementation Details

A.1. LuxCore projector implementation

The following section will show how to make a LuxCore projector. The section will show how to set the spot parameter given the camera parameters and image resolution, and calculate the camera matrix given the spot shape and image resolution. To make a projector with LuxCore in Blender, spawn a spot light in the scene and open an image to project as shown in A.1b. The *size* parameter, under *spot shape* is the only parameter related to the camera parameters. Given an image, denote the width of the image as having an resolution r_x and height r_y. An important observation, is that the *spot shape* parameter behaves differently for an image that has $r_x > r_y$ and $r_y > r_x$. For both the projected images in A.1a, the same spot size of 20 deg is used, however the images both have different widths and heights. The mathematics of calculating the relationship between the spot parameters and camera parameters must be divided into two cases, the first case for a projected image that is wider than it is tall, and the second case for an image that is taller than it is wide. The first case where $r_x > r_y$ is shown in A.2a. The focal length is denoted f, the spot shape parameter θ and the sensor width and height is s_w and s_h respectively. From the illustration, the following geometrical relationship can be made

$$\tan\frac{\theta}{2} = \frac{\frac{1}{2}s_h}{f} \tag{A.1}$$

where the sensor height can be expressed with the pixel size ρ and resolution r_y as $s_h = r_y\rho$. Rearranging the equation and solving for θ we get

$$\theta = 2\tan^{-1}(\frac{s_h}{2f}) = 2\tan^{-1}(\frac{r_y\rho}{2f}) \tag{A.2}$$

For the other case when $r_y > r_x$ the calculation is similar, starting with

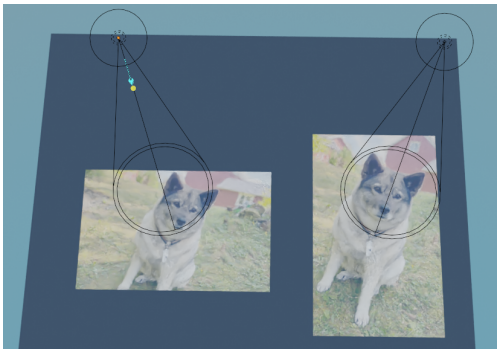
$$\tan\frac{\theta}{2} = \frac{\frac{1}{2}s_w}{f} \tag{A.3}$$

and rearranging to

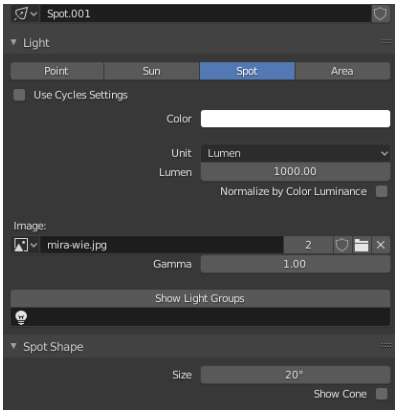
$$\theta = 2\tan^{-1}(\frac{s_w}{2f}) = 2\tan^{-1}(\frac{r_x\rho}{2f}) \tag{A.4}$$

The two cases can then be summarized as follows

$$\theta = \left\{ \begin{array}{ll} 2\tan^{-1}(\frac{r_y\rho}{2f}), & \text{for } r_x \geq r_y \\ 2\tan^{-1}(\frac{r_x\rho}{2f}), & \text{for } r_y > r_x \end{array} \right\} = 2\tan^{-1}(\frac{\min(r_x, r_y)\rho}{2f}) \tag{A.5}$$

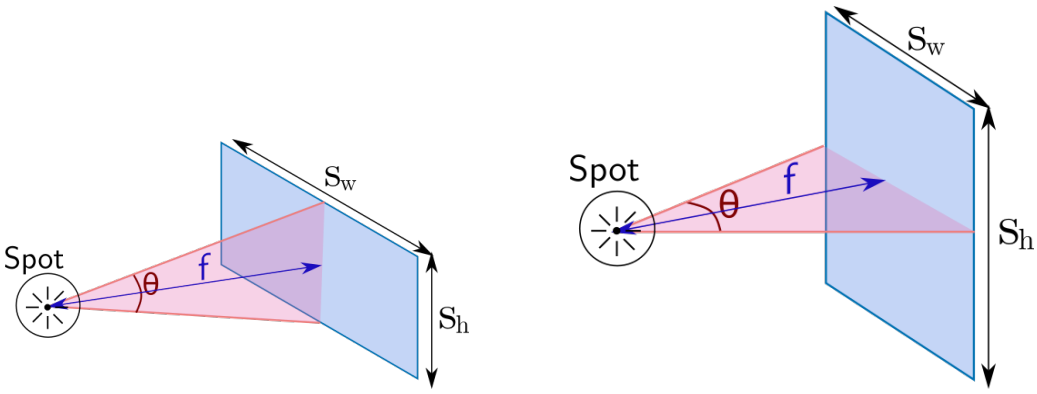


(a) Tall and wide image projected with spot light



(b) Spot light settings

Figure A.1.: Spot light with LuxCore in Blender



(a) Spot light geometry for wide image **(b)** Spot light geometry for tall image

Figure A.2.: Spot light geometry

A.2. Code

The code for the project can be found at https://github.com/olaals/masteroppgave, which links to all github repositories used in the thesis. The Blender specific repository is found at https://github.com/olaals/multivision, which is constructed as a pip (python package manager) package and used within Blender. Constructing the various laser scanning systems within Blender took up a considerable part of the allocated time during the thesis. These systems are implemented in the file *oa_luxcore.py*, and an overall schematic of the design, along with important functions for each class, is shown in A.3. An object oriented approach was found to be reasonable for the systems, since each system contained many internal states and parameters.

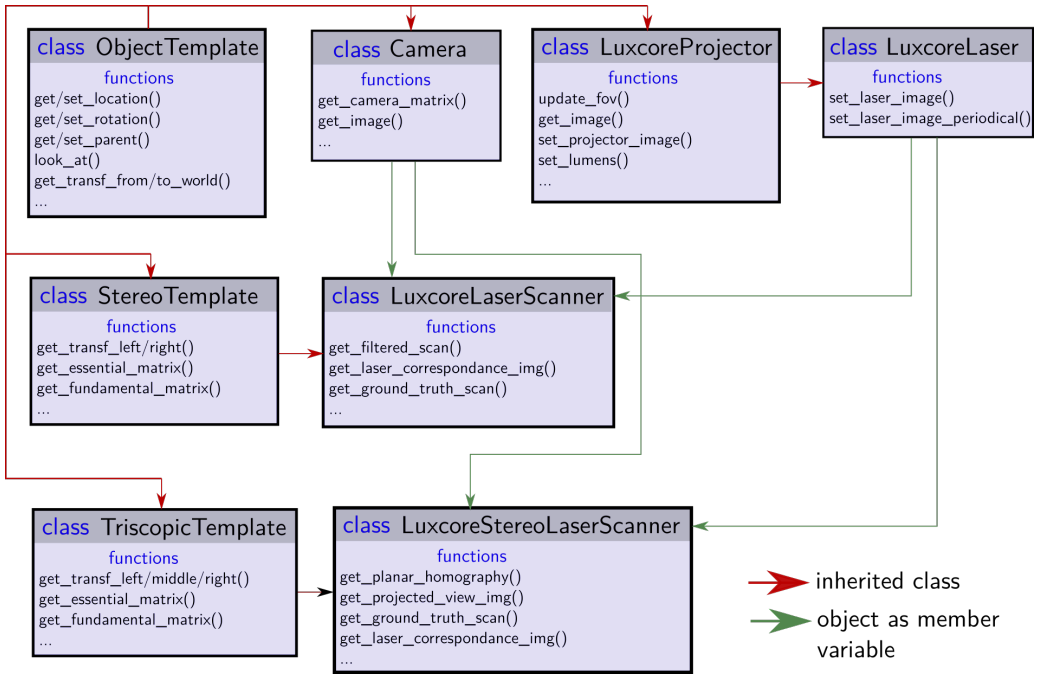


Figure A.3.: Code structure for laser scanning systems in Blender

Appendix B.

Dataset Samples and Predictions

B.1. Mesh dataset

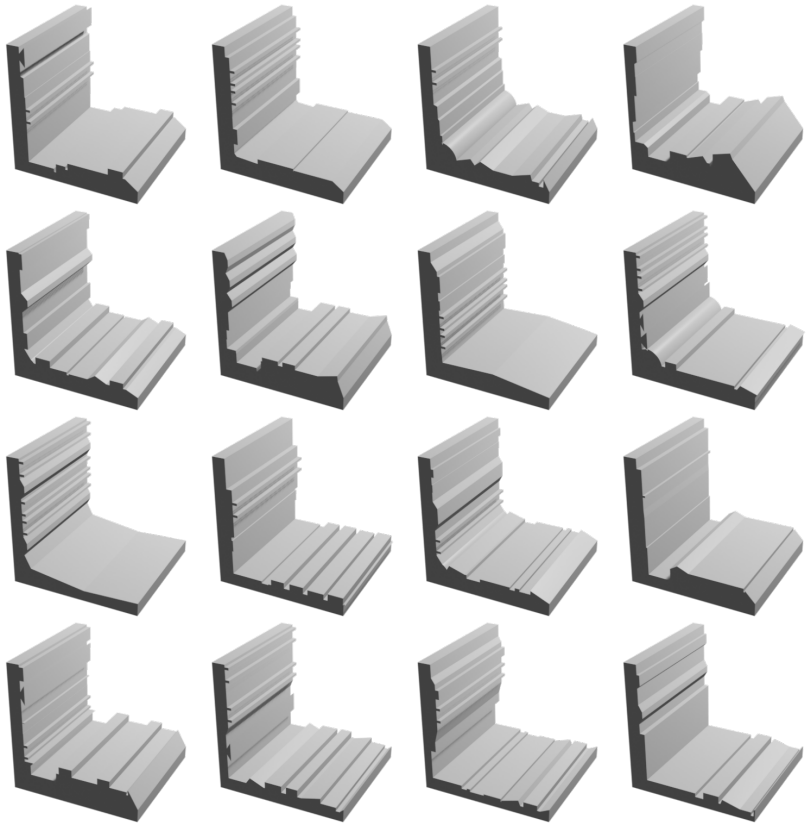


Figure B.1.: 16 examples out of the 4300 meshes generated for the mesh datasets

B.2. Additional end-to-end U-Net predictions

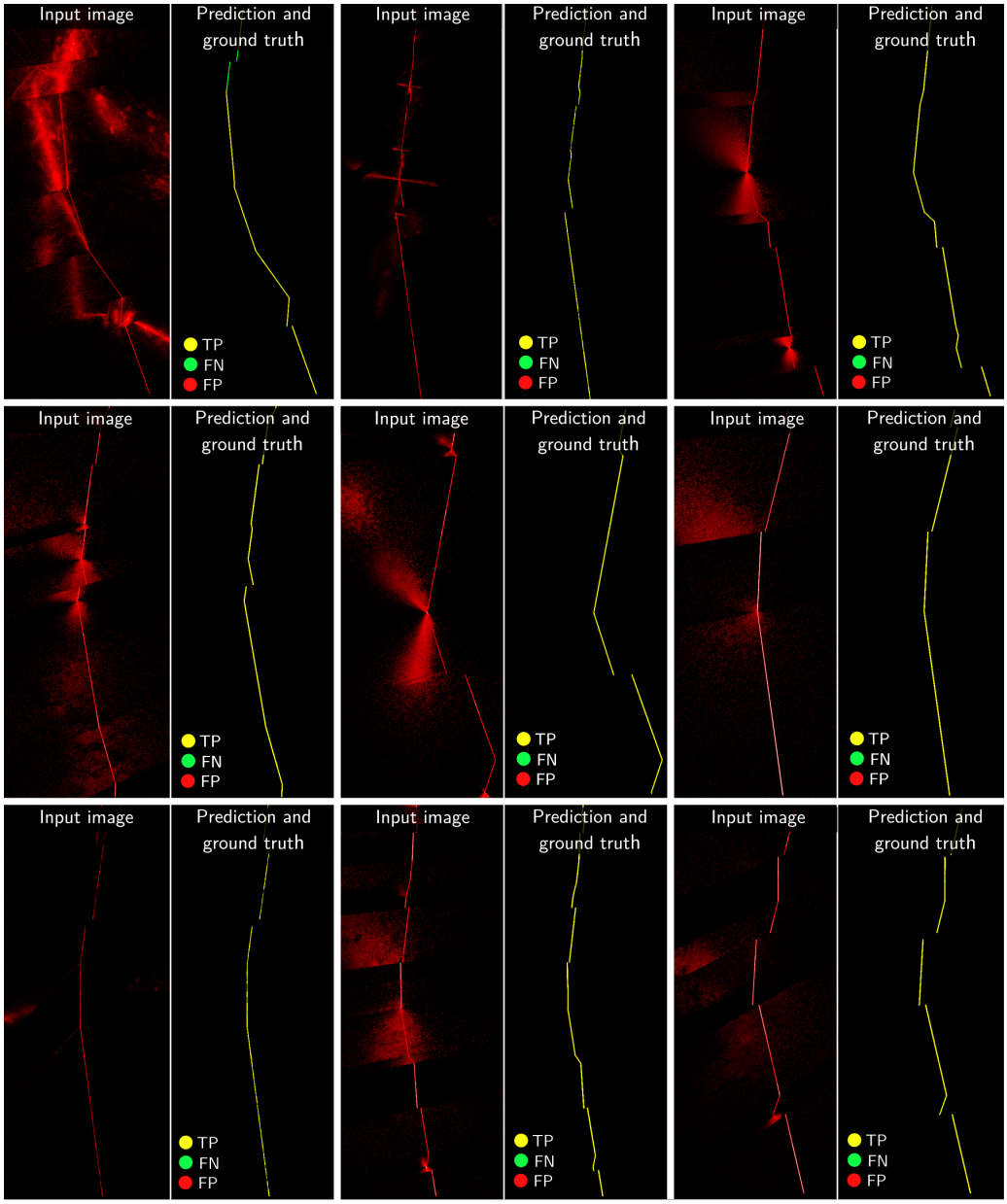


Figure B.2.: Additional PBR test set samples from end-to-end machine learning method

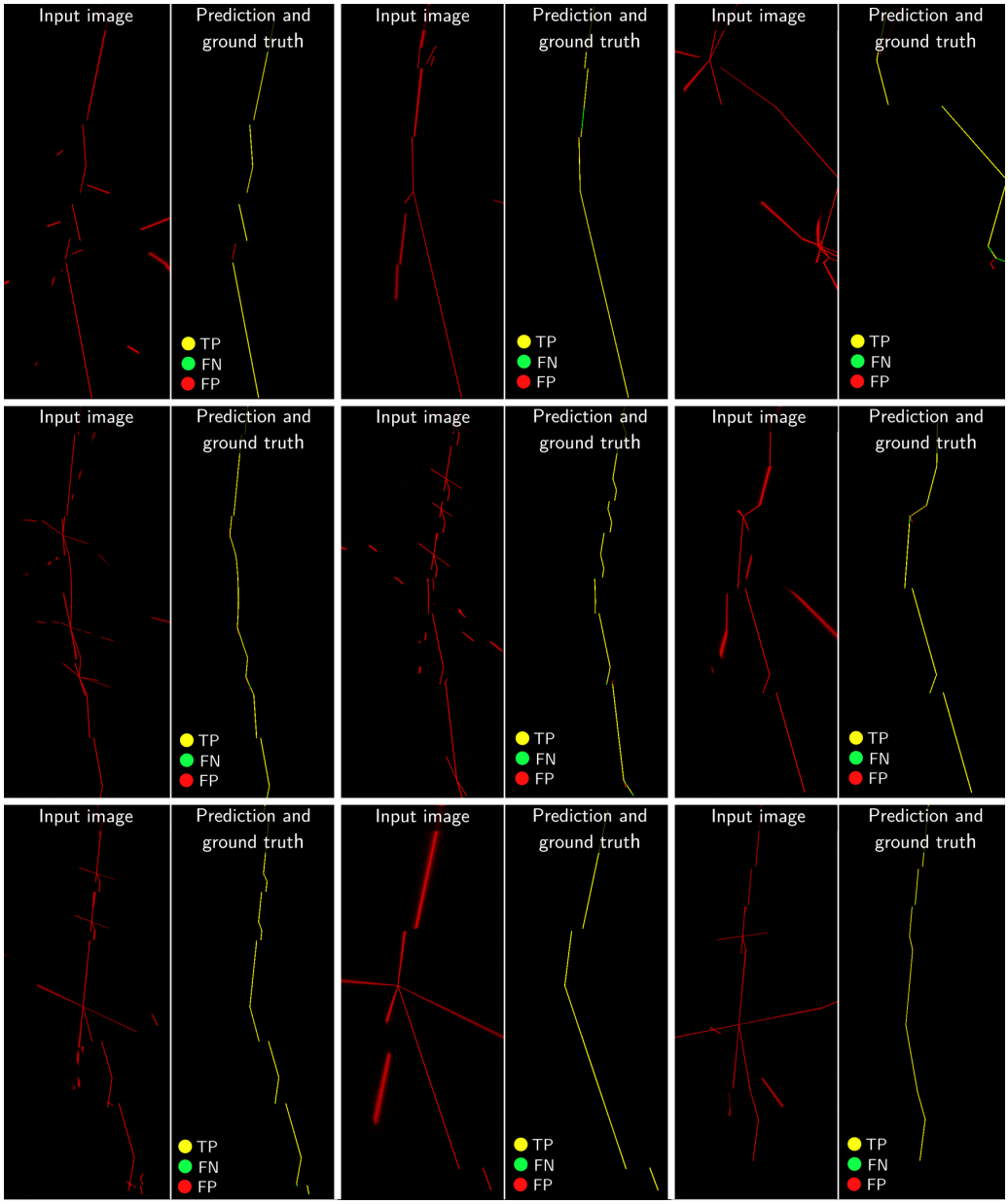


Figure B.3.: Additional specular test set samples from end-to-end machine learning method

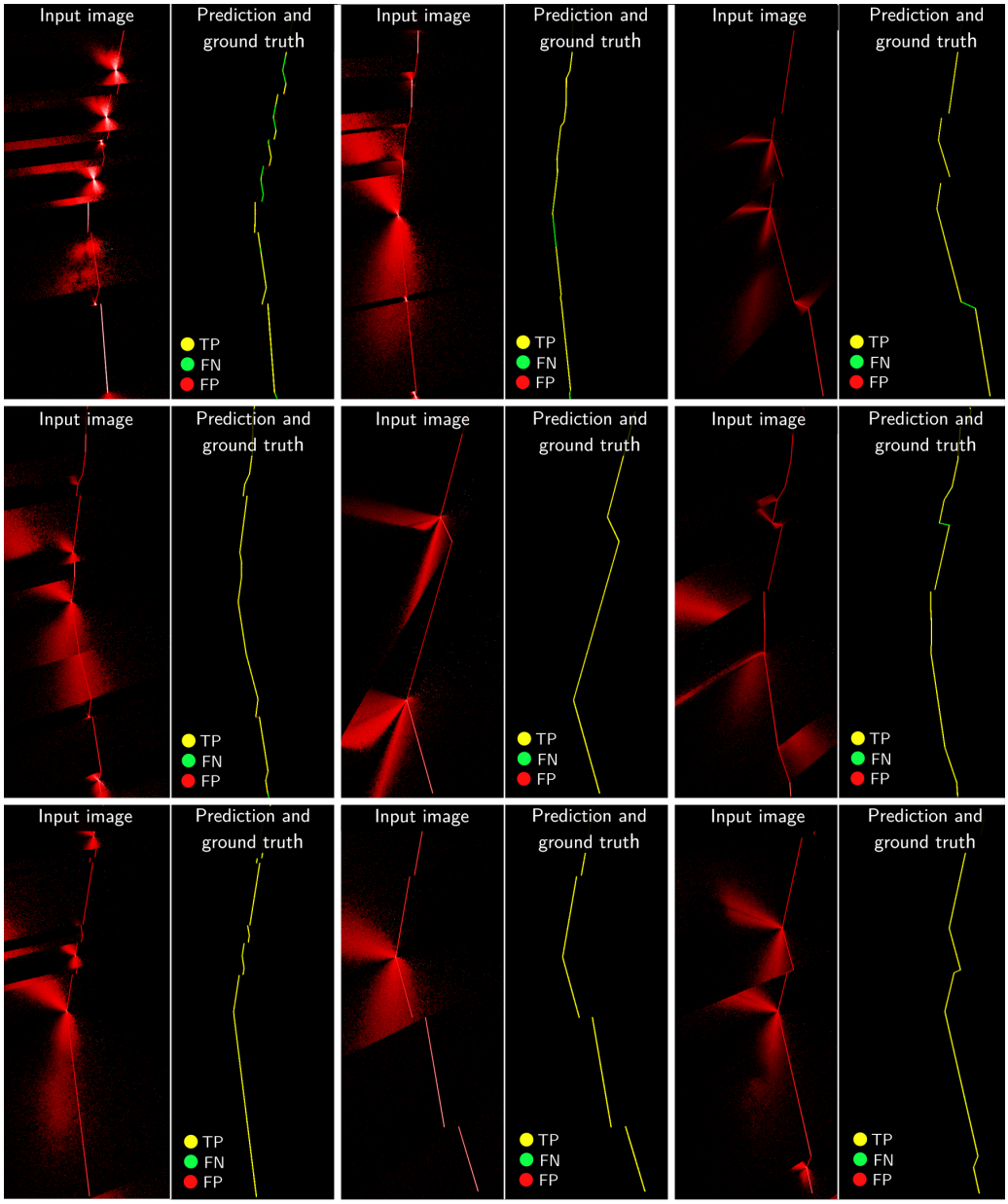


Figure B.4.: Additional blurry test set samples from end-to-end machine learning method

B.3. Additional geometric consistency U-Net predictions

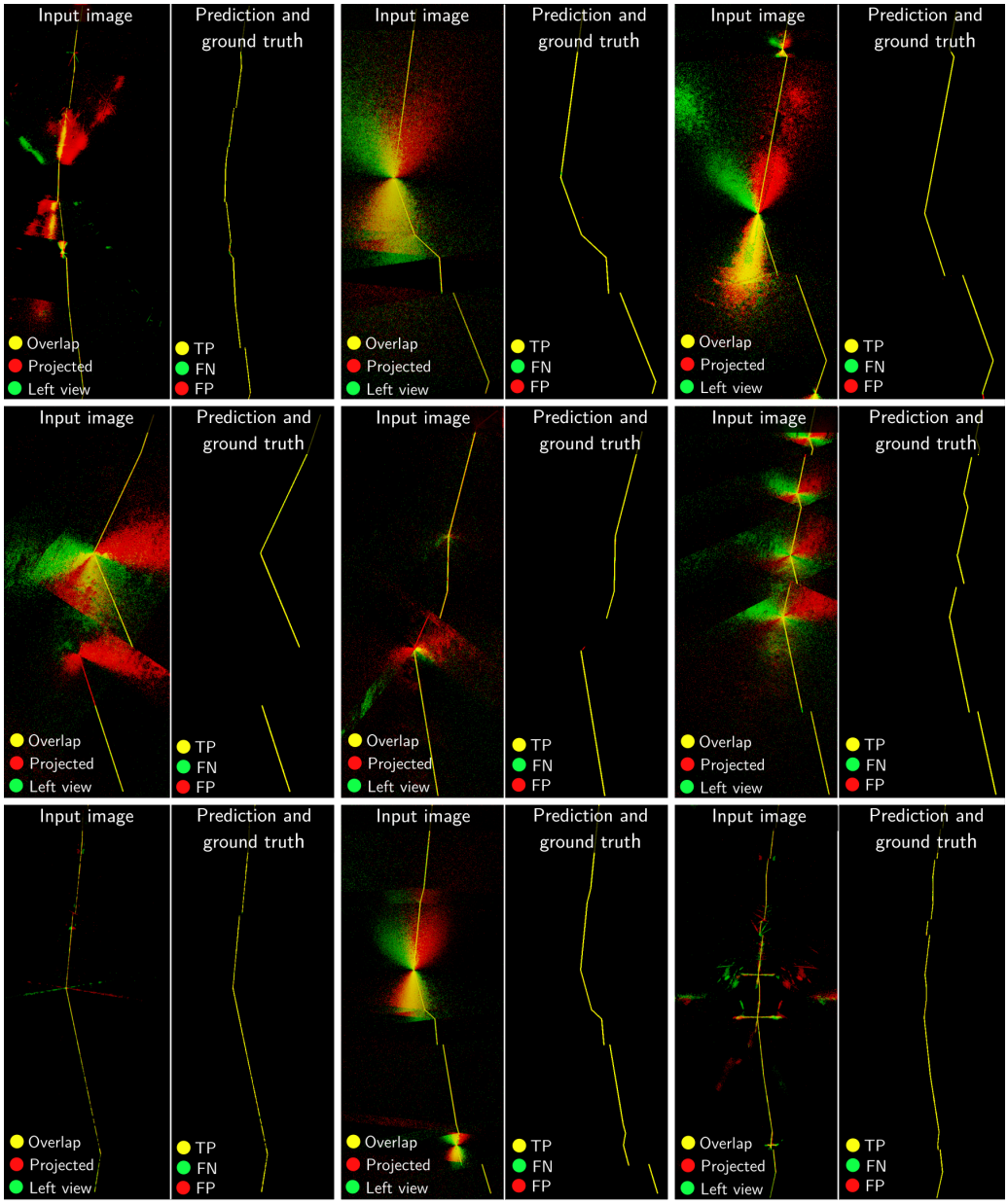


Figure B.5.: Additional PBR test set samples from geometric consistency method

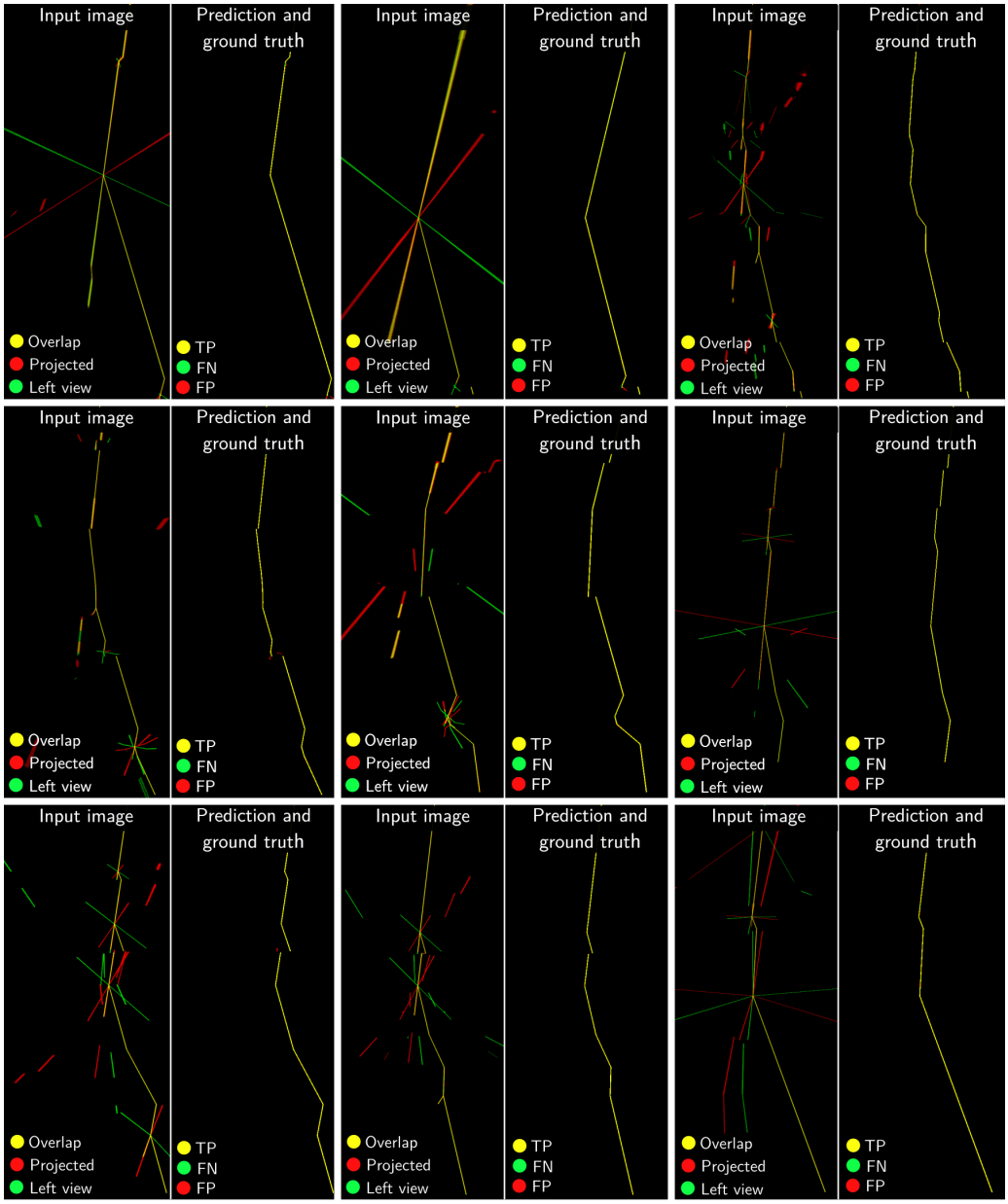


Figure B.6.: Additional specular test set samples from geometric consistency method

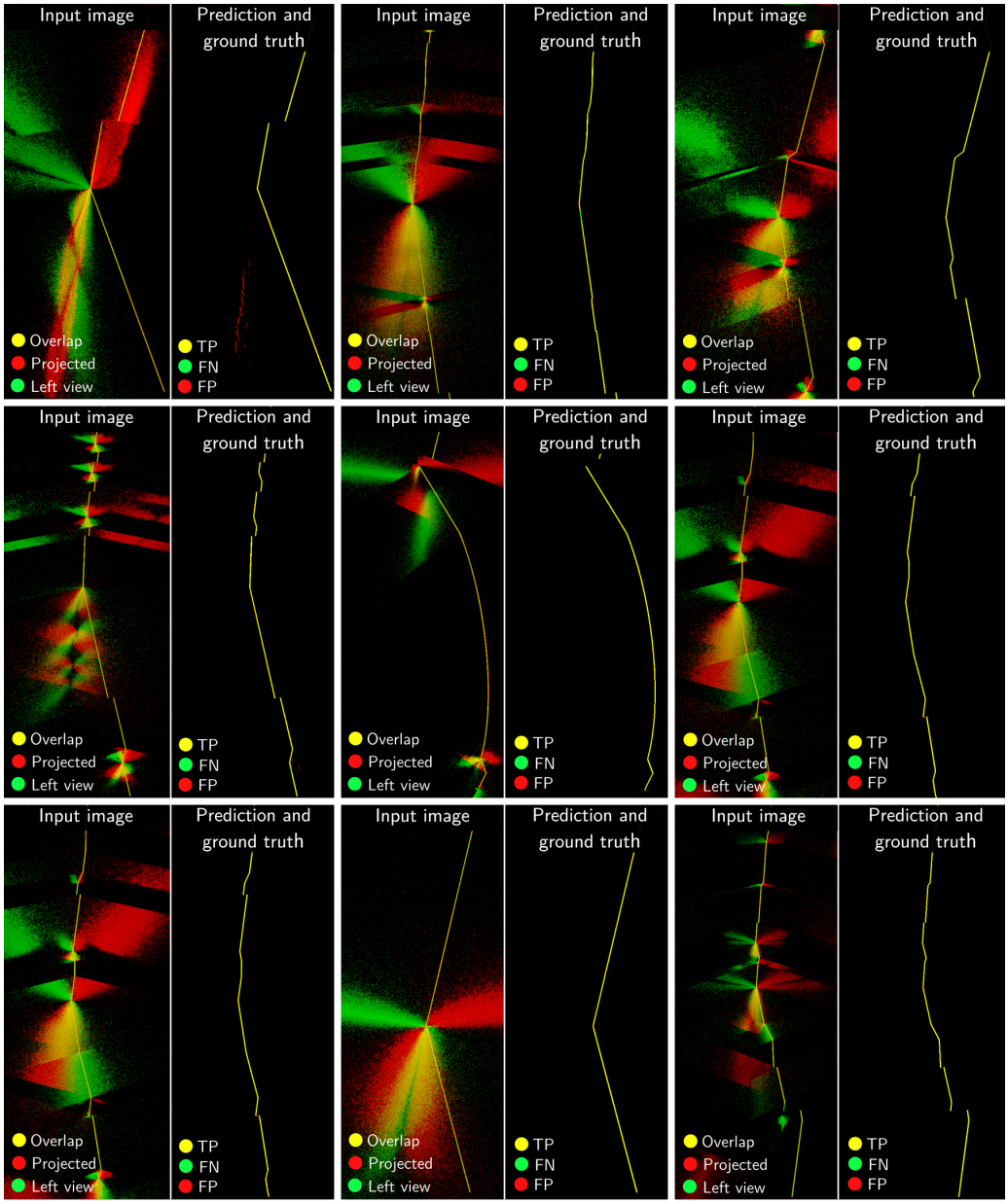


Figure B.7.: Additional blurry test set samples from geometric consistency method

B.4. Additional epipolar consistency U-Net predictions

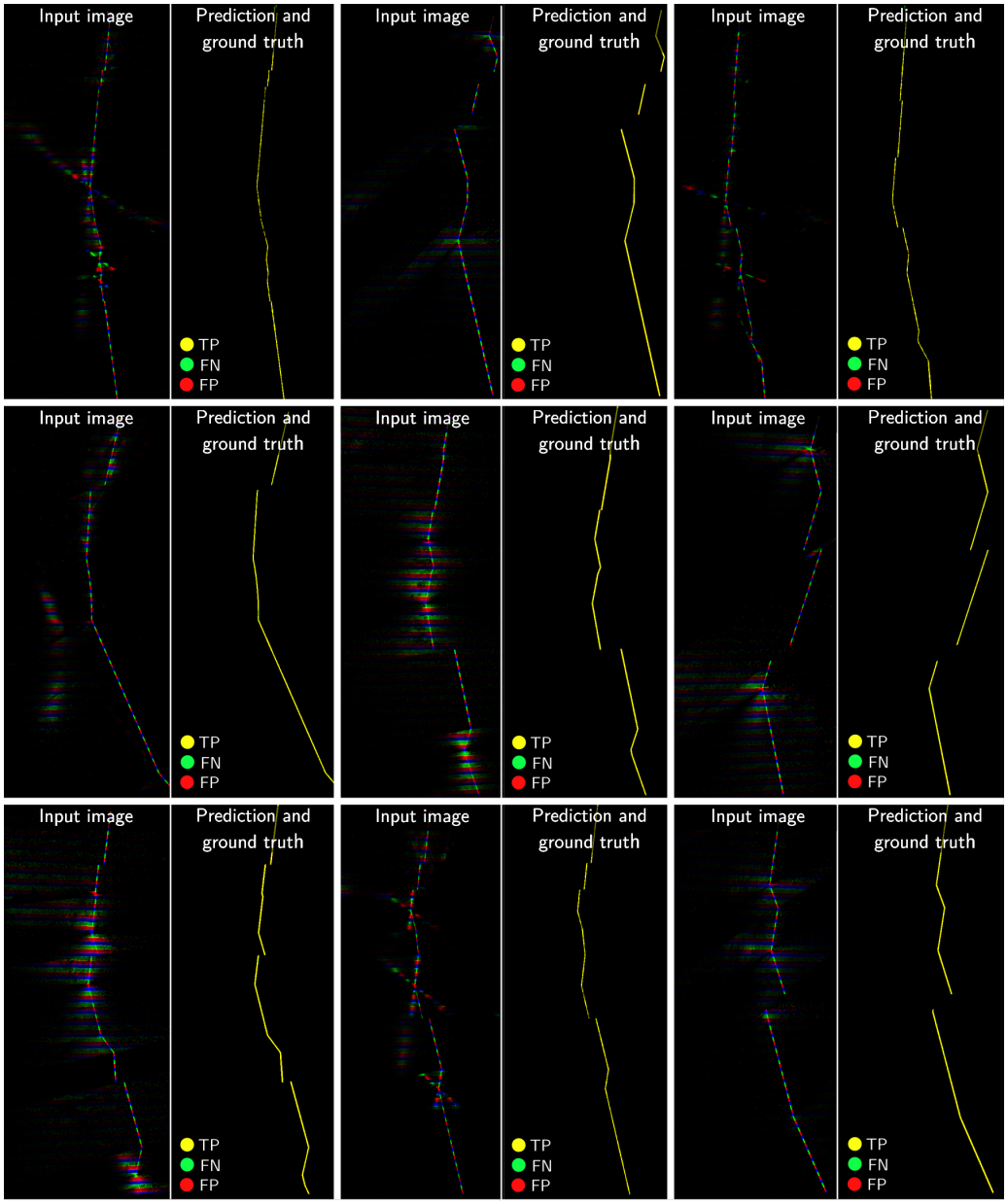


Figure B.8.: Additional PBR test set samples from epipolar consistency method

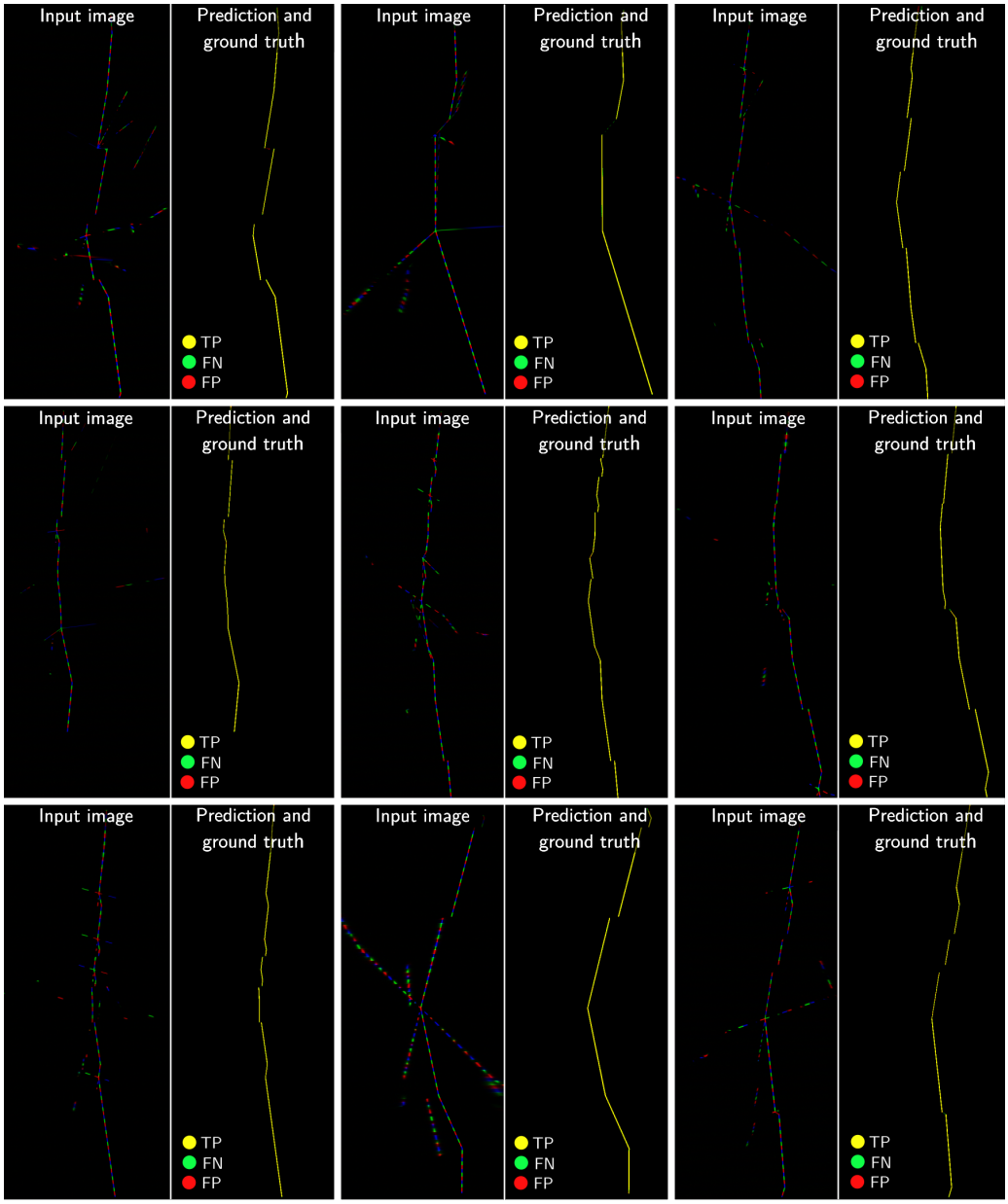


Figure B.9.: Additional specular test set samples from epipolar consistency method

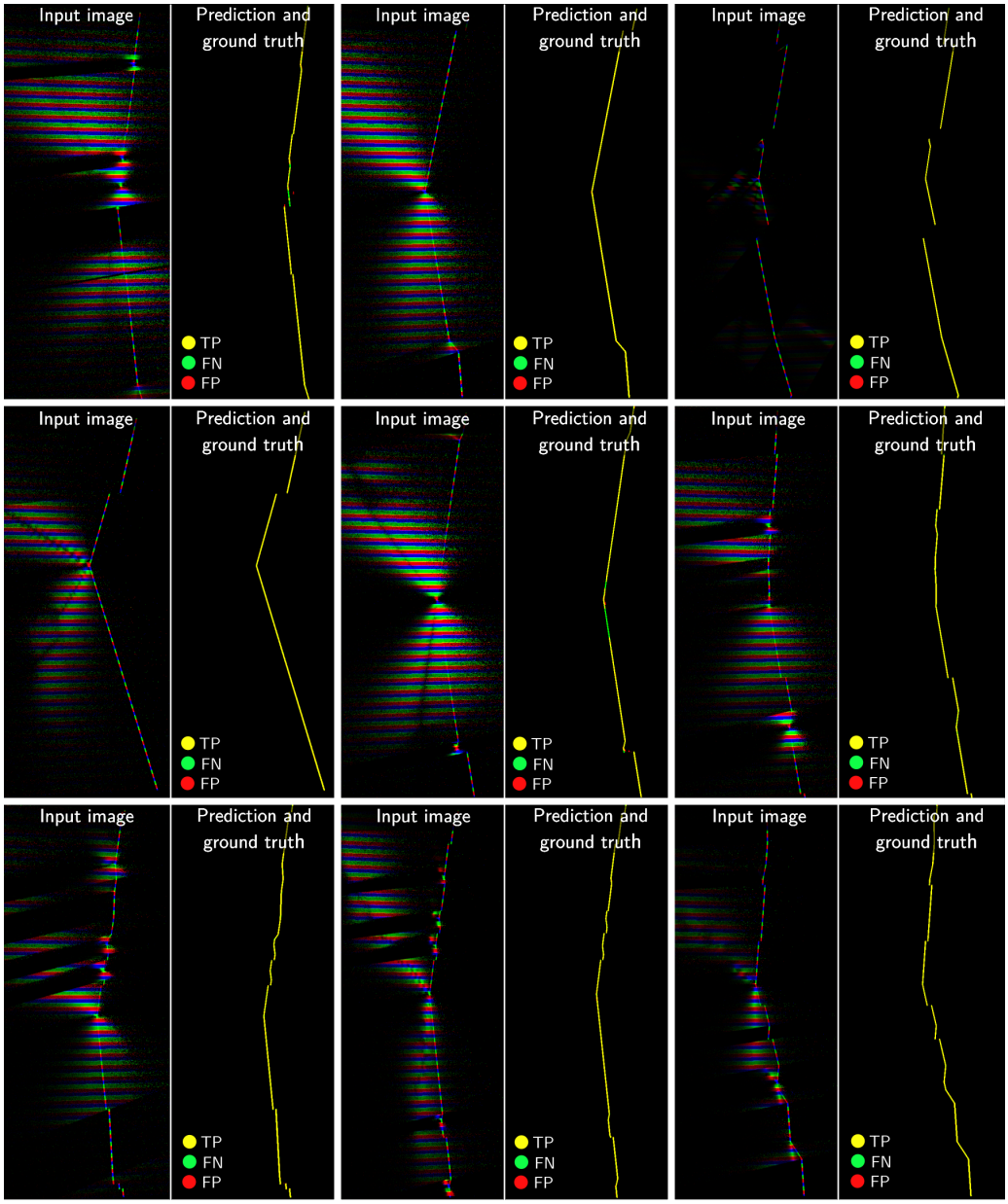


Figure B.10.: Additional blurry test set samples from epipolar consistency method

B.5. Additional geometric and epipolar consistency U-Net predictions

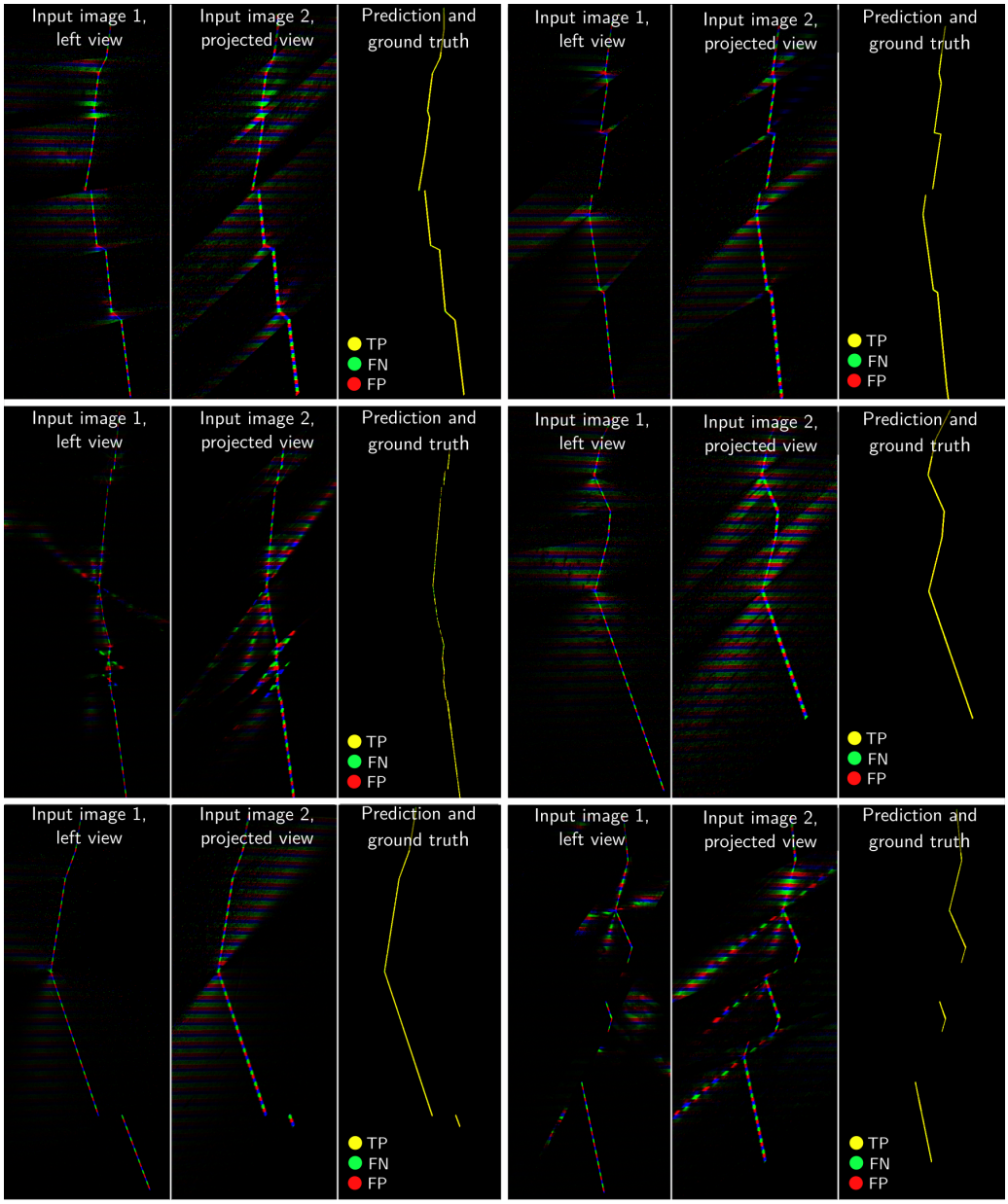


Figure B.11.: Additional PBR test set samples from geometric and epipolar consistency method

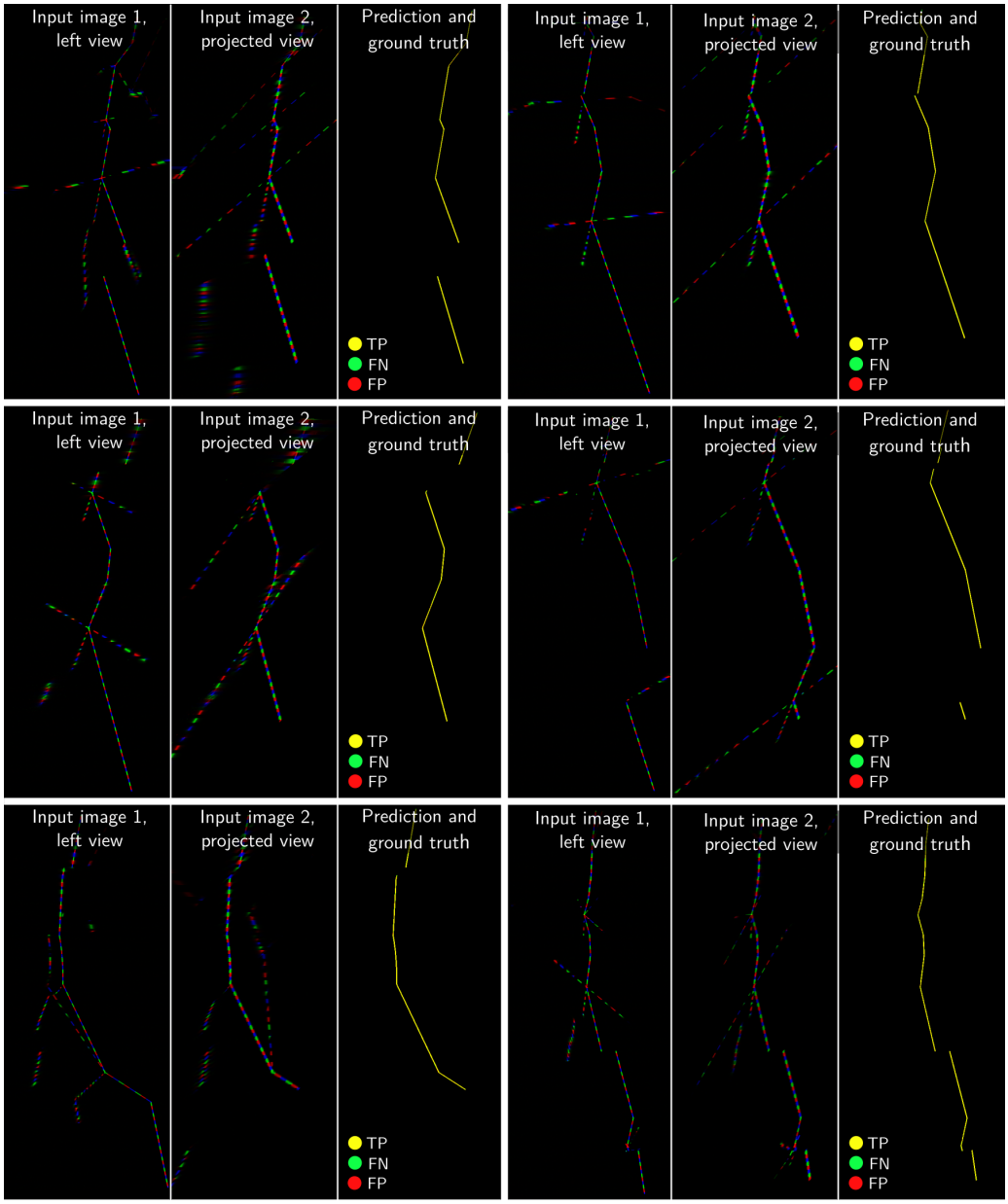


Figure B.12.: Additional specular test set samples from geometric and epipolar consistency method

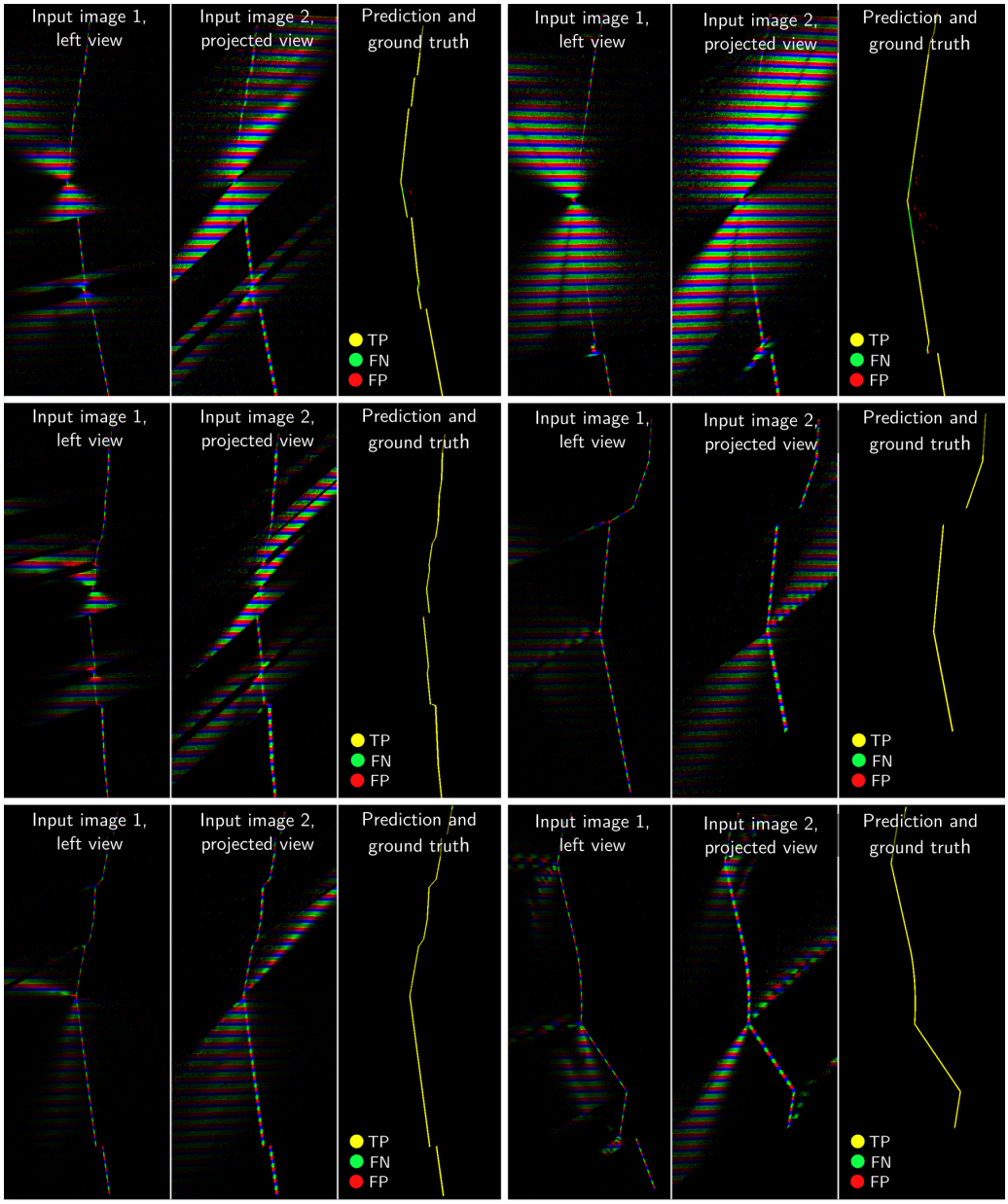


Figure B.13.: Additional blurry test set samples from geometric and epipolar consistency method

Ola Alstad

Convolutional Neural Networks for Filtering Reflections in Laser Scanner Systems

NTNU
Norwegian University of
Science and Technology