Master's thesis

2019

Storm Jaran Bruvoll Westlie

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Storm Jaran Bruvoll Westlie

A stereo image dataset of a fish model in a fresh water tank, and using it to compare stereo-matching algorithms

June 2019

NTNU
Norwegian University of
Science and Technology

A stereo image dataset of a fish model in a fresh water tank, and using it to compare stereo-matching algorithms

Cybernetics and Robotics
Submission date: June 2019
Supervisor: Annette Stahl

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

The foundation of the Master’s Thesis

Colaborating with Optoscale

Optoscale is a company located in Trondheim which delivers the product BioScope, a measurement tool tailored to estimate the biomass of fishes in a pen through a stereo camera setup, to the fish farming industry. The company wanted to investigate if their current stereo correspondence algorithm used to estimate the disparity map could be out performed by deep learning stereo matchers. This information provided the basis for the paper.

Project description

The project description was created after a meeting between the author, Storm Westlie, his supervisor, Annette Stahl, and the contact from Optoscale, Ingar Nerbø. The description consisted of three main parts.

1. Investigate the field of deep learning stereo matchers.
2. Create a stereo image dataset with accompanying disparity map ground truths.
3. Choose a subset of the deep learning stereo matchers found and use the dataset created in point 2 to compare them to Optoscale’s current algorithm.

Essential requirements

Equipment and drivers

- Nvidia GeForce GTX 1080 Ti
    - Nvidia Graphics Driver 384.130
- Intel Core i7 @ 3.7 GHz
- Optoscale’s BioScope
- Freshwater pool in a controlled environment provided by Optoscale
- Fish model provided by Optoscale

Operating systems and software

- Ubuntu 16.04 LTS
- MATLAB R2018b

Implementations, platforms and frameworks

- MATLAB's Semi-Global Block Matching method (2011)
- Luo, W. et al.'s "Efficient Deep Learning for Stereo Matching" method (2016)
- Žbontar, J., and Lecun, Y.'s "Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches" fast method (2015)
- CUDA 9.0
- cuDNN 7.0.5
- Torch 7
- OpenCV 3.4.1

Acknowledgements

Annette Stahl, Supervisor

Annette contributed from the beginning to the end of the master. Annette helped in finding a final project formulation, bringing attention to weaknesses in the master that could be improved and in general gave valuable input on the various phases of the master.

Ingar Stian Nerbø, contact at Optoscale

Ingar was the author's contact person at Optoscale. He contributed in the capture of the model fish dataset, input on how one could make a ground truth via stereo correspondence methods and provided source files of code used by Optoscale when rectifying stereo images and finding disparity maps.

Marco Leonardi, PhD student at NTNU's department of Engineering Cybernetics

Marco helped kick start the project by providing the basics of how stereo correspondence algorithms operate. Additionally, Marco provided insight into new issues that arise when attempting to stereo match on pictures captured under water and whether deep learning methods could have potential in such environments.

Abstract

In this paper, an overview of different techniques used to create ground truth disparity maps has been provided focusing on their accuracies and general requirements. The field of well-established deep learning stereo matchers in literature has been dissected by providing an overview of their accuracy results on the stereo evaluation contest KITTI 2015 and including details of their implementations focusing on the completeness of the code repository and its runtime requirements. Furthermore, a stereo image dataset of a fish model in a freshwater tank with empirical accuracy deemed to be 3 pixels has been created using MATLAB's Semi-Global Block Matching at core and optimizing the disparity range on each image pair individually by measuring the smallest disparity. The dataset was used to compare the accuracy results of two deep learning stereo-matching algorithms, MC-CNN-fst and Content-CNN, without their post-processing steps, to the accuracy results of a stereo-matching algorithm used by the Norwegian company Optoscale. Finally, the results showed that the deep learning methods managed to out-perform Optoscale's algorithm in terms of accuracy.

Sammendrag

En oversikt over ulike metoder som kan bli brukt til å lage et “riktig” disparity-kart har blitt laget, der fokuset lå på deres nøyaktighet og generelle krav for at metoden skal være gyldig. Videre har det blitt laget en oversikt over veletablerte metoder innenfor dyp læring som kan brukes til å estimere det “riktige” disparity-kartet, der fokuset lå på deres nøyaktighetsresultater som man fant gjennom stereobildekonkurransen KITTI 2015. Oversikten inkluderte også detaljer rundt hver dyp læring metodes implementasjon, der fokuset lå på fullkommenheten til implementasjonens kodearkiv og dens krav til kjøretid. I tillegg har det blitt laget et stereobildedatasett av en fiskemodell i en ferskvannstank med empirisk nøyaktighet som anses å ligge rundt 3 piksler, med en metode som bruker MATLAB’s Semi-Global Block Matching i kjernen som optimerer disparity-rekkevidden ved å måle den minste disparity-verdien for hvert enkelt stereobildepar. Datasettet ble brukt til å sammenligne nøyaktigheten til to dyp læring metoder for «stereo-matching», MC-CNN-fst og Content-CNN, uten deres etterbehandlingsteknikker, mot nøyaktigheten det norske selskapet Optoscale sin «stereo-matching»-algoritme produserte. Til slutt viste resultatene at dyp læring metodene klarte å slå Optoscale sin metode når det kommer til nøyaktighet.

Table of Contents

1 Introduction

1.1 Motivation

The monitoring of a fish population is a vital aspect when it comes to fish farming. Through knowledge of the fish population's general state of health, operators can make better decisions crucial to the well-being and growth of the fish. One piece of technology which enables the measuring of a fish population's health is underwater stereo cameras. The images such cameras provide, can be used in determining a fish' biomass through algorithm's that solve the stereo correspondence problem, and by estimating the biomasses of several fish, one gets a clear picture of whether the conditions in the fish pen are optimal or needs to be changed.

The stereo correspondence problem is that of finding the disparity between pixels in two image pairs, one from the left camera and the other from the right, which are shifted horizontally and have been captured almost simultaneous in time. Disparity is the horizontal difference in location between two pixels, one in the left image and the other in the right, which correspond to the same 3D-point on an object in the cameras' field of view. By knowing the disparity d, one can establish the depth z to the object through the formula $z = \frac{fb}{d}$, where f is the camera's focal length and b the baseline which denotes the distance between the two cameras' centers (Žbontar & LeCun, 2016, pp.1-2). From the depth, one can compute a 3D-point cloud of the object, and if the object in question were a fish, be able to estimate its biomass.

To solve the stereo correspondence problem, several stereo-matching algorithms have been proposed. (Konolige, 1997) extracted small patches centered about pixels in the left and right image and used the pixel intensities within the patches to find corresponding pixels. (Hirschmuller, 2007) transformed the image by finding each pixel's entropy and used the mutual information between pixels of the left and right image to match corresponding pixels. These algorithms were state-of-the-art at the time they were conceptualized, however, in the recent years deep learning stereo-matching algorithms have shown to out-perform them in every aspect as illustrated by the upper part of the well-renown stereo evaluation contest KITTI 2015 (Geiger et al., 2015a)'s leaderboard being dominated by deep learning stereo matchers. Although their performances are state-of-the-art, deep learning methods do come with a drawback; they require data of the disparity values in advance to be able to learn.

A company located in Trondheim, Norway which uses stereo cameras to estimate the biomass of fish is Optoscale. Optoscale wanted to know if their current stereo-matching algorithm could be out-performed by deep learning stereo matchers. Consequently, the focus of this paper is to compare their current stereo-matching algorithm's accuracy to state-of-the-art deep learning algorithms through creating a stereo image dataset of a fish model in a freshwater tank; as there are no publicly available datasets that cover the underwater environment.

1.2 Contribution

The contributions made by this report are the following.

1. An analysis which focuses on practical considerations, and accuracies of well renown methods which may be used in the creation of ground truth stereo datasets.
2. An analysis which focuses on the results and runtime performances of trustworthy state-of-the-art Deep Learning methods for stereo-matching, and an analysis of their code repositories with respect to the completeness of the repository's code and its runtime requirements.
3. A method which may be used to improve the results of a given stereo matcher by manually estimating the disparity range and visually controlling the results.
4. A stereo image dataset of a fish model in a fresh water tank with ground truth disparity maps of empirical accuracy. The images captured are of a fish model in a freshwater tank at different viewpoints, and distances.
5. A comparison of accuracy results obtained between Optoscale's stereo-matching algorithm, (Luo et al., 2016)'s Content-CNN without post-processing, and (Žbontar & LeCun, 2016)'s MC-CNN-fst without post-processing on a stereo image dataset of a fish model in a freshwater tank.
6. Insight into why it is advantageous to create stereo image datasets with corresponding ground truth disparity maps with a method of mathematical accuracy.

1.3 Outline

Chapter	Purpose
2	The theory chapter which introduces the reader to the concepts and details of the methods and techniques used to create the dataset and evaluate stereo-matching algorithms on it.
3	The method chapter which describes how the author proceeded when searching for information, creating the dataset and training the deep learning methods chosen.
4	The experiments section which outlines the hardware and software used, which metrics where chosen for the accuracy results and what kind of experiments were conducted.
5	The results chapter which shows what information was found, and the accuracies the various stereo-matching methods got on the dataset.
6	The discussion chapter which discusses the results found, focusing on strengths and weaknesses, what could be improved and why possible reasons to why some results differ from others.
7	The conclusion which highlights the most important findings of the report.
8	The future work chapter, which describes paths that may be taken forward in the light of this report.
9	The bibliography containing all the sources used.
10	The appendix which contains the table of figures and the scripts that were either produced or edited.

The reader might notice that the usual chapter “related work” is missing. The author would argue that the report as a whole gives insight into related work. Especially when it comes to the theory described on deep learning stereo matchers and overviews of them created in the results section.

2 Theory

2.1 Pinhole camera model, calibration, image rectification and depth to disparity

2.1.1 Mapping a 3D-point from the camera reference system to the digital image plane

The following explanation is based on (Hata & Savarese, n.d.). It is included in the report to better understand the camera model used to transform disparity values to 3D-coordinates, and in the process understand the values present in disparity maps. Pixel skew and distortion has been ignored to simplify the equations. The model assumes that no lens is present within the camera and that the aperture is a single point.

The pinhole camera model is a widely used mathematical model for transforming real world coordinates onto an image plane and vice versa. The model tries to replicate the inner workings of an actual camera. Before jumping into the math behind it a few aspects will be explained. Consider Figure 1 below.

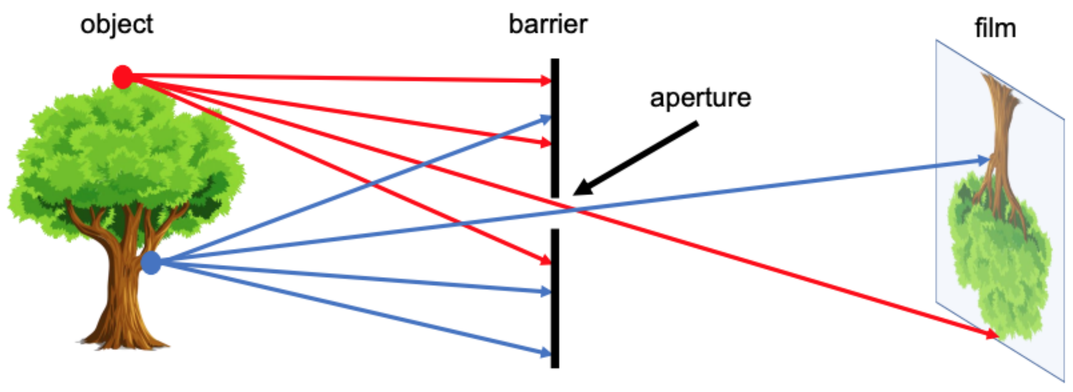


Figure 1: Flow of reflected light rays from object into camera. Source: Figure 1 in (Hata & Savarese, n.d., p.1).

Light rays reflect of the object onto the barrier of the camera. Light will only pass through the opening or aperture of the camera and all other rays are blocked from entering. Finally, light rays that do pass through the aperture are captured by the camera in its image plane (film). This is the basic concept of any color imaging camera. Furthermore, the scene will be broken down into mathematical parts. See Figure 2 below.

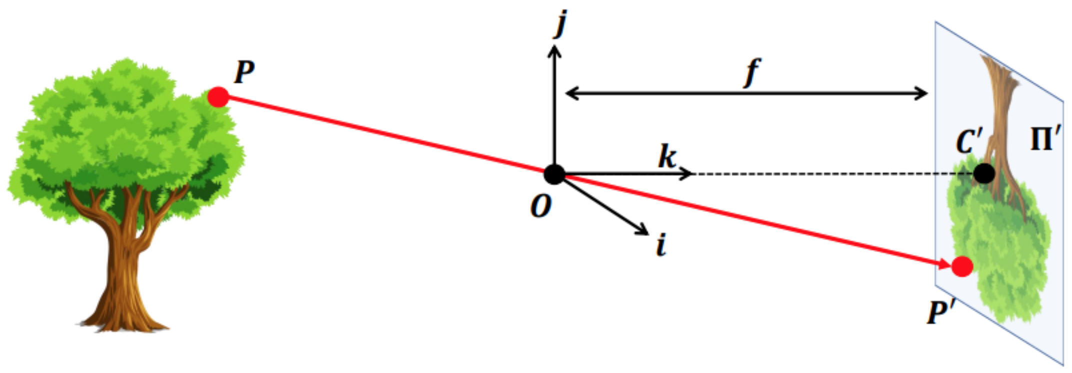


Figure 2: The main parts of a simple pinhole camera model with one reference frame. Source: Figure 2 in (Hata & Savarese, n.d., p.2).

The orthogonal coordinate system with origin O, $[i, j, k]$, is known as the camera reference system where k is perpendicular to the image plane, j points upwards relative to k and i points to the left of k*'s* positive direction.

The point $P = [x, y, z]^T$ is a vector in the camera reference system describing the 3D-coordinate of a point on a 3D object visible to the camera. $P' = [x', y']^T$, which we will denote $P'^i = \left[x'^i, y'^i\right]^T$, describes the projection of point P onto the camera's image plane $\prod'$. O denotes the position of the aperture, also known as the pinhole, of the camera and C' is its projection onto the image plane known as the camera's principal point. The line between O and C' is the camera's optical axis. f is the camera's focal length, and describes the length between the camera's pinhole and its image plane.

If we extract the z coordinate of P and insert it in a new point $P_z = [0,0, z]^T$, The triangles $P'C'O$ and POP_z are similar. Thus, after rearranging the two equations we find by the law of similar triangles the following relations between the components of P and P'^i as seen in equation 1 in (Hata & Savarese, n.d., p.3), and shown in equation 1 below with slightly different notation.

$$P'^i = [f\frac{x}{z}, f\frac{y}{z}]^T \qquad 1$$

These coordinates are described relative to the image plane reference system. We also want to see how 3D-coordinates relative to the camera reference system are mapped to the digital image plane, namely the point $P' = [x', y']^T$. After applying positive scaling factors, k and l, to x'^i and y'^i that change their unit inherited from the camera reference system to pixels and by shifting the origin of the coordinate system to the upper left corner of the digital image plane by adding the x and y coordinate of the principal point C', we get the following equation as seen in equation 4 in (Hata & Savarese, n.d., p.6), where α and β have been replaced with f_x and f_y respectively following the terminology of (Szeliski, 2010, p.52), Equation 2.57.

$$P' = [fk\frac{x}{z} + c_{x,} fl\frac{y}{z} + c_y]^T = [f_x\frac{x}{z} + c_x, f_y\frac{y}{z} + c_y]^T \qquad 2$$

2.1.2 Moving from disparity to depth and vice versa

The following theory is based on (Bolles et al., 1987, pp.9-10). It is included to shed light on the mathematics which allows one to compute the depth to a point on a 3D-object via stereo cameras and disparity. By having such knowledge, one can derive how the disparity values of a 3D-object should transition along its shape. The depth found will be relative to one of the cameras' reference system. The theory assumes identical cameras that are either parallel or their images have been stereo rectified, and inherits the assumptions of the pinhole model.

First off, we take a look at Figure 3 illustrating the most important aspects of the theory from a bird's view of the parallel camera setup. The figure is based on figure 4 in (Bolles et al., 1987, p.9). The image plane where $x_1'^i$ and $x_2'^i$ reside has been moved in front of the pinhole of the camera to simplify the observation of similar triangles. All coordinates and distances are in the camera reference system of either camera 1, cam_1, or 2, cam_2, and thus share its unit. D is the depth to the point p, b is the baseline between the cameras, f is the cameras' focal length, $x_1'^i$ is the x-coordinate of the projection of p onto the image plane of cam_1 , $x_2'^i$ is the x-coordinate of the projection of p onto the image plane of cam_2 and l_p is the horizontal distance between $cam_2$and p. The points i_{1-6} are auxillary points which will be used to define similar triangles. There are two unknown values, namely D and l_p.

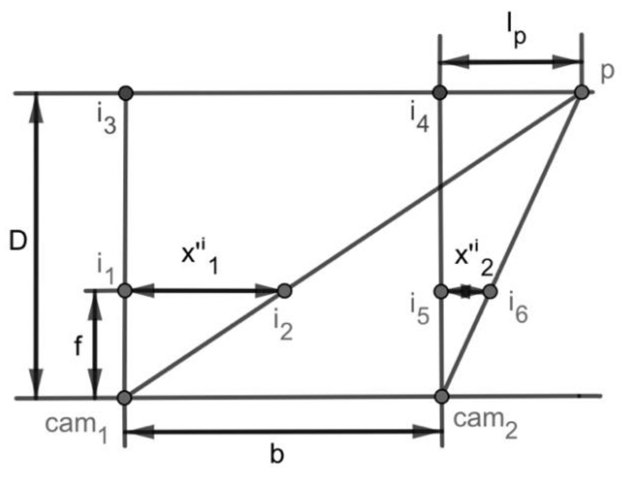


Figure 3: The geometry of a parallel stereo camera setup.

We notice that the triangle $c_1 i_1 i_2$ is similar to $c_1 i_3 p$ and that the triangle $c_2 i_5 i_6$ is similar to $c_2 i_4 p$. This observation yields the following two equations.

$$\frac{D}{f} = \frac{b + l_p}{x_1'^i} \tag{3}$$

$$\frac{l_p}{x_2'^i} = \frac{D}{f} \tag{4}$$

We define the disparity measured in the image plane as $d^i = x_1'^i - x_2'^i$ and the disparity measured in the digital image plane as $d = x_1' - x_2'$. By solving Equation 4 for l_p, inserting the result into Equation 3, and rearranging it to solve for D we find the formula which relates the disparity in the image plane to the depth.

$$D = \frac{fb}{x_1'^i - x_2'^i} = \frac{fb}{d^i} \tag{5}$$

However, Equation 5 only relates the disparity values in the image plane to the depth. We want insight into how the depth changes with regards to the disparity in the digital image plane. Under the assumption that the cameras are identical, we see through Equation 1, and the definition of d^i and d that the difference between the two is the scaling factor k from Equation 2. Finally, we insert Equation 6 into Equation 5 which yields the relation between the depth and disparity in the digital image plane.

$$\Rightarrow d^i = \frac{d}{k} \tag{6}$$

$$D = \frac{fbk}{d} = f_x \frac{b}{d} \tag{7}$$

Finally, we want the relation between the position $P = [x, y, z]^T$ of a point in the camera reference system and its disparity in the digital image plane. By rewriting Equation 2 to solve for x and y, and defining x_1' and y_1' as the x- and y-coordinate of the point in the digital image plane of cam_1, in addition to denoting c_x and c_y as the coordinates of the camera's principle point, we get the following relation between the disparity in the digital image plane and the 3D-coordinates of the point in the camera reference frame of cam_1.

$$z = f_x \frac{b}{d} \tag{8}$$

$$x = (x_1' - c_x)\frac{z}{f_x} \tag{9}$$

$$y = (y_1' - c_y)\frac{z}{f_y} \tag{10}$$

From the equation 8-10 we see that the position, especially its depth z, of a point on a 3D-object is inversely proportional to the disparity found in the digital image plane. Thus, one may conclude that the 3D-points closest to the camera have a higher disparity value than those further away. As a side note, from here on when "disparity" is mentioned, it is in reference to the disparity measured in the digital image plane.

2.1.3 The camera's intrinsic matrix and equations on matrix-vector form

The camera's intrinsic matrix is a matrix concatenating all the parameters needed to perform the steps of projecting a 3D-point in the camera reference system to the digital image plane. This section will outline how one converts the set of equations in the section above into a products of matrices and vectors.

Firstly, the coordinate systems are augmented into homogenous coordinate systems, by adding an extra dimension with value equal to 1, and from there on we may convert Equation 2 as seen in Equation 5 in (Hata & Savarese, n.d., p.7).

$$P_h' = \begin{bmatrix} f_x x + c_x z \\ f_y y + c_y z \\ z \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad 11$$

Furthermore, equation 11 may be decomposed into the following terms as seen in Equation 7 in (Hata & Savarese, n.d., p.7).

$$P_h' = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} [I \quad \mathbf{0}] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = K[I \quad \mathbf{0}]P_h \qquad 12$$

Where the matrix K is known as the intrinsic matrix, I is a 3x3 identity matrix and $\mathbf{0}$ a 3x1 vector with all entries having the value 0. We also notice that if we want to recover P' one simply divides P_h' by its third coordinate and extracts the two first.

2.1.4 Expanding the concept to 3D-points defined in the world reference system

Finally, we want to be able to map a point P^w in a 3D-world reference system to the digital image plane. We do so by defining the transformation from the world reference system to the camera system as seen in Equation 9 in (Hata & Savarese, n.d., p.8).

$$P_h = \begin{bmatrix} R & t \\ \mathbf{0} & 1 \end{bmatrix} P_h^w \qquad 13$$

where R is a 3x3 rotation matrix, t a 3x1 translation matrix, $\mathbf{0}$ a 1x3 vector with all entries set to zero. $\begin{bmatrix} R & t \\ \mathbf{0} & 1 \end{bmatrix}$ is of size 4x4 and known as the homogeneous transformation matrix. Finally, by inserting equation 13 into 12 we find the mapping from an arbitrary 3D-world reference system to the digital image plane as seen in Equation 10 in (Hata & Savarese, n.d., p.8).

$$P_h' = K[R \quad t]P_h^w = MP_h^w \qquad 14$$

where $[R \quad t]$ is the camera's extrinsic matrix of size 3x4 and M is known as the camera matrix with size 3x4. Thus, by finding M one can project any point in the 3D-world coordinate system to the digital image plane as long as it is within the cameras field of view.

2.1.5 The general idea behind calibration

Usually, the parameters that go into the camera's intrinsic and extrinsic matrices are not known. Camera calibration is the process that determines these. Several methods of doing so exist, and in this paper the focus will be on the simple method of using a calibration rig as it explains the most important parts.

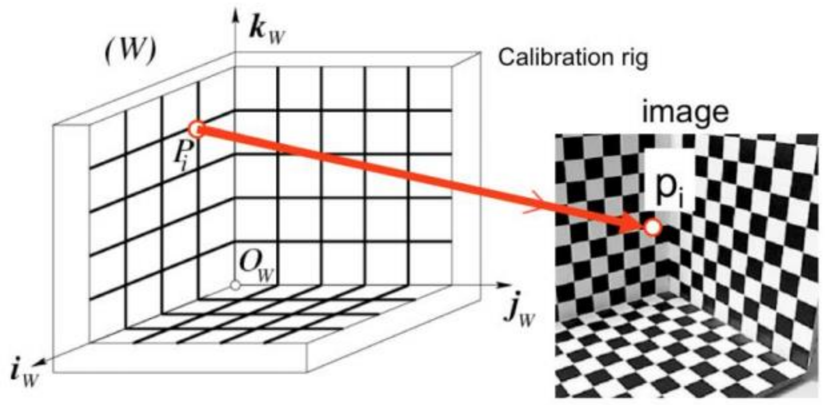


Figure 4: Calibration rig and its world reference system. Source: Figure 7 in (Hata & Savarese, n.d., p.9).

We consider Figure 4 which shows a camera calibration rig with a checkerboard pattern and the definition of the world reference system as seen. In order to calibrate, one needs to determine the locations of key points, such as the corners of the squares in the checkerboard, in the world reference system $P_1^w, P_2^w, \dots, P_n^w$, for instance by doing measurements on the rig itself. Furthermore, by detecting all corners in the digital image of the scene, one can link those points $P_1', P_2', \dots, P_n'$ in the digital image plane to their corresponding points in the world reference system.

2.1.6 Calculating the unknown variables of the camera matrix

Before stating the equation used to find the unknown variables in M, we will look at the number of unknowns in it. (Hata & Savarese, n.d., p.8) states that R has 3 degrees of freedom (DOF). Then the number of unknowns in $\mathbf{t}$ is 3, and in this paper the intrinsic matrix has 4 unknowns as we have ignored the effects of pixel skew. In total that leaves us with 10 DOF. Furthermore, we will have a look at equation 11 in (Hata & Savarese, n.d., p.9) on a slightly different form.

$$P_h' = \begin{bmatrix} x_h' \\ y_h' \\ z \end{bmatrix} = MP_h^w = \begin{bmatrix} m_1 P_h^w \\ m_2 P_h^w \\ m_3 P_h^w \end{bmatrix} \qquad 15$$

Where equation 15 simply is equation 14, where the rows of M, m_1, m_2, m_3**,** have been extracted. Next, we divide by $m_3 P_h^w$**,** as it is a scalar, and remove the last coordinate to recover our original digital image plane coordinate system as seen in equation 16, and get equation 17 by multiplying equation 16 with $m_3 P_h^w$ and move every term to the left side.

$$\Rightarrow P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \frac{m_1 P_h^w}{m_3 P_h^w} \\ \frac{m_2 P_h^w}{m_3 P_h^w} \end{bmatrix} \qquad 16$$

$$\begin{bmatrix} x' m_3 P_h^w \\ y' m_3 P_h^w \end{bmatrix} - \begin{bmatrix} m_1 P_h^w \\ m_2 P_h^w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad 17$$

We notice that Equation 17 applies two constraints and so by gathering 5 pairs of corresponding points we may determine the value of every unknown in the camera matrix. However, common practice is to gather several corresponding pairs to shield against degenerate points and solve the resulting minimization problem with a constraint as seen in Equation 12 and 13 in (Bolles et al., 1987).

2.1.7 Stereo calibration through MATLAB and a measure of calibration accuracy

MATLAB provides an easy way (Mathworks, n.d. e) to calibrate camera's in a stereo camera setup. One simply provides several images of a checkerboard, with known board and square sizes, captured with the stereo camera setup, and through it the method outputs the components of the camera matrix for each camera, and other useful matrices. MATLAB's implementation differs from the method described above in that the world coordinates of points does not have to be known in advance, and so it is highly popular.

To measure the accuracy of the calibration, one considers the reprojection error. The algorithm has found the world coordinates of the corners of the checkerboard, and so via the camera matrices found, these points are projected into image coordinates in the digital image plane. Subsequently, the projected points are compared to those detected by the app by finding the distance between them with. Additionally, (Mathworks, n.d. b), “Examine Reprojection Errors”, mentions that a projection error of less than a pixel is considered to be acceptable. Moreover, the app has several images which may be used in determining reprojection errors, and so the mean reprojection error would be a better measure of the calibration’s accuracy.

2.1.8 Stereo image rectification

As cited from the chapter “Description” of MATLAB’s documentation page on the rectification of stereo images (Mathworks, n.d. f).

> “Stereo image rectification projects images onto a common image plane in such a way that the corresponding points have the same row coordinates. This image projection makes the image appear as though the two cameras are parallel.”.

In other words, if a point in the world reference system has been projected to the digital images of two stereo camera’s with different viewpoints, the rectification makes sure that both points may be found on the same row index in their respective images. It also enables the theory from section 2.1 to be used.

Furtheremore, the rectification has positive effects for stereo matchers, in that when attempting to match two pixels of the same 3D-point that are present in the images of camera 1 and camera 2 respectively, the search space for a match is limited to a single row (Monasse et al., 2010, p.1).

2.2 Image processing techniques

2.2.1 Grayscale image thresholding and binary images

Consider a grayscale image I_{gray}. We want to set all intensities below a certain threshold in I_g to zero, and at the same time remember which values in the image had intensities above the threshold. We create a binary image I_{bin} of the same size as I_{gray} with one alteration. It can only contain two values, 0 and 1. Next, we iterate over every entry in I_{gray} and check if the intensity value is below the threshold. If it is below the threshold, we set the entry to zero and do the same in I_{bin}. If it is above or equal to the threshold, we do not change the value in the grayscale image and set the corresponding entry in the binary image to 1. In the end, we have thresholded the grayscale image and created a binary image which denotes every position in the grayscale image that had an intensity value above or equal to the threshold.

2.2.2 Gray level morphological operations

This theory assumes a grayscale image. We start off by defining the structural element, $s(i,j)$, which is a shape such as a rectangle, disk or octagon of fixed size centered at a pixel location (i,j) of image $I(i,j)$. The structuring element can be viewed as a support region in that only values within it are used in morphological operation (Mathworks, n.d. c). A morphological operation would be an image processing technique that process images based on a structural element.

Furthermore, we will have a look at the definition of gray level morphological erosion and dilation as described in (Albregtsen, 2013, p.26) with slightly different notation.

$$[I \ominus s](i,j) = \min_{(s,t)\in s} I(i-s, j-t) \tag{18}$$

$$[I \oplus s](i,j) = \max_{(s,t)\in s} I(i-s, j-t) \tag{19}$$

Equation 18 describes the gray level morphological erosion operation, where the entry (i,j) of a new image $[I \ominus b]$ is set to the smallest intensity value present within the bounds of the structuring element. Equation 19 describes the gray level morphological dilation, where the entry (i,j) of a new image $[I \oplus b]$ is set to the largest intensity value present within the bounds of the structuring element.

Next, we will look at gray level morphological opening. Gray level morphological opening is a combination of dilation and erosion as seen in (Albregtsen, 2013, p.15).

$$I \circ s = (I \ominus b) \oplus b \tag{20}$$

Consequently, morphological opening extracts the “largest of the smallest” intensity values present in the image I.

Second to last, we will look at gray level morphological closing. Morphological closing is morphological opening in reverse order as seen in (Albregtsen, 2013, p.28).

$$I \bullet s = (I \oplus b) \ominus b \tag{21}$$

Consequently, morphological closing extracts the “smallest of the largest” intensity values present in the image I.

Finally, we will look at the gray level white Top-hat filter. Its operation is defined as follows (Albregtsen, 2013, p.34).

$$I = I - \; I \circ s \tag{22}$$

The same source mentions that using the white Top-hat filter detects light objects on a dark background. In other words, the intensity of dark background pixels is reduced, and so the light objects come better to view.

2.2.3 Median filters

A median filter operates via support regions on an image. The support region is defined as a window of size mxn, where both m and n are positive odd whole numbers. The median filter, when applied to grayscale images, superimposes the window on every location (i, j) of the image. Since both its height and width is odd, there is a center pixel within the window, and the median filter sets the value of the center pixel equal to the median of the intensity values its window has captured. Say our window is of size 1x5 and is superimposed on a grayscale image of size 1x5 with center at location $(1,3)$. The intensity values within the window are $[1, 10, 17, 7, 68]$ and by sorting it from low to high we get $[1, 7, 10, 17, 68]$. Consequently, the median filter sets the intensity value at location $(1,3)$ to 10.

Furthermore, the median filter is useful in removing random noise, salt-n-pepper noise, periodic patterns (Huang et al., 1979, p.1), and outliers in the form of extremals.

2.2.4 Connected components in the context of binary images

The general outline of the connected components algorithm for binary images is shown in the chapter "Algorithms" of (Mathworks, n.d. a), and cited as follows.

"

1. Search for the next unlabeled pixel, p.
2. Use a flood-fill algorithm to label all the pixels in the connected component containing p.
3. Repeat steps 1 and 2 until all the pixels are labeled.

"

The labeling is done by specifying a connectivity. The connectivity can for instance be 8 or 4 connectivity. In the 8 connectivity, all pixels enclosing a pixel that is to be labeled is considered. In the 4 connectivity, only the next pixel in vertical and horizontal direction are considered.

The algorithm creates labeled clusters of pixels with the same intensity, and by defining that clusters containing less than a specific number of pixels should be removed, one is able to remove smaller clusters that are not of interest.

2.2.5 Sobel Operator

The Sobel operator was introduced as a way of estimating the intensity gradient at a given location (i, j) in a grayscale image, and is a discrete differentiation operator. It consists of two matrices which when superimposed on the grayscale image approximate the components, g_x and g_y, of the intensity gradient in the horizontal and vertical direction respectively. They are defined as follows as seen in (Vincent & Folorunso, 2009, p.102).

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{23}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{24}$$

After imposing the matrices on a given pixel location (i, j), we find the value of the gradient $|g|$ and its angle θ as.

$$|g| = \sqrt{g_x^2 + g_y^2} \tag{25}$$

$$\theta = \tan^{-1}\frac{g_y}{g_x} \tag{26}$$

Consequently, estimating the intensity gradient at a given pixel location (i, j).

2.2.6 Canny edge detector

The canny edge detector is a popular method used when one wants to isolate the edges of an object. The detector's method consists of four stages as mentioned in (OpenCV tutorial, n.d.), and assumes a grayscale image as input.

1. Noise reduction. If noise is present in the image, an approximation of the gradient will be affected by it, most likely giving a worse approximation. Consequently, a filter is applied to reduce the noise.
2. Gradient approximation. Edges symbolize a jump in intensity, for instance when moving from a light object to a dark one. Thus, by finding an approximation of the gradient at every pixel, one can use it to determine if a pixel represents the edge of an object. This can be done through for instance the Sobel operator, which produces an image of the gradient's magnitudes.
3. Non-maximum suppression. To find the most suited candidates to be edges, every pixel in the magnitude image is checked to see if it is a local maximum. Local means the two closest pixels along the direction of a pixel's gradient. If the pixel is not a local maximum, then it is not considered to be part of the edge.
4. Hysteresis thresholding. The final step is to define two thresholds, one high, and one low. If the magnitude of the gradient is larger than the high threshold, the pixel is considered to be an edge. If the magnitude is lower than the low threshold, the pixel is not considered to be an edge. Finally, if the magnitude is in between both thresholds, and the pixel is connected to another pixel known to be an edge, it is considered to be an edge itself.

2.3 Disparity maps and disparity range

A disparity map, $D_b(x, y)$, is the primary output of a stereo matcher and shares dimensions with the image it is in reference to, i.e. the base image, which usually is the left image of a stereo image pair. Each entry (x, y) of the disparity map contains the believed disparity between a point in the base image and the corresponding point in the match image, which usually is the right image of a stereo image pair, which when projected to the world reference system has the same coordinates.

2.3.1 Expected values in the disparity map

Owing to Equation 8 we know that the disparity is inversely proportional to the depth of an object in 3D-space. Consequently, one expects parts of an object that is close to the cameras to yield higher disparity values and vice versa.

2.3.2 Disparity range

The disparity range decides the values disparities can have in the disparity map, and consequently the search space for a stereo matcher. Consider a grayscale base, and match image I_b and I_m of the same size. We assume the images have been perfectly rectified. Point $p_1 = (x_1, y_1)$ resides in I_b, and its match $q_1 = (x_1 - d, y_1)$ in the match image, where d is the unknown integer disparity. Without having chosen the range, or interval, of the disparity, the matching algorithm has to consider every pixel in row y_1 as a possible match. However, by assuming the smallest and largest disparity possible between the two points, d_{min} and d_{max}, is known, one can restrict d to the interval $d \in \mathbb{Z} : d \in [d_{min}, d_{max}]$. Owing to the disparity range, points similar to q_1 that are outside the disparity range have been excluded from the search, thus diminishing the probability of choosing a wrong match. Additionally, the error between the true disparity and the estimated has been bounded to the range.

2.4 Dense two-frame stereo-matching algorithms

2.4.1 The general steps of a two-frame stereo-matching algorithms

We start by looking at the various aspects of a stereo matcher. Firstly, a dense stereo matcher seeks to find all possible matches and disparities between the pixels of two stereo images. Secondly, the search for a match is normally limited to the rows of the image after having stereo rectified them (Szeliski, 2010, p.538). Thus, saving computational time and limiting the search space from all pixels in an image to only one of its rows. Thirdly, the steps taken by a stereo matcher usually consists of four steps or a subset of those. Lastly, a stereo matcher can either be local, global or a mix of the two called semi-global as seen in (Scharstein & Szeliski, 2001, pp.4-5) and (Hirschmuller, 2007, p.2).

(Scharstein & Szeliski, 2001, p.3) outlines the four steps a dense two-frame stereo matcher usually makes (or a subset of those). The four steps as seen in the paper are:

1. Matching cost computation
2. Cost (support) aggregation
3. Disparity computation/optimization
4. Disparity refinement

2.4.2 Matching cost

Firstly, the matching cost, or in other words finding a measure of dissimilarity between two pixels, is described. The concept relies on finding a descriptor of a pixel and subsequently using a metric to compare how dissimilar the pixel's descriptor is to another. The most common descriptors for a pixel would be the pixel and its neighbor's intensity values or transforms of the intensities such as the Census Transform (Zabih & Woodfill, 2005) and image gradients (Klaus et al., 2006, p.2). To finalize the calculation of the matching cost and find a measure of the dissimilarity, metrics such as the squared or absolute sum of difference or the Hamming Distance could be used. The dissimilarity between position (x, y) in the base image and position

$(x - d, y)$ of the match image are stored in a 3-dimentional space known as the disparity space image $C(x, y, d)$ (Scharstein & Szeliski, 2001, p.4). Thus, the volume C contains all matching costs between pixels in the base image and the associated pixels in the match image for every disparity in the disparity range.

2.4.3 Cost aggregation

The next step is the "cost (support) aggregation". As described in (Scharstein & Szeliski, 2001, p.4) the costs in the disparity space image are aggregated by the use of support regions. An example of using support regions to aggregate the matching costs would be 2D or 3D convolving a window or block of fixed size over the disparity space image. The operations performed within the window could for instance be to set the value of the cost the window is centered on to the average of all costs within the window's reach. In essence this step exploits local information about a position in the disparity space image to aggregate the costs.

2.4.4 Disparity optimization and computation

The "disparity computation/optimization" step branches out in two directions. For a local method, the step consists of using the "winner-take-all" strategy to find the disparity of a point. For each pixel at position (x, y) the disparity associated with the lowest cost is chosen. On the other hand, a global method looks to use information about all matching costs in addition to imposing smoothness constraints on disparities and their neighboring disparities before coming to a final conclusion (Scharstein & Szeliski, 2001, p.5). Consequently, the "winner-take-all" strategy will not necessarily define the final disparity map for global methods as violations of the constraints are penalized as seen in Equation 11 in (Hirschmuller, 2007, p.3).

2.4.5 Disparity refinement

Finally, there is the "disparity refinement" step, also known as the post-processing of the disparity map. In this step, the first draft of the disparity map is available and thus it is concerned with refining it. Such refinition techniques could be to include sub-pixel accuracy and the detection of occluded areas (Scharstein & Szeliski, 2001, pp.5-6) or the removal of outliers by convolving a median filter over the disparity map (Hirschmuller, 2007, p.5).

2.4.6 Evaluation of Dense two-frame stereo-matching algorithms

Disparity maps denote the distance, or number of pixels, between two pixels in their respective two images, which have either been captured by parallel cameras or been rectified. Consequently, the standard evaluation metric is the "n-pixel" error metric. As an example, we will look at the 3-pixel error metric used by the KITTI 2015 stereo evaluation contest (Menze & Geiger, 2015, p.5). If the "true" disparity in the ground truth is 15, and the predicted 12, then the predicted is deemed correct. However, if the predicted is say 11, the distance would be larger than 3, and it would be marked as wrong. Thus, by counting the number of pixels that were marked as wrong and divide it by the number of valid disparities in the "true" disparity map, one gets a value quantifying the percental amount of wrong predictions a stereo-matching algorithm made with respect to the "true" disparity map. Furthermore, as seen on KITTI 2015's stereo evaluation webpage (Geiger et al., 2015a), accuracies are given in different categories such as "foreground", "background", "occlusion", and "all" depending on which category one is interested in.

2.5 MATLAB's Semi-Global Block Matching

Semi-Global Block Matching (SGBM) method is available in MATLAB (Mathworks, n.d. d), and uses OpenCV's "StereoSGBM" class (Opencv StereoSGBM, n.d.) which may be seen by typing 'open disparity' in MATLAB's console, and follow the implemenation's code. The theory is included in the report as it is at the core of Optoscale's stereo-matching algorithm. The method will be described in terms of a subset of the four steps which give the general outline of a dense two-frame stereo matcher.

2.5.1 The matching cost used

Before finding the matching cost, the grayscale image intensities are transformed by estimating their gradients in the x-direction through the first Sobel derivative and clipping them to a predefined range (Kaehler & Bradski, 2017), i.e. the result are bounded to the range [-clipping value, clipping value]. Consequently, the matching cost is based on image gradients which can be robust against differences in camera gain, and bias (Klaus et al., 2006, p.2). Furthermore, the matching cost is found by computing the sum of absolute difference between two support regions in the base image and matching image respectively, in combination with the sampling insensitive Birchfield-Tomasi subpixel metric (Birchfield & Carlo, 1999). The final result is the disparity space image $C(x, y, d\ ) = C(p, d\ )$.

2.5.2 Disparity optimization by a semi-global energy function

In (Hirschmuller, 2007) the next step falls under what is called the “cost aggregation” category, however following the definitions introduced in the subchapter “Dense two-frame stereo matchers” of this paper it lands in the “Disparity computation/optimization” category. Before jumping into the equations, a few variables are defined. r is a small integer directional step, for instance $[-1,0]$. P_1 and P_2 are scalar parameters penalizing changes in disparity when moving from one disparity estimate at a location to a neighboring location found by moving from p to $p - r$. P_1 penalizes a change of 1 in disparity, and P_2 larger changes. $C(p, d\ )$ is the initial disparity space image. Thus, we can define the semi-global energy function, semi-global because it does not include all paths leading to $C(p, d^d\ )$, as the following recursive function as seen in equation 13 in (Hirschmuller, 2007, p.4).

$$\begin{gathered} L_{\boldsymbol{r}}(p, d\ ) = \ C(p, d\ ) + \min(L_{\boldsymbol{r}}(p - r, d), \\ L_{\boldsymbol{r}}(p - r, d - 1) + P_1, \\ L_{\boldsymbol{r}}(p - r, d + 1) + P_1, \\ \min_i L_{\boldsymbol{r}}(p - r, i) + P_2) - \min_k L_{\boldsymbol{r}}(p - r, k) \end{gathered} \tag{27}$$

The algorithm does the following. $L_{\boldsymbol{r}}(p, d\ )$ is set to the sum of the initial cost plus the smallest path cost associated with a step in direction r from p, and subtracts “the minimum path cost of the previous pixel from the whole term” (Hirschmuller, 2007, p.4) . The final term is included to bound the maximum value a given point in the semi-global energy function can have without it growing too large. Consequently, discontinuities in the disparity map are penalized, however since the penalty is constant, natural discontinuities in the disparity map may still be preserved given the choice of the penalty parameters (Hirschmuller, 2007, p.3).

2.5.3 Disparity calculation by finding the match with the smallest cost

The final step in finding the disparity map is to join the costs of the paths in a final disparity space image $S(p, d)$ as seen in, and create the disparity map D_b associated with the base image by choosing the disparity corresponding with the minimum cost at each location p as seen in equation 14 in (Hirschmuller, 2007, p.4).

$$S(p, d) = \sum_{\boldsymbol{r}} L_{\boldsymbol{r}}(p, d) \tag{28}$$

$$D_b(p) = \min_d S(p, d) \tag{29}$$

2.5.4 SGBM’s three steps of disparity refinition

Three methods of “disparity refinement” are included in the method. Firstly, a contrast threshold is introduced to define the boundaries of the first derivative of the Sobel operator. The interaction between the threshold and clipping values is defined in the source code of MATLAB’s “disparity” function (Mathworks, n.d. d) at line 168 which one may read by entering “open disparity” in MATLAB’s console. Via the code, one reads that the clipping value which defines the range of the first derivative is equal to $63 * ContrastThreshold$.

Secondly, a measure of a match's uniqueness is used to determine the reliability of the match. The procedure is paraphrased from the description of "UniquenessThreshold" in (Mathworks, n.d. d). Let K be the best disparity estimate, and V be the corresponding SAD (*sum of absolute difference) value*. Consider V as the smallest SAD over the whole disparity range, and v as the smallest SAD value over the disparity range excluding $K-1$, K, and $K+1$. If $v < V(1 + 0.01 * UniquenessThreshold)$, then the function marks the disparity for the pixel as unreliable. Consequently, by setting the uniqueness threshold high, more emphasis is put on the SAD calculations alone. Furthermore, when looking at the interaction between the contrast and uniqueness thresholds, a smaller "contrast threshold" value most likely leads to fewer pixels being marked as unreliable. The cause is most likely the range of the values coming from the first derivative of the Sobel operator being truncated by a clip value which is proportional to the "contrast threshold", leading to SAD values being more equal given enough first derivatives having initial values that exceed the range.

Lastly, the final method of the "disparity refinement" step checks if the disparity map found, when using for instance the left image as base, is in alignment with the disparity map found when using the right image as base. Firstly, one finds the best disparity estimate from a point p_1 in the base image which leads to the corresponding point p_2 in the match image. Secondly one turns the table and finds the best disparity estimate from the same point p_2 in the match image which leads to the corresponding point p_3 in the base image. If the distance between p_1 and p_3 differ more than "DistanceThreshold", the pixel is marked as unreliable as seen in the description of "DistanceTreshold" in (Mathworks, n.d. d). The search for a match is performed on rectified images, and so the distance would be the absolute value of the difference between both points' x-coordinates as they share the same y-coordinate. (Banks & Corke, 2001, p.4) mentions that the technique's primary function is to detect occlusions.

2.6 Convolutional Neural Networks

2.6.1 Neural network terminology

Inference

When running inference with a CNN, one simply asks the network to make its prediction by having it processing the input given.

Hyperparameter

A hyperparameter in the context of neural networks are parameters set before training or inferring that control the structure of the network, for instance how many convolutional layers it should contain, what the filter size should be, and in general variables which determine how the network trains.

Epoch

An epoch is how many times the network has processed a subset, or the set, of training examples. For instance, an epoch could be defined as 50 iterations of mini-batches or having processed all available training data.

Ground truth

The ground truth contains the values the neural network tries to predict. If we've got two points of the same object in 3D-space in separate images, and we know the disparity between them are 100, that would be the ground truth.

2.6.2 The concept of Convolutional Neural Networks

A convolutional neural network is a specialized neural network meant to operate on images. In essence, a convolutional neural network is a sequence of layers consisting of learnable weights and biases that "transforms one volume of activations to another through a differentiable function". The citation is from the chapter "Layers used to build ConvNets" in (Stanford University, n.d. a) .

The cascading structure of a convolutional neural network consists of convolutional layers taking a volume as input and outputting a new, transformed, volume. Each individual convolutional layer can be looked at as a module taking an input volume, transforming the input volume via convolutional filters, methods such as normalization techniques and activation functions until finally outputting an output volume. Together, the convolutional filters and activation functions mimic the function of neurons in the brain (Stanford University, n.d. a). To understand the general flow from input to output in a layer we will have a look at Figure 5.

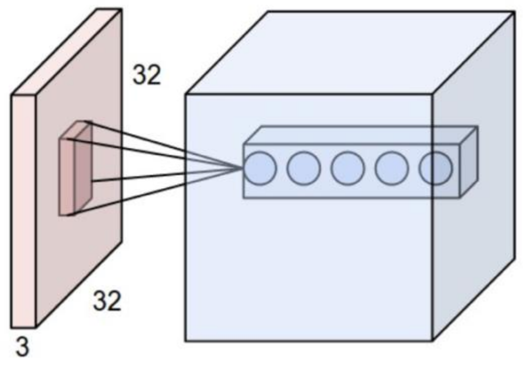


Figure 5: Illustration of the input volume (red) and the output volume the filters create (blue). Source: Example 2 from (Stanford University, n.d. a).

The 32x32x3 input volume in light pink would be a color image of 3 channels, and the dark pink area in it is the subset of the input volume the filters are currently considering. Finally, the blue box is the output volume of the convolutional layer. Each circle represents a filter's result when operating on a particular subset of the input volume. In this example, there are thus five convolutional filters. Furthermore, in a deep convolutional neural network several convolutional layers are added to a cascade, and the layers that are not the first one in the cascade take the output volume from the previous layer as input. Given certain hyperparameters which will be discussed later on, one can control the size of all output volumes (Stanford University, n.d. a).

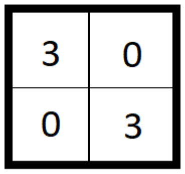
2.6.3 Rectified Linear Unit activation function

Rectified Linear Units (ReLU) is commonly used as the activation function in between convolutional layers due to it accelerating the process of training when using the stochastic gradient descent scheme compared to other non-linear activation functions, and its formula is as seen in the description of "ReLU" in (Stanford University, n.d. c).

$$f(x) = max(0, x) \tag{30}$$

2.6.4 Convolutional filters

We start off by describing theory on filters in the context of convolutional neural networks, it is based on (Stanford University, n.d. a) and (Westlie, 2018, pp.17-19). One filter is in general a 3-dimensional array which size depends on its spatial extent F and the depth D of its input. Consequently, the full size of the filter becomes FxFxD, and the area it covers, not the volume, its receptive field. Additionally, each element of the array is known as a weight, and every filter has a bias connected to it. The values of both its weights and biases are learned when training the convolutional neural network. In Figure 6 below, the value of F is 2, and D is 1, and the filter may for instance encode a diagonal feature.



3	0
0	3

Figure 6: A simple 2x2x1 filter without its bias. Source: Figure 3 in (Westlie, 2018, p.17).

Moreover, a filter has two hyperparameters associated with it, namely its size which depends on F and its stride. To understand the concept of stride, we will have a look at a simple example which is based on the example "Convolution Demo" in (Stanford University, n.d. a). Consider a convolutional layer that takes a volume of size 4x4x1 as input and has a single filter of size 2x2 with bias equal to 0, and uses ReLU as the activation function. Firstly, the filter is superimposed on the input volume in for instance the top left corner, and then the dot product is made. Figure 7 depicts the operations made by the filter on a 4x4x1 input volume that encodes a diagonal line.

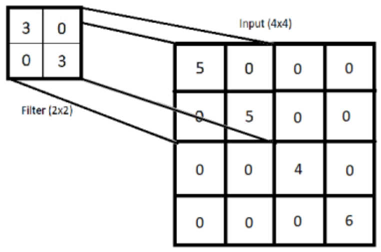


Figure 7: A filter superimposed on the input volume. Source: Figure 4 from (Westlie, 2018, p.18).

The result of the dot product is 30, we add the bias, which is 0, and pass the value through the activation function. Thus, 30 is the final output of the filter at this particular location. Next, the filter needs to slide over other values of the input. This is usually done by sliding the receptive field of the filter a number of places along the row dimension, which is known as its stride (S). "… when the filter reaches the end of the row, it moves down S rows and starts from left to right yet again." (Westlie, 2018, p.18). Figure 8 illustrates the process with a stride of 2, and Figure 9 the resulting output volume of the convolutional filter.

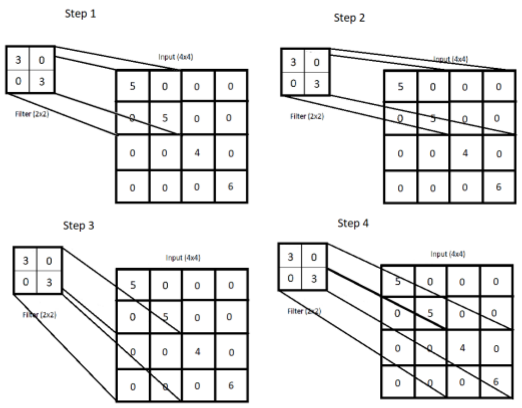
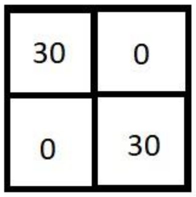


Figure 8: The convolutional operation of a filter with stride 2. Source: Figure 5 in (Westlie, 2018, p.18).



30	0
0	30

Figure 9: The output volume of the convolutional layer with 1 filter where the depth is 1. Source: Figure 6 in (Westlie, 2018, p.19).

As a result of the filter's operation, our 4x4x1 input volume has been downscaled to 2x2x1 while still containing enough information to tell us that a diagonal line was encoded in it (Westlie, 2018, p.19), due to the weights of the filter.

Moreover, a convolutional layer usually consists of several filters with depth equal to the depth of the input volume, depending on how many features one needs to detect on an object before being able to state with high confidence which class the object belongs to. If the task was to simply determine whether a diagonal exists, then one could suffice. However, if the task is to determine if the object in question is a fish, one would most likely need more information on different features before making a decision.

2.6.5 Determining the size of a convolutional layer's output

We would like to look at the size of the output volume of a convolutional layer. We assume that filters are the only module within the layer that may change the size of the input volume and that zero padding is used. If the input volume is of size $W_1 x H_1 x D_1$, the number of filters being K with spatial extent F having stride S, the formulas for the width W_2, height H_2 and depth D_2 of the output volume become as seen in the summary of convolutional layers in (Stanford University, n.d. a).

$$W_2 = \frac{W_1 - F}{S} + 1 \tag{31}$$

$$H_2 = \frac{H_1 - F}{S} + 1 \tag{32}$$

$$D_2 = K \tag{33}$$

2.6.6 Multiclass loss functions and the Softmax classifier

The multiclass hinge loss is defined as follows as seen in the chapter "Multiclass Support Vector Machine loss" of (Stanford University, n.d. b):

$$L(s) = \sum_{j \neq y_i} \max\left(0, s_j - s_{y_i} + \triangle\right) = \sum_{j \neq y_i} \max\left(0, (s_j + \triangle) - s_{y_i}\right) \tag{34}$$

Where s_j denotes the scores outputted from the classifier that are not the "true" score, and s_{y_i} the "true" score we want to predict. $\triangle$ is a hyperparameter ensuring that the loss is non-zero whenever the "true" score is larger than the sum of the predicted score and $\triangle$. Moreover, we see that the multiclass support vector machine loss is a measure of distance between the "true" score and the predicted plus some margin. Consequently, by minimizing it with respect to the network's parameters, which is possible since both s_j and s_{y_i} are outputs of network which have been processed by the network's parameters, we minimize this distance leading the network's estimated score closer to the "true" score. We also see that only when $(s_j + \triangle)$ is smaller than the true score will the loss be zero and thus we can assume that the loss function wants the network to output small scores for bad matches, and higher scores for good matches.

Furthermore, we will take a look at the multiclass cross-entropy loss function. We start of by looking at the definition of information entropy. Information entropy was defined in (Shannon, 1948) as a measure of how uncertain one can be on the outcome given "a set of possible events whose probabilities of occurrence are p_1, p_2, …, p_n" (Shannon, 1948, p.10) with unit *bits*. Its formula is given just below theorem 2 in (Shannon, 1948, p.11), where the symbol S has been chosen in this paper to represent the information entropy.

$$S(p) = -\sum_{i=1}^{n} p_i \log(p_i) \tag{35}$$

Another interpretation of information entropy which aligns better with the cross-entropy loss is how uncertain we are on the outcome of a set of events given its probability distribution.

Moving into the framework of convolutional neural networks, we need to define the Softmax classifier before the cross-entropy loss. Consider z as the vectorization of the output volume of the convolutional neural networks final layer before classification, for instance a convolutional layer, with length equal to the number of classes the convolutional neural network is set to predict, and z_j as the value of entry j in z belonging to a particular class. A Softmax classifier interprets this value as the unnormalized log probability of the correct class being j and turns that unnormalized log probability into a normalized probability, with value in range $< 0,1 >$. Its formula, which converts the log probability z_j to a normalized probability, is as seen in the chapter "Softmax classifier" of (Stanford University, n.d. a).

$$f_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \qquad 36$$

where N denotes the number of classes. It is important to note that the vector $f(z)$ outputs an estimated probability distribution over the set of all classes.

This leads us to the multiclass cross-entropy loss function. It is defined in Equation 37 according to (Stanford University, n.d. b) from the source's section "Information theory view", and put into context with the Softmax function in Equation 38. $S(p, q)$ represents the cross-entropy given the "true" probability distribution $p(x)$ and the estimated probability distribution $q(x)$, and $L_{CE}(p, f)$ when it is used as a loss function where the Softmax function is used to estimate the probability distribution.

$$S(p, q) = -\sum_{x} p(x)\log(q(x)) \qquad 37$$

$$L_{CE}(p,f) = -\sum_{i=1}^{N} p(i)\log(f_i(z)) \qquad 38$$

$f_i(z)$ is the output from the Softmax function for class i, and $p(x)$ normally is the Kronecker delta Dirac function putting all its mass on the correct class, and N is the number of classes. Say the correct class is k, then only when i equals k would $p(i)$ be non-zero and set to 1. Finally, if we look at p as a vector with one entry being 1, the rest set to 0, containing the "true" probability distribution over the set of classes and $f(z)$ as the estimated, (Stanford University, n.d. b), in the source's section "Information theory view", mentions that $L_{CE}(p,f)$ becomes a measure of distance between the two distributions. Consequently, by minimizing it with respect to the network's parameters, which is possible since $f_i(z)$ depends on the parameters of the network, one minimizes the distance between the "true" and the estimated probability distribution.

2.6.7 How Deep Learning methods learn through mini-batch gradient descent

The following theory is based on (Keskar et al., 2017, pp.1-2). The learning process of neural networks can be formulated as a non-convex optimization problem as seen in equation 39, which has been written with slightly different notation than as seen in equation 1 from (Keskar et al., 2017, p.1).

$$\min_{x\in\mathbb{R}^n} L(x) \coloneqq \frac{1}{M}\sum_{i=1}^{M} L_i(x) \qquad 39$$

Where $L_i(x)$ is a loss function for the data samples $i \in \{1,2,\dots,M\}$. Furthermore, to minimize the function, the mini-batch gradient descent may be used as seen in equation 2 from (Keskar et al., 2017, p.2).

$$x_{k+1} = x_k - \alpha_k \left(\frac{1}{|B_k|} \sum_{i \in B_k} \nabla L_i(x_k) \right) \qquad 40$$

Where x_k and x_{k+1} are the values of the weights before and after the update, α_k the learning rate, B_k the mini-batch, $|B_k|$ the number of samples in the mini-batch, and $\nabla L_i(x_k)$ the gradient of the loss function evaluated at x_k. In essence, the network learns by processing all samples within the mini-batch, and then updates the weight's according to equation 40. The method is called mini-batch gradient descent as it uses only a subset of the data samples available in the dataset to estimate the gradient of the loss function at every step.

2.6.8 Batch normalization

The following theory is based on (Ioffe & Szegedy, 2015). Batch normalization, if chosen to be included in a convolutional neural network, is a part of its convolutional layers, and may be used when the network trains on mini-batches. Batch normalization facilitates the usage of higher learning rates, negates the effects weight initialization may have on the convolutional neural network and reduces the risk of overfitting (Ioffe & Szegedy, 2015, p.1).

The following citation from (Santurkar et al., 2019, p.2) summarizes the most important aspects of batch normalization.

> Broadly speaking, BatchNorm is a mechanism that aims to stabilize the distribution (over a minibatch) of inputs to a given network layer during training. This is achieved by augmenting the network with additional layers that set the first two moments (mean and variance) of the distribution of each activation to be zero and one respectively. Then, the batch normalized inputs are also typically scaled and shifted based on trainable parameters to preserve model expressivity. This normalization is applied before the non-linearity of the previous layer.

2.6.9 Aspects to consider when training neural networks with mini-batch gradient descent

Overfitting

Overfitting occurs when the dataset used to train a neural network is limited in for instance size or diversity. Then, neural networks will not learn the complicated relationships between the input and output, but rather the relationship between the sampling noise present in the training data and the output. Consequently, the network will struggle to generalize on new data, as the noise in the new data might vary from that present in the training data (Srivastava et al., 2014, p.1).

Several methods have been introduced to combat this issue. (Srivastava et al., 2014, p.1) mentions splitting the dataset into a training and validation set, which is a technique called cross-validation. When the accuracy of the validation set diverges from that of the training set, especially when the training accuracy increases, and validation decreases, one should stop training as it is a clear sign that the network is overfitting. Another method would simply be to create a large training set that covers most aspects of the data, so that the network gets new inputs and does not learn the patterns of the sampling noise.

Learning rate scheme

In the introduction of (Brownlee, 2019a) it is mentioned that using a large learning rate can lead to unstable performances, while small learning rates may lead to the network being stuck, i.e in a sub-optimal solution yielding low prediction accuracy. Furthermore, if opting for a fixed learning rate scheme one needs to choose a small learning rate that finds a compromise between having unstable performance results and not failing to train. Choosing such a scheme would make the learning process slow. To gain good results while still training relatively quickly, one could introduce a parameter of "patience". Meaning that the learning rate stays fixed for a number of epochs before being decayed at the every "patience" epoch as seen in the chapter "Drop Learning Rate on Plateau" of (Brownlee, 2019b). For instance, one could opt to lower the learning rate every 5th epoch.

Furthermore, if one simple does not want to bother with finding the exact optimal learning rate, there exists learning rate optimizers (Ruder, 2017) that make educated guesses as to what the learning rate should be at every iteration of the training process when for instance using mini-batch gradient descent to train the network.

Batch size

(Keskar et al., 2017, pp.2-3) defines the typical size of a mini-batch B_k used with mini-batch gradient descent as $|B_k| \in \{32,64,\dots,512\}$. The same source mentions that larger mini-batch sizes tend to decrease a network's ability to generalize due to the network converging to sharp minimizers of the loss function. Smaller batch sizes, however, tend to find flat minimizers which gives the network a better ability to generalize to new data. Consequently, by training a network with a batch size of say 256, the network could produce worse results when inferring on new data than if one were to use a smaller batch size.

Finally, (Bengio, 2012, p.5) reports that an increase in the batch size allows one to slightly increase the learning rate because the gradient found will be less noisy due to averaging over the batch size.

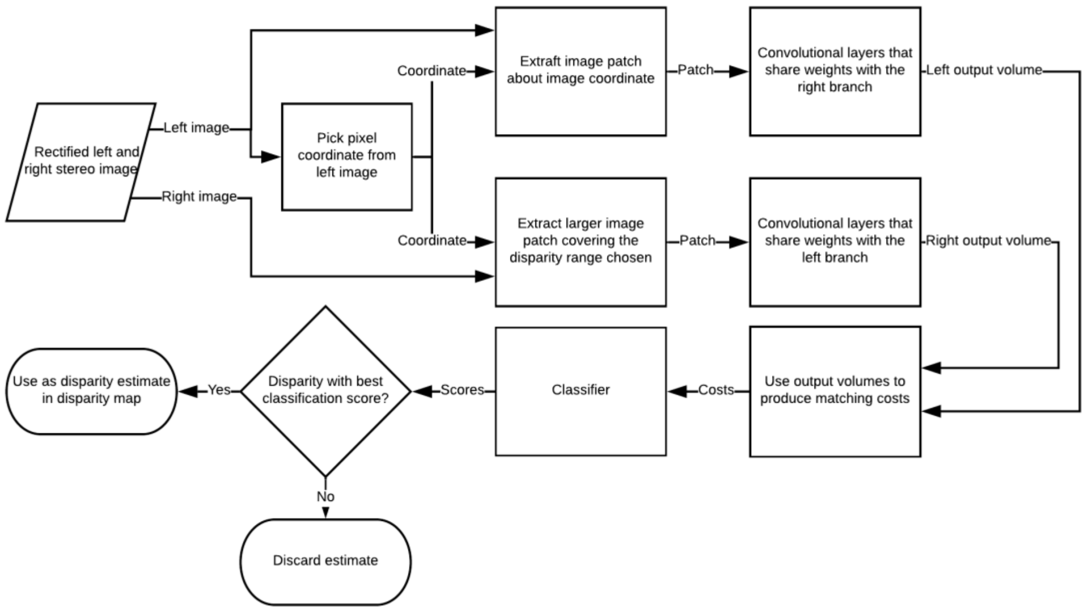
2.7 Convolutional Neural Network stereo-matching algorithms

2.7.1 Why can convolutional neural networks be stereo matchers

A key concept of stereo matchers is the calculation of a matching cost, or in other words finding a measure of dissimilarity between two pixels. Following the development of C*onvolutional Neural Networks* which learn distinctive features of an object by training on heaps of data, a new way of creating descriptors from pixel intensity values was discovered. Essentially, the convolutional neural network's output after having operated on the image patch, is an output volume where each entry quantifies whether a specific feature was present or not. The features in question are specific to an object such as parts of its shape or color information. Consequently, the output of the convolutional neural network may be viewed as a descriptor of a pixel and its surroundings (Zagoruyko & Komodakis, 2015, p.5). Furthermore, common ways to measure the dissimilarity, matching cost, between two patches centered about a pixel would be to input both descriptors to an artificial neural network (Žbontar & LeCun, n.d., p.6) or join them by the dot product or cosine similarity as seen in (Žbontar & LeCun, 2016, p.5).

2.7.2 Using Convolutional Neural Networks to estimate disparity

We will begin the description by looking at Figure 10, a workflow diagram which shows the general steps taken from inputting a pair of rectified stereo images to outputting a disparity estimate.



Rectified left and right stereo image
Left image
Right image
Pick pixel coordinate from left image
Coordinate
Coordinate
Extraft image patch about image coordinate
Extract larger image patch covering the disparity range chosen
Patch
Patch
Convolutional layers that share weights with the right branch
Convolutional layers that share weights with the left branch
Left output volume
Right output volume
Use output volumes to produce matching costs
Costs
Classifier
Scores
Disparity with best classification score?
Yes
Use as disparity estimate in disparity map
No
Discard estimate

Figure 10: Workflow diagram depicting the general steps a convolutional neural network takes when estimating disparity.

The right and left branch are realized by a siamese convolutional neural network, which means that the filters in each branch shares weights. The difference between each branch would be the size of their input and output volumes. We consider a grayscale image as input to the network, with K filters in the final layer before classification, and that the disparity range is [0,170]. The goal of the left branch is to output a volume which describes a patch centered about an arbitrary pixel in the left image with size $1x1xK$. $1x1xK$ denotes the branch has created a descriptor for one pixel and its surroundings with the output of the final K filters. The goal of the right branch is to output a volume describing all pixels within the disparity range of the right patch, centered about the pixel location chosen in the left image. Consequently, the final size of the output volume becomes $1x(170+1)xK \Leftrightarrow 1x171xK$. $1x171xK$ denotes that the branch has created a descriptor for 171 pixels and their surroundings with the output of the final K filters covering the entire disparity range. Furthermore, the two output volumes are joined, leading to a matching cost volume of size 1x171x1 which is flattened to a vector of length 171. Finally, this vector is used in classification to decide which of the 171 disparities got the best score.

2.8 The specifics of MC-CNN-fst's Convolutional Neural Network part

2.8.1 Inside MC-CNN-fst's convolutional layers

MC-cnn-fst's convolutional layers consist of a spatial convolution module implementing the filters and biases. Furtheremore, ReLU is used as the activation function at every convolutional layer except the last one to not lose "information encoded in the negative values" (Luo et al., 2016, p.5697).

2.8.2 Loss function and classification

The last layer of the network before classification is a "StereoJoin" layer which computes the cosine similarity for every pair of patches associated with a distinct disparity by normalizing and taking the dot product. This serves as a measure of similarity between both patches, however the assumption that a score of 1 is perfect (parallel vectors) is not true in this case as we will see shortly.

MC-CNN-fst uses the following variant of the multiclass hinge loss function, L_{MC}.

$$L_{MC} = \max(0, \triangle + s_{-} - s_{+}) \qquad 41$$

$\triangle = 0.2$ is the margin, s_{-} is the score of a negative sample and s_{+} a positive. Exactly how s_{-} and s_{+} are derived is described in the following citation from (Žbontar & LeCun, 2016, p.4).

> A positive example is a pair of patches, one from the left and one from the right image, whose center pixels are the images of the same 3D point, while a negative example is a pair of patches where this is not the case.

In addition, the same source mentions that the center pixel of the right patch leading to s_+ is randomly chosen to be within 1 pixel of the "true" pixel, i.e. one shifts the right center pixel's location 1 position to the right or the left. The reason being is that the method's "disparity refinement" operations perform better when not only the best match is assigned a low score by the network, but also matches close to it. The center pixel of the right patch leading to s_- is chosen the same way, however, stays within for instance 10 pixels of the "true" pixel.

Moreover, owing to equation 41 we see that the loss is zero only when $s_+ \geq s_- + \triangle$. Both s_- and s_+ are outputs of the network. We update the parameters of the network by minimizing its loss, and so what defines the value of s_+, and s_- is the network itself and not the natural way of thinking that s_+ should be 1 because then vectors are parallel. We can however say that the network optimizes its weights to give a good match higher score than a bad plus the margin.

Finally, MC-CNN implements a simple classifier as the final stage of the network. The classifier assumes we have got similarity scores for every disparity in the disparity range (that is not outside the bounds of the image) and chooses the disparity associated with the highest score.

2.9 The specifics of Content-CNN's Convolutional Neural Network part

2.9.1 Inside Content-CNN's convolutional layers

The convolutional layers of Content-CNN contain the following methods. First of there is a spatial convolution module implementing the weights and biases of the filters, then a batch normalization module. ReLU is used as the activation function on all convolutional layers except the last one to keep the information the negative values encode.

2.9.2 Classification method and loss function

Furthermore, Content-CNN uses the Softmax classifier as part of the final stage of the network. As we recall, the Softmax classifier outputs a probability distribution over the entire disparity range. Consequently, the disparity that is chosen as the best estimate is associated with the highest probability.

Finally, the loss function used is the multiclass cross entropy loss with some interesting nuisances. It is shown below and extracted from (Luo et al., 2016, p.5697).

$$L_{Content} = \sum_{i,y_i} p_{gt}(y_i) \log(p_i(y_i, w)) \tag{42}$$

Where $p_{gt}(y_i)$, the "true" probability distribution, is not the Kronecker delta function, but rather a smooth target distribution, and $p_i(y_i, w)$ the output of the Softmax classifier. The entry of the target distribution related to the "true" disparity is set to 0.5. At the entries 1 and 2 pixels away from the "true" disparity, a probability of 0.2, and 0.05 is present respectively. Finally, all other disparities are deemed to be bad and therefore their entries are set to 0.0. The reasoning behind incentivizing matches that are not entirely on point is due to two aspects. Firstly, by rewarding matches close to the true match, one opens the network to capture correlations between the "true" disparity $\pm$ 2. Thus, there is some leeway in the network in that it does not have to strictly differentiate between the perfect match and all others. Secondly, the network is optimized with the 3-pixel-error in mind, and so rewarding matches within this accepted range of the "true" match is beneficial.

3 Method

3.1 Gathering information on methods for ground truth disparity map creation

3.1.1 Finding methods for ground truth disparity map creation

Google Scholar, GitHub and Google were used in finding methods for ground truth disparity map creation. Searches on keywords such as “stereo dataset”, laid the foundation of the search, and the resulting papers and descriptions found through it were skimmed for new sources. NTNU’s ORIA was used to determine whether a paper had been peer-reviewed, and Google to determine the papers’ citation count.

3.1.2 Determining the methods’ accuracy and general requirements

When it comes to the accuracy, the paper’s associated with each method was skimmed over and searches on keywords “accuracy”, and “error” was performed. Moreover, when looking for their general requirements the same process was repeated and the following keywords used in the search: “assumption”, “allow”, “static” and “dynamic”.

3.2 Gathering information on deep learning stereo matchers

3.2.1 Finding methods and their implementations

Google and the KITTI 2015 stereo evaluation’s leaderboard (Geiger et al., 2015b) were the main sources of information. The leaderboard included references to the method’s paper and its implementation, and so it became the main source used in finding other methods and their implementations. NTNU’s ORIA was used to determine whether a paper had been peer-reviewed, and Google to determine the papers’ citation count.

3.2.2 Dissecting the details of each implementation's code

To find whether the code for training, inferring, preparation of data, data augmentation and post-processing/disparity refinement was available or not, the repositories "usage" chapter was consulted. Additionally, the code files used in inferring and training were read in order to determine if they cover one of the mentioned categories. Finally, the paper of each implementation was skimmed over and a search using the keywords "code", "train", "pre", "preprocess", "inferring", "data", "augmentation", "post" and "refine" was conducted for each paper.

3.2.3 Finding the requirements needed to run each implementation

When looking for the specifications needed to run each method, the following method was used. The papers and code repositories were skimmed over, and a search on relevant keywords such as "gpu", "amd", "nvidia", "cuda", "cudnn", "operating system" was conducted. In addition, when having found the frameworks, drivers and libraries needed, their required specifications were found by Googling for instance "torch requirements".

3.2.4 Finding the methods' accuracy results and runtime

The KITTI 2015 Stereo Evaluation (Geiger et al., 2015b) was consulted when looking for accuracy results and runtime, as all methods found had been compared to one another through it. Only the accuracies of the "D1-all" metric was included, as it is a measure of how many disparity estimates were wrong when considering all disparities in the ground truth disparity map. Additionally, the papers of each method were read if the accuracy or runtime was missing from the leaderboard.

3.3 Capturing the images for the dataset

The images were captured at Optoscale`s facilities. A fish model attached to a rod as seen in Figure 11 was lowered in front of Optoscale's BioScope (Optoscale AS, n.d.) in a freshwater pool as seen in Figure 12. The pool was enclosed by curtains in order to reduce the illumination coming from the outside. Figure 13 shows the scene as it was when the cameras were capturing. The cameras captured a series of 200 image pairs taken at 1 second intervals, and used a software trigger to synchronize the capture. The fish model was slowly moved while recording to get images of different angles and distances. Figure 14 shows an example of the color images captured. In total 3 series were captured making the dataset consist of **600** image pairs.

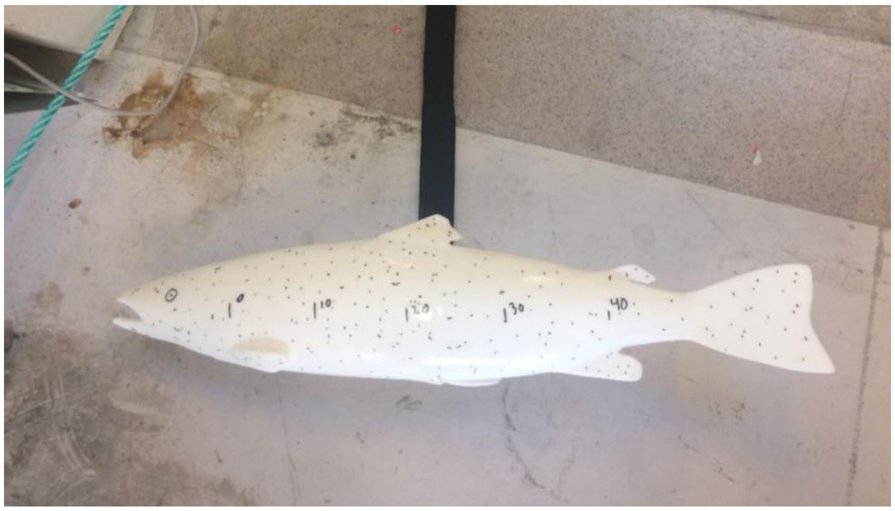


Figure 11: A picture of the fish model.

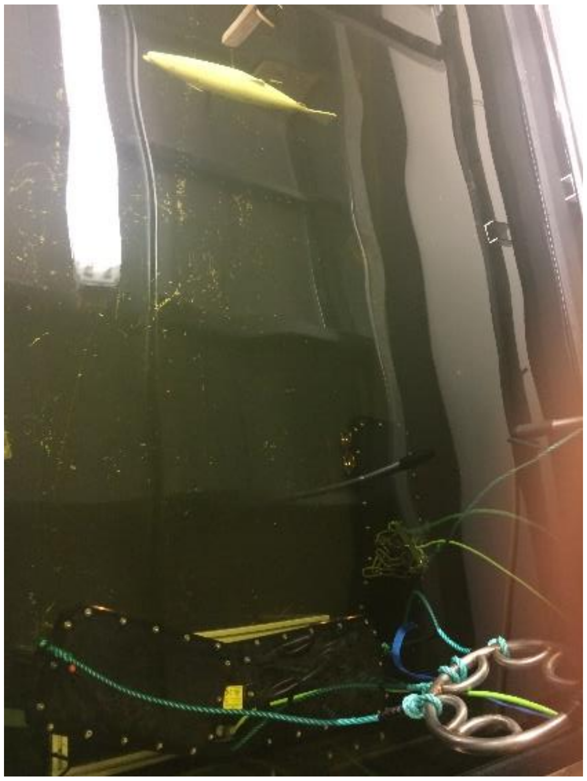


Figure 12: A picture of the setup in the freshwater tank. Optoscale's BioScope is at the bottom of the image.

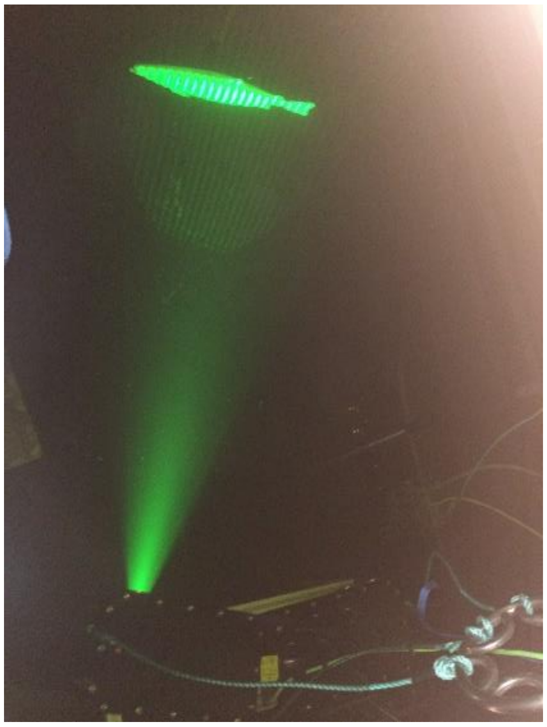


Figure 13: Picture of the setup when its capturing image. The source of the strong illumination seen in the upper right corner was not present when the real images were taken.

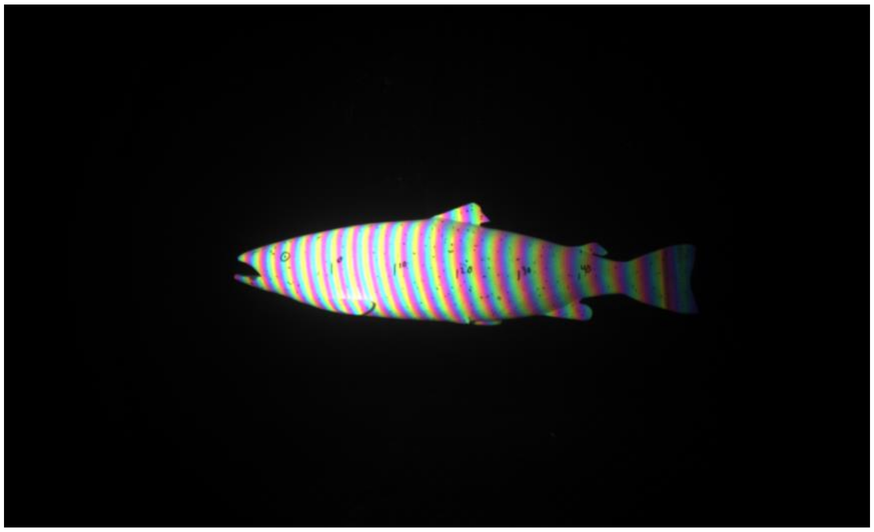


Figure 14: A picture captured by Optoscale's BioScope.

3.4 Choosing method which will be used to create ground truth disparity maps

Optoscale and the author agreed that ground truth disparity maps would be created by optimizing their algorithm which uses MATLAB’s SGBM stereo matcher. The decision has consequences which is discussed in section 6.3.

3.5 From raw images to ground truth disparity map

3.5.1 Stereo rectification

Stereo rectification was the first step needed to be performed in order to limit the search for a match to the horizontal axis, and to enable stereo-matching algorithms which assume this had been done. Optoscale provided the stereo parameters for the cameras, which had a mean projection error of 0.1924 pixels over 136 images of a checkerboard at different angles and distances in the freshwater pool. Finally, the image pair was passed to MATLAB’s “rectifyStereoImages” function along with the stereo parameters.

3.5.2 Removing stripe patterns when converting to gray scale

A traditional RGB to grayscale conversion would result in an image which preserves the striped structured light pattern as seen in Figure 15. The darkened stripes may cause issues when for instance thresholding the image later on as their intensities are lower and thus closer to the background pixels' intensity values. The issue is fixed by using the following "gray scale" conversion, which was received from Optoscale, and described with permission. $I_{rgb}(i,j,k)$ is an RGB image, and $I_{"gray"}(i,j)$ the "gray scaled" image of same size, where i denotes the row index, j the column index and k the channel. Remove the effects of structured light by setting the intensity of $I_{"gray"}(i,j)$ according to the following formula.

$$I_{"gray"}(i,j) = \frac{\sum_{k=1}^{3} I_{rgb}(i,j,k)}{3} \qquad 43$$

The result of the conversion is shown in Figure 16.

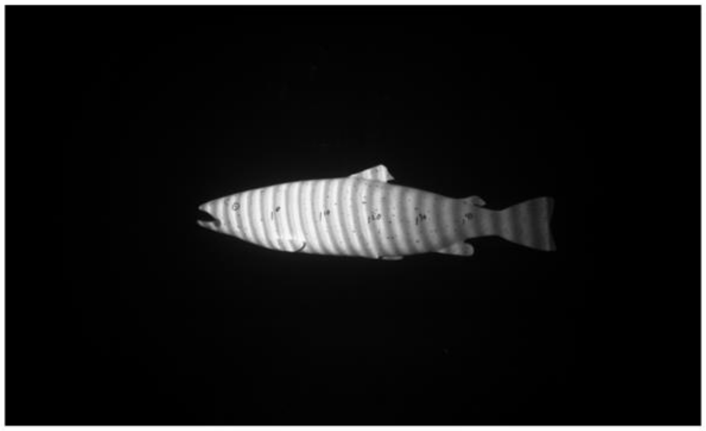


Figure 15: Result when using standard grayscale conversion.

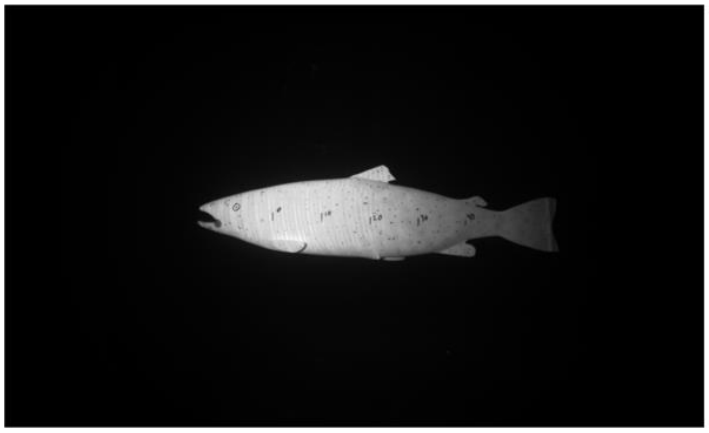


Figure 16: Result when using the "gray scale" method.

3.5.3 Outlining a method to find the binary image masking the pixels belonging to the fish

Finding the mask of the fish present in each image is an important step in creating the disparity map that will serve as ground truth. The mask enables the removal of disparity estimates that do not belong to the fish, in addition to allowing the cropping of images. To do so, a filtering pipeline consisting of 8 filter modules and a control module was created to identify the pixels in the image which belongs to the fish and not the background or other noise elements. The entire pipeline is summarized by the workflow diagram in Figure 17, and the script implementing the filter modules may be found in section 10.2.1.

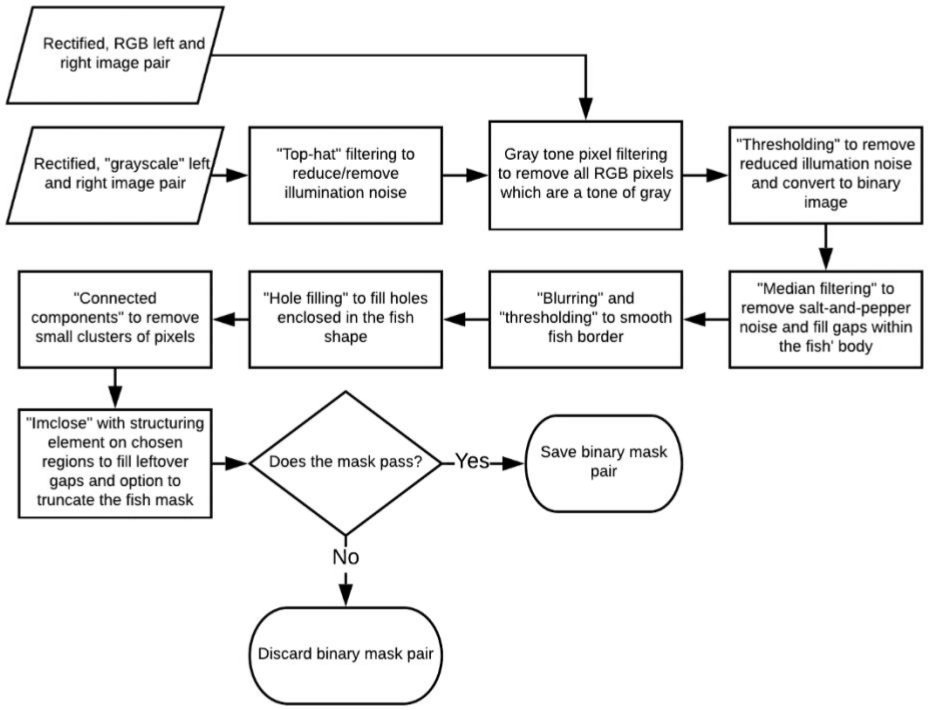

Rectified, RGB left and right image pair
Rectified, "grayscale" left and right image pair
"Top-hat" filtering to reduce/remove illumination noise
Gray tone pixel filtering to remove all RGB pixels which are a tone of gray
"Thresholding" to remove reduced illumation noise and convert to binary image
"Median filtering" to remove salt-and-pepper noise and fill gaps within the fish' body
"Blurring" and "thresholding" to smooth fish border
"Hole filling" to fill holes enclosed in the fish shape
"Connected components" to remove small clusters of pixels
"Imclose" with structuring element on chosen regions to fill leftover gaps and option to truncate the fish mask
Does the mask pass?
Yes
Save binary mask pair
No
Discard binary mask pair

Figure 17: Workflow diagram depicting the process of finding the fish' binary image.

3.5.4 White Top-hat filtering to reduce the non-uniform noise

White Top-hat filtering is effective in separating bright objects from dark ones, and thus it is the first step of the filtering pipeline. Figure 18 shows the binary image of the grayscale image before top-hat filtering, and Figure 19 shows the binary image after the filtering. Binary images were used to show the result of the operation, as it better illustrates the effects of top-hat filtering than showing the top-hat filtered grayscale image. The white pixels in the binary image correspond to pixels in the grayscale image with intensity above zero. The structuring element was chosen as a disk with radius 60 pixels.

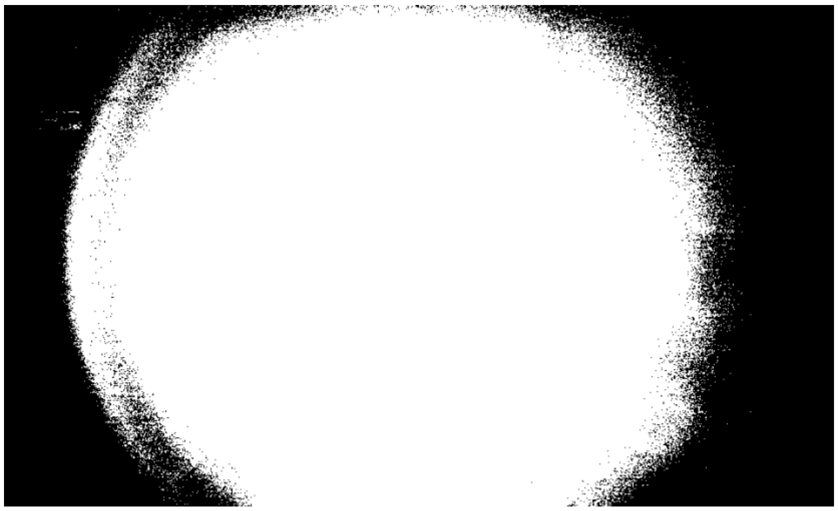


Figure 18: Binary image of the grayscale where all white pixels coincide with intensity values in the grayscale image being above zero.

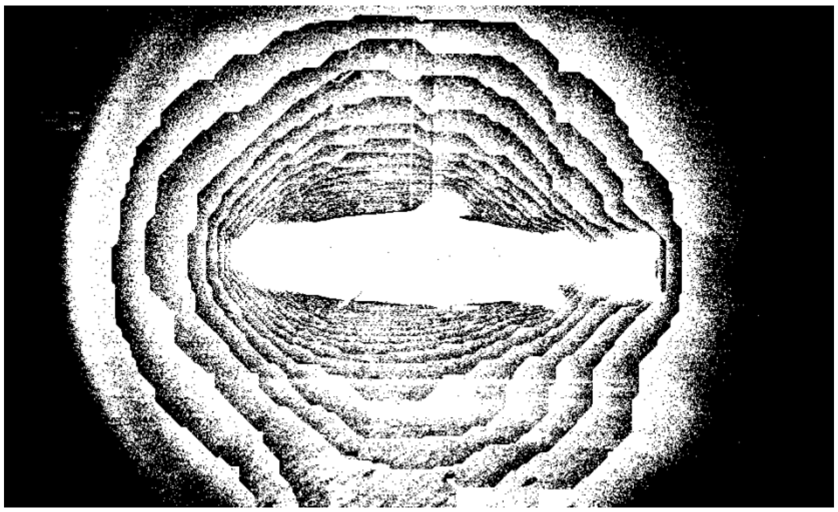


Figure 19: Result of the Top-hat filter.

3.5.5 Removing background pixels not colorized by the structured light

In Figure 14 the fish is clearly colorized, whereas the background is in essence different tones of gray. This observation was confirmed by using the “Digital Color Meter” app available in MacOS (Apple, n.d.). Consequently, one should be able to remove some non-fish related pixels by checking if they are tones of gray. One can check if the difference between the red and green, green and blue and blue and red channel values of each pixel is within 1 value of each other. If so, they make out a tone of gray and may be removed from the binary image. The method is described below, and its result is shown in Figure 20.

- Let $i = 1,2, \dots, I_h, j = 1,2, \dots, I_w$ and $k = 1,2,3$ be the row, column and channel numbers of an RGB image.
- Let $I(i, j, k)$ be the value of a pixel at location (i, j) and color channel k.
- Let $D_{12}^{abs}(i, j)$, $D_{23}^{abs}(i, j)$ and $D_{31}^{abs}(i, j)$ denote the absolute value of the difference between the image’s red and green, green and blue and blue and red channel’s value at pixel location (i, j). For instance, $D_{12}^{abs}(i, j) = |I(i, j, 1) - I(i, j, 2)|$.
- Let $M(i, j)$ be an empty binary image with the same width and height as I.
- Iterate through every value of i and j and set the entry in $M(i, j)$ according to equation 44.

$$M(i,j) = \begin{cases} 0 & if \;\; D_{12}^{abs}(i,j) \leq 1 \land D_{23}^{abs}(i,j) \leq 1 \land D_{31}^{abs}(i,j) \leq 1 \\ 1 & otherwise \end{cases} \qquad 44$$

- Finally, set $I(i, j, k)$ to the result of the element wise multiplication of $I(i, j, k)$ and $M(i, j)$ for every channel k to remove gray toned pixels from the image.

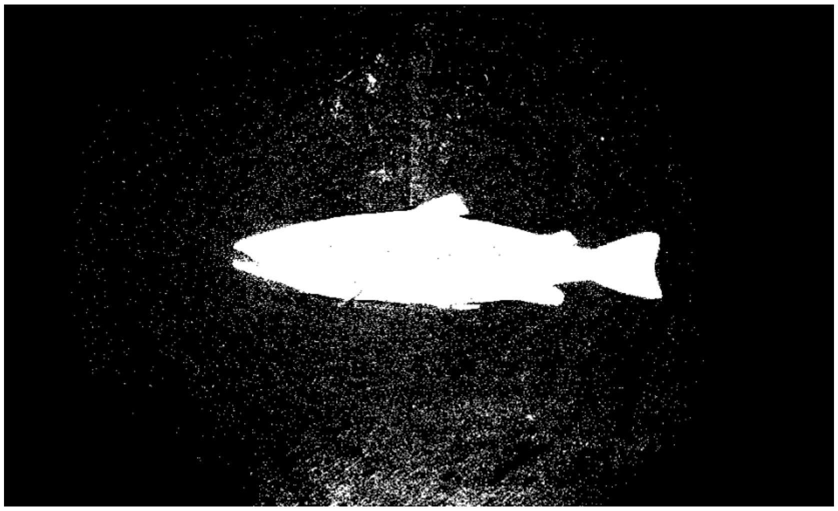


Figure 20: Result of removing pixels that are tones of gray.

3.5.6 Thresholding to remove low intensity noise

The top-hat filter has laid the foundation for thresholding to take place. Since small blobs of nonuniform lighting has gotten its values greatly reduced or set to zero, the first step of the thresholding procedure is to set all pixels below a certain threshold to zero. By experimenting on certain images of the dataset which contained the highest and lowest intensity values, it was determined that the optimal threshold value should be the mean of all pixel intensities in the grayscale image divided by $\frac{1}{3}$. Thus, all pixels with intensity less than or equal to this value was set to zero. The result of the operation is shown in Figure 21.

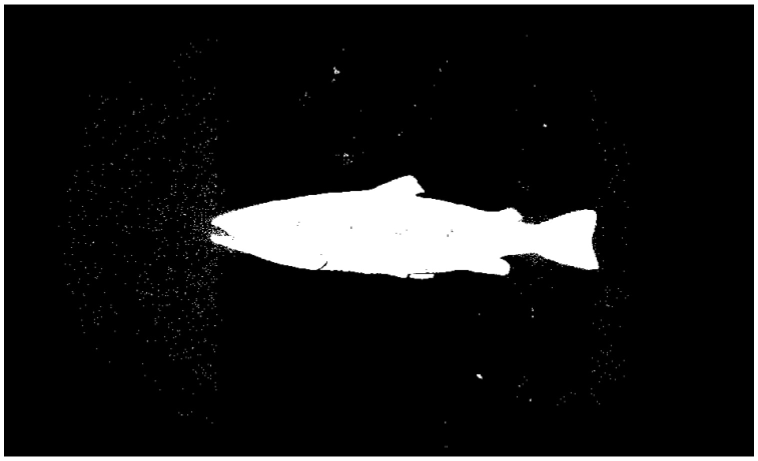


Figure 21: Result of mean thresholding.

Furthermore, a form of column thresholding was introduced to remove most of the illumination noise that was resilient to the previous filters. As the structured light emitter emits vertical color beams, one would assume that most pixels that belong to the fish in a column of the image would have a relatively small variation in intensity values compared to for instance the variance in intensity of all pixels belonging to the fish or even the rows of pixels. The method consists of finding the mean intensity value of a column where all column pixels with value 0 are excluded from the calculation of the mean and then scale the mean by a factor with value less than 1 to not remove values below the mean that still belong to the fish. By experimenting on images from the dataset with small and high pixel intensities it was found that when scaling the column mean by $\frac{1}{10}$, pixels that belong to the fish were preserved and non-fish related pixels were excluded. Furthermore, the output of the thresholding filter is a binary image. The result of the operation is shown in Figure 22.

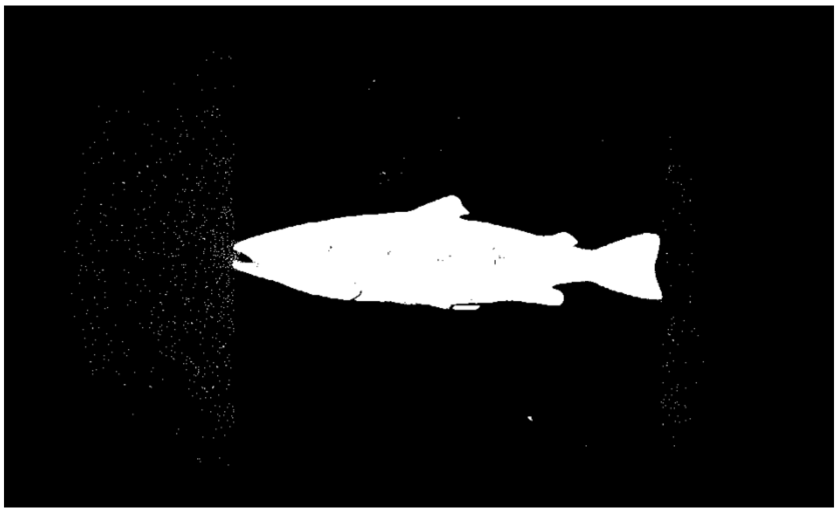


Figure 22: Result of the specialized column thresholding method.

3.5.7 Median filtering to remove salt-n-pepper noise and repair gaps in the binary image

The median filter is needed to remove the scattered small blobs of non-fish related pixels left, in addition to repairing small pixel gaps along the edge of the fish' body that were caused by the previous filters. The filter size was set to 5x5, and the result is shown in Figure 23.

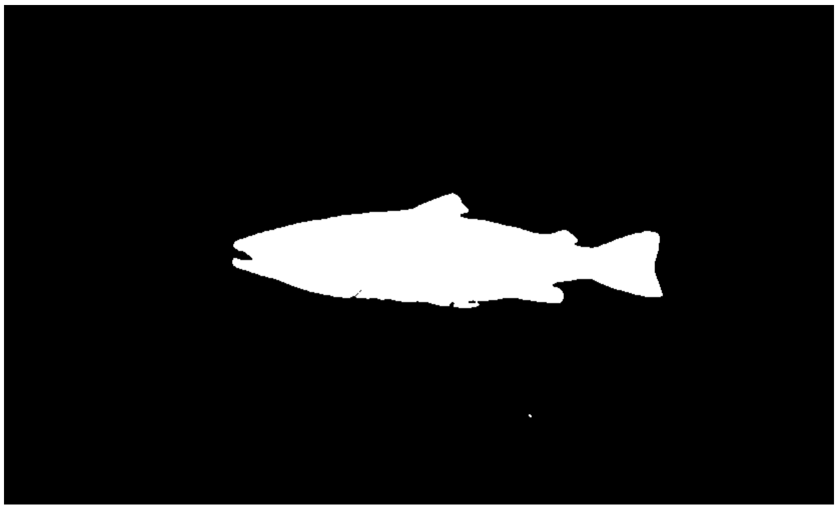


Figure 23: Result of the median filter.

3.5.8 Blurring and thresholding to smooth edges and fill gaps

The previous filters have in some instances caused the border of the fish to appear rough. From our knowledge of the fish' shape, the borders should be smooth. Moreover, it was discovered that the filter also serves as a gap filler. The method blurs the binary image by a filter of size 7x7, and slides over the image through 2D-convolution. The resulting image was thresholded at 0.5. The result of the operation is shown in Figure 24.

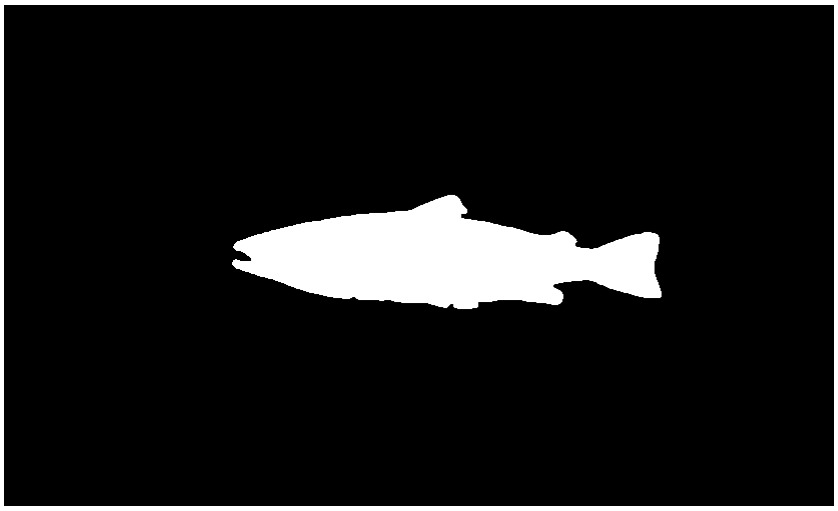


Figure 24: Result of the edge smoothing operation.

3.5.9 Filling larger holes within the binary image

Hole filling is needed as previous filters are not perfect and thus some pixels within the border of the fish have been set to 0. As the border of the fish has been established by the former filters, the following hole filling algorithm was used. The algorithm defines the background pixel as the intensity value at the edge of the image, so in this instance it would be 0. Then it checks if it can pave a path along background pixels from the edge of the image to a certain pixel. If this is not possible for a given pixel, its assumed to be a hole within an object and its value is set to 1. As not all images are affected by every filter, a new image that is outputted from the blurring module is shown in Figure 25. The result of the operation is shown in Figure 26.

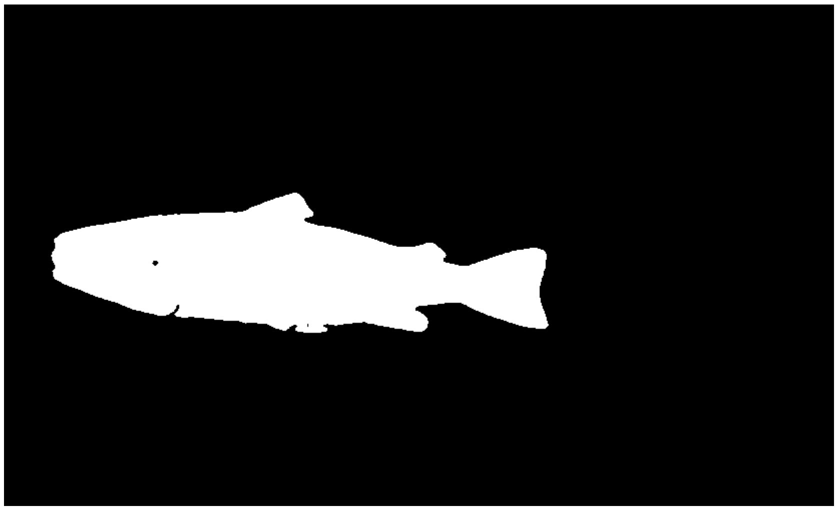


Figure 25: The binary image before hole filling.

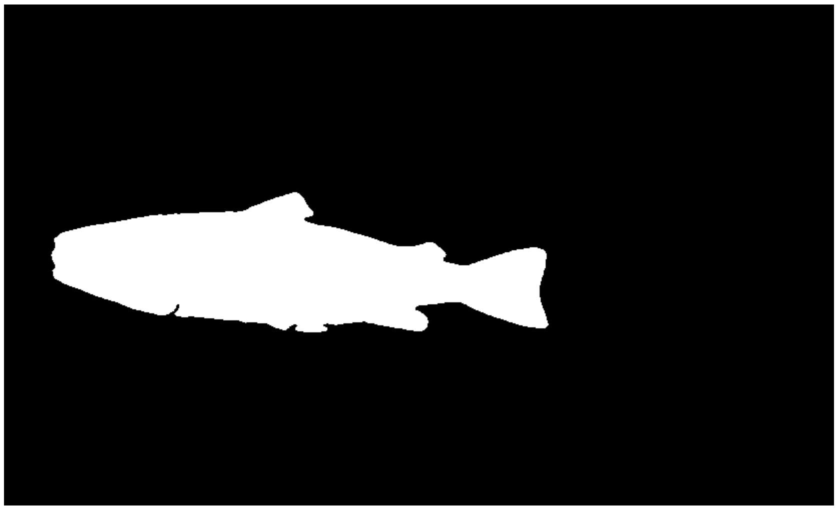


Figure 26: The binary image after hole filling.

3.5.10 Removing smaller blobs of pixels through connected components

The eight step is to remove all clusters consisting of 20,000 or less pixels with value 1. As we are quite confident that we have found the majority of pixels belong to the fish, all other smaller blobs can be removed. A new image where the connected components module became relevant is shown in Figure 27, and the result of the operation is shown in Figure 28.

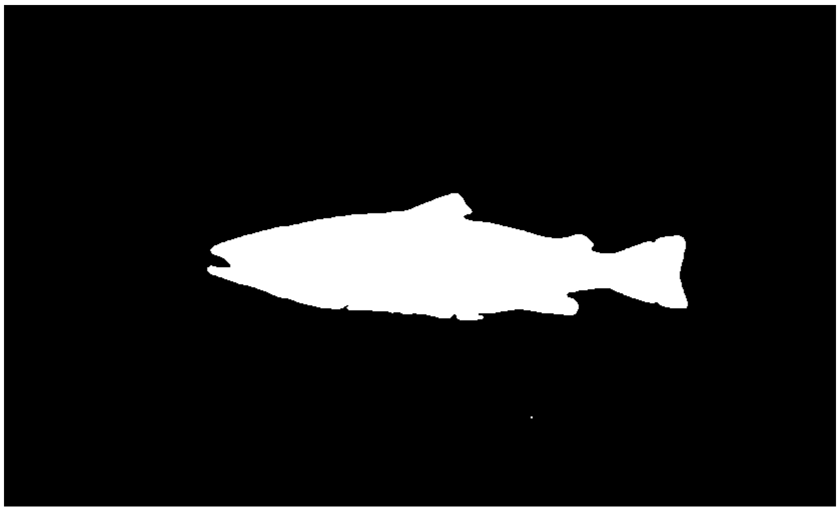


Figure 27: Example of a binary image where a cluster of pixels not belonging to is present.

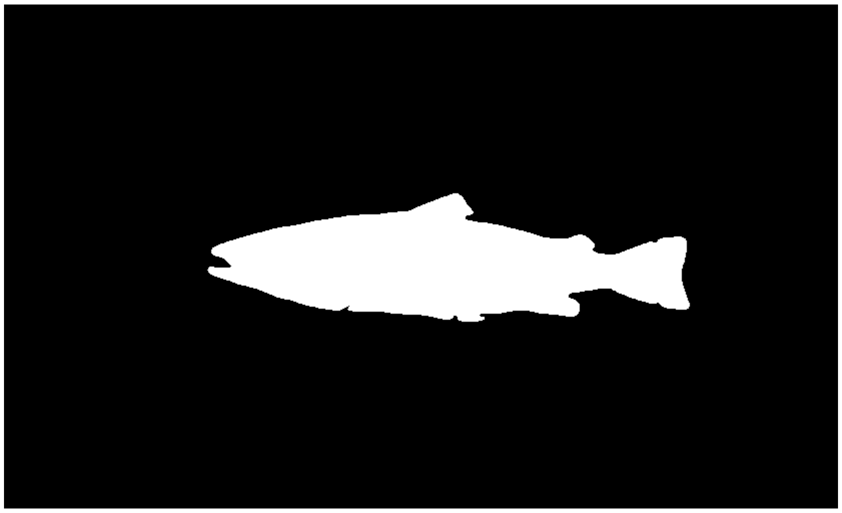


Figure 28: Result of the cluster size threshold method.

3.5.11 Manually mark where to fill gaps and crop unwanted parts

This step allows the user to manually choose sections of the image were a filter should operate and to remove certain parts of the binary mask if it was uncertain that it captured the pixels belonging to the fish. A script was created in MATLAB, which may be seen in section 10.2.2, that allows the user to manually set the range of columns a filter should operate on. The binary image was plotted, and the user may interactively choose two ranges of columns, $[c^1, c^2]$ and $[c^3, c^4]$, on which MATLAB's morphological "imclose" method, with a structuring element in the form of a line with length 19, will operate on by choosing points with the mouse cursor. Additionally, the user may choose two additional points which define the left and right sector where all pixel's values within those sectors will be set to zero. To exemplify the latter part, the fifth point denotes column c^5, the sixth c^6 and I_w denotes the image width. Then all pixels within columns [1, c^5] and $[c^6, I_w]$ will be set to zero. The latter part of the method is useful as sometimes it was hard to tell whether the most left or right region of the image belonged to the fish or not. Consequently, by truncating the image at a safe location, one can be more certain that only fish related pixels are included. Figure 29 shows the operation of interactively choosing points, and Figure 30 shows the result.

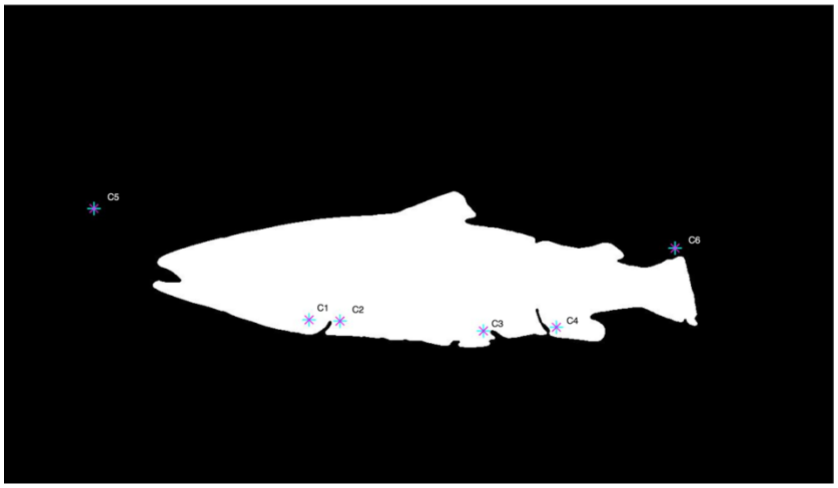


Figure 29: The interactive process of marking the gaps that should be filled and areas to truncate.

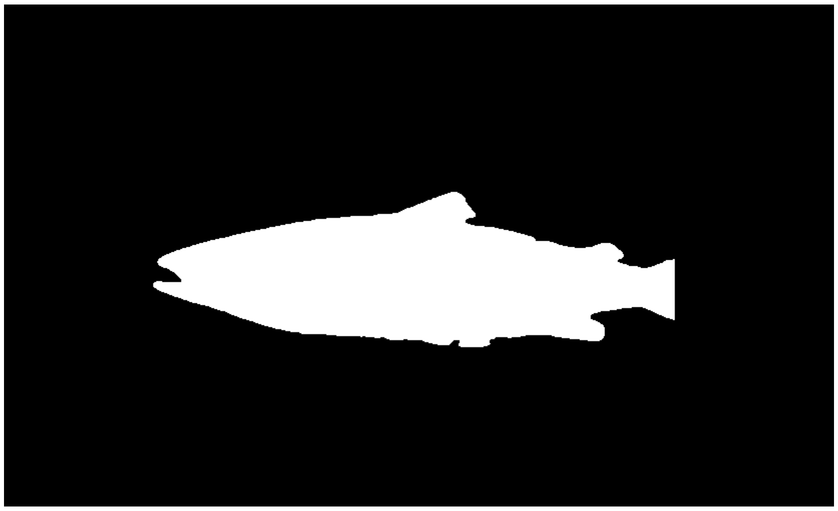


Figure 30: Result of the interactive process.

3.5.12 Controlling how well the binary image fits the original fish image

The final step of the procedure was to control the fit of the binary mask. To do so, the boundary of the binary masked fish was found through MATLAB's "canny edge" detection method as shown in Figure 31. Subsequently, the boundary image was imposed onto the "grayscale" image that coincided with the binary mask. To separate the pixels belonging to the fish and the pixels that are part of the boundary of the binary mask, the boundary pixels were set to the highest value 255. Figure 32 depicts the left and right images after the boundary of the binary mask was imposed. Furthermore, the control phase consisted of visually inspecting if only pixels belonging to the fish were present within the boundary or if it captured background pixels as well. If It was obvious that the boundary included background pixels far from the borders of the fish, the image pair was discarded. It was paid special attention to the left binary mask fit, as this is the most important when it comes to creating the ground truth disparity map. Figure 32 also shows an example of an accepted binary mask, whereas Figure 33 shows an example of a discarded one. In total, **598** image pairs were accepted and 2 discarded. The script implementing the method may be found in section 10.2.3.

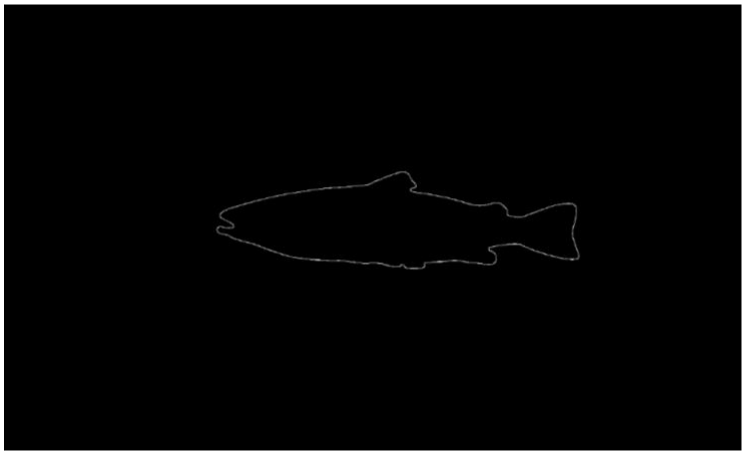


Figure 31: The result of the canny edge detector.

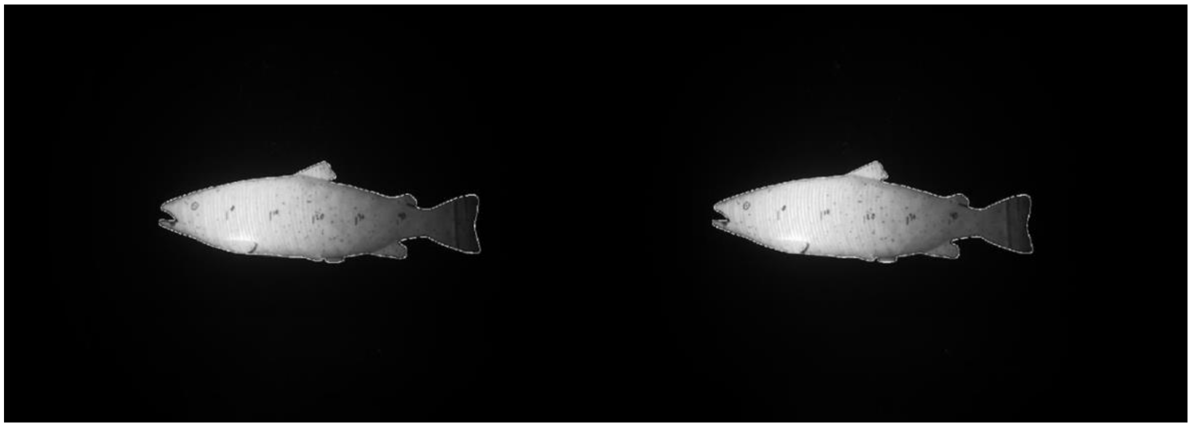


Figure 32: Imposing the border on the grayscale images of the fish and checking if it fits.

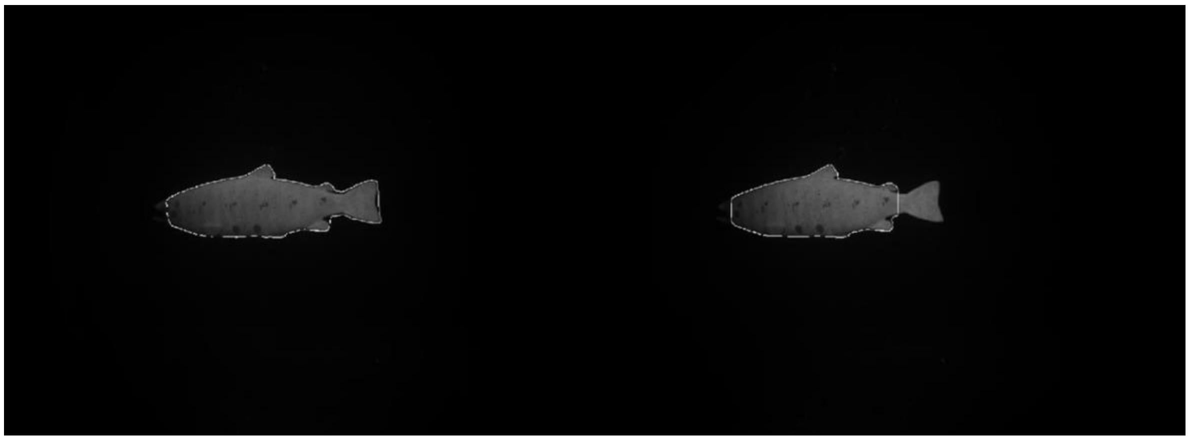


Figure 33: An example where the binary image did not fit the fish.

3.6 Cropping the fish' images

In order to reduce training time and memory usage of the GPU, the fish images were cropped to a smaller size. The requirements for the crop will be that the disparity ranges stay as is, in addition to having as many fish related pixels as possible available for stereo matching. As the binary images define the pixels that belong to the fish, they will be used to find the crop region. Two separate algorithms were created to define the crop regions parameters. The script executing the cropping algorithm may be found in section 10.2.4.

3.6.1 Finding the height of the crop region

Finding the height of the crop region is slightly easier than finding its width. The images have been rectified, consequently the same point on the fish in both left and right image can be found along their respective x axes. Thus, the maximum height found will not lead to data loss directly unless the rectification failed. However, the stereo matching algorithms mentioned in this report rely on a square sized window imposed on a pixel to use the neighboring pixels in both x and y direction to describe it. Consequently, an offset must be added to the maximum height found. The offset, o_{crop}, was chosen to be 36, as the largest number of neighboring pixels in a given direction used by any of the stereo matching methods is 18. When accounting for the opposite direction, offset must be double that.

The crop height, H_{crop}, was found through the following procedure. Let I_h denote the image height.

- Iterate through each row of the fish' left and right binary image from top to bottom until the row contains a pixel with value 1. Define the row numbers as y_L^{min} and y_R^{min}.
- Repeat step 1 except iterate from bottom to top. Define the row numbers as y_L^{max} and y_R^{max}.
- Calculate the crop region's height, H_{crop}, and top left y-coordinate, Y^{min}, from the equations below.

Let Y_L^{min} and Y_R^{min} be candidates to the smallest y-coordinate and Y_L^{max} and Y_R^{max} be candidates to the largest y-coordinate.

$$Y_L^{min} = y_L^{min} - \frac{o_{crop}}{2} \quad 45$$

$$Y_R^{min} = y_R^{min} - \frac{o_{crop}}{2} \quad 46$$

$$Y_L^{max} = y_L^{max} + \frac{o_{crop}}{2} \quad 47$$

$$Y_R^{max} = y_R^{max} + \frac{o_{crop}}{2} \quad 48$$

Let Y^{min} be one of the candidates for smallest x-coordinate or 1 and Y^{max} be one of the candidates for largest x-coordinate or I_h given the conditions.

$$Y^{min} = \begin{cases} \min\left(Y_L^{min}, Y_R^{min}\right) & if \;\; Y_L^{min} > 1 \wedge Y_R^{min} > 1 \\ Y_L^{min} & if \;\; Y_L^{min} > 1 \wedge Y_R^{min} < 1 \\ Y_R^{min} & if \;\; Y_L^{min} < 1 \wedge Y_R^{min} > 1 \\ 1 & otherwise \end{cases} \quad 49$$

$$Y^{max} = \begin{cases} \max(Y_L^{max}, Y_R^{max}) & if \;\; Y_L^{max} < I_h \wedge Y_R^{max} < I_h \\ Y_L^{max} & if \;\; Y_L^{max} < I_h \wedge Y_R^{max} > I_h \\ Y_R^{max} & if \;\; Y_L^{max} > I_h \wedge Y_R^{max} < I_h \\ I_h & otherwise \end{cases} \quad 50$$

Let H_{crop} denote the height of the crop region.

$$H_{crop} = Y^{max} - Y^{min} \quad 51$$

3.6.2 Finding the width of the crop region

When finding the width of the crop region, one needs to account for the following issues. Firstly, both cropped images need to be of the same size. Secondly, the stereo-matching algorithms used in this report truncates the left edge of the disparity map by the largest value of the chosen disparity range, namely the max disparity or d^{max}. To exemplify, let 75 denote the max disparity value. If one were to crop precisely at the first fish related pixel from the left of the image, the stereo algorithm would truncate 75 of the most left columns in the image by marking the pixels defined by those columns as unreliable. Thirdly, the preprocessing part of the deep learning methods exclude d^{max} columns of pixels from the left and right edges of an image. Consequently, if a fish related pixel is less than d^{max} pixels from one of the edges of the image, it is not included in the training data which is of disadvantage as data gets lost if not accounted for. Lastly, one needs to account for the square window size issue mentioned in the section above about finding the height of the crop region.

The width of the crop region was found through the following procedure. Let I_w denote the image width and d^{max} denote the maximum disparity.

- Iterate through each column of the fish' left and right binary image from left to right until the column contains a pixel with value 1. Define the column numbers as x_L^{min} and x_R^{min} respectively.
- Repeat the step above except iterate from right to left. Define the column numbers as x_L^{max} and x_R^{max} respectively.
- Find an estimate to d^{max} by plotting the stereo anaglyph of left and right image pairs where the fish is close to the cameras and measure the disparity between points believed to be the closest to the cameras. Add an error margin of 10 to d^{max} in case the true d^{max} was missed. d^{max} was found to be 170.
- Calculate the crop region's width, W_{crop}, and top left x-coordinate, X^{min}, from the equations below.

Let X_L^{min} and X_R^{min} be candidates to the smallest x-coordinate and X_L^{max} and X_R^{max} be candidates to the largest x-coordinate.

$$X_L^{min} = x_L^{min} - \frac{o_{crop}}{2} - d^{max} \tag{52}$$

$$X_R^{min} = x_R^{min} - \frac{o_{crop}}{2} - d^{max} \tag{53}$$

$$X_L^{max} = x_L^{max} + \frac{o_{crop}}{2} + d^{max} \tag{54}$$

$$X_R^{max} = x_R^{max} + \frac{o_{crop}}{2} + d^{max} \tag{55}$$

Let X^{min} be one of the candidates for smallest x-coordinate or 1 and X^{max} be one of the candidates for largest x-coordinate or I_w given the conditions.

$$X^{min} = \begin{cases} \min\left(X_L^{min}, X_R^{min}\right) & if \;\; X_L^{min} > 1 \wedge X_R^{min} > 1 \\ X_L^{min} & if \;\; X_L^{min} > 1 \wedge X_R^{min} < 1 \\ X_R^{min} & if \;\; X_L^{min} < 1 \wedge X_R^{min} > 1 \\ 1 & otherwise \end{cases} \tag{56}$$

$$X^{max} = \begin{cases} \max(X_L^{max}, X_R^{max}) & if \;\; X_L^{max} < I_w \wedge X_R^{max} < I_w \\ X_L^{max} & if \;\; X_L^{max} < I_w \wedge X_R^{max} > I_w \\ X_R^{max} & if \;\; X_L^{max} > I_w \wedge X_R^{max} < I_w \\ I_w & otherwise \end{cases} \tag{57}$$

Let W_{crop} denote the width of the crop region.

$$W_{crop} = X^{max} - X^{min} \tag{58}$$

Finally, the images I_L and I_R, X^{min}, Y^{min}, W_{crop} and H_{crop} were passed to MATLAB's "imcrop" function to complete the cropping of the images.

3.7 Details of Optoscale's stereo-matching algorithm

The following section describes the details of Optoscale's stereo-matching algorithm, with permission from Optoscale to do so. The core of Optoscale's algorithm is MATLAB's Semi-Global Block Matching method. The method is run independently on every channel of the color images to produce three disparity maps of the same scene. To concatenate the three disparity maps into one, the median between the three estimates is chosen. The script implementing it may be found in section 10.2.5.

3.8 Optimizing Optoscale's algorithm used to find disparity maps

The parameters in Table 1 were used in the optimized algorithm. They were found through trial and error by controlling the disparities found by the optimized algorithm as a whole. "Distance threshold" was deactivated as the dataset created does not include occlusions.

Parameter	**Value**
Method	SemiGlobal
Disparity range	See chapter 3.8.1
Block size	11
Contrast threshold	0.5
Uniqueness threshold	1
Distance threshold	Disabled

Table 1: Parameters of MATLAB's SGBM used when optimizing Optoscale's method.

3.8.1 Setting the disparity range manually by measuring the smallest and largest disparity

In order to make the matches as accurate as possible, the disparity range was found manually to make it as small as possible. The process is summarized by the workflow diagram in Figure 34, and the script implementing it may be found in section 10.2.6.

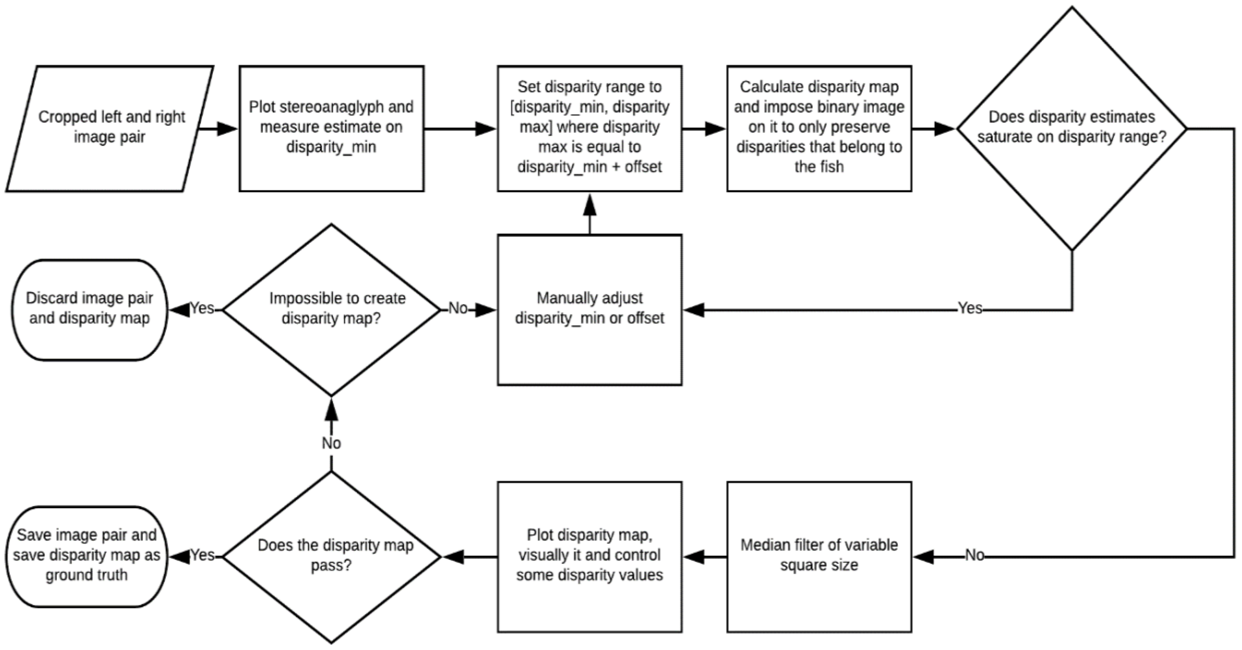

Cropped left and right image pair
Plot stereoanaglyph and measure estimate on disparity_min
Set disparity range to [disparity_min, disparity max] where disparity max is equal to disparity_min + offset
Calculate disparity map and impose binary image on it to only preserve disparities that belong to the fish
Does disparity estimates saturate on disparity range?
Discard image pair and disparity map
Yes
Impossible to create disparity map?
No
Manually adjust disparity_min or offset
Yes
No
Save image pair and save disparity map as ground truth
Yes
Does the disparity map pass?
Plot disparity map, visually it and control some disparity values
Median filter of variable square size
No

Figure 34: Workflow diagram depicting the process used to find the disparity range of each image pair.

The stereo anaglyph of the right and left image was plotted in MATLAB and manual measurements were taken from spots on the fish as seen in Figure 35 to find an estimate of the smallest disparity present, d^{min}. The maximum disparity was set to $d^{min} + offset$. As stated on MATLAB's disparity documentation page (Mathworks, n.d. d), the difference between d^{min} and d^{max}, $offset$, must be divisible by 16. Thus, $offset$ was defined as $16 * n$, where n is a parameter that can be adjusted and has default value 1. Finally, the disparity range was set to $[d^{min}, d^{max}]$. Before arguing why this method detects wrong matches, it should be noted that the binary image was imposed on the disparity map, and so the method only checks disparity estimates within the fish body. This method weeds out wrong matches due to the following reasons. Firstly, if the disparity range is slightly off, the best match found is most likely the closest pixel to the real one as their neighborhoods overlap, however the true match is outside the disparity range. As a result, the disparity found will saturate on either d^{min} or d^{max}. Secondly, roughly 100,000 pixels are matched per image pair, and if the disparity range is completely off, then the probability of all those matches being within the disparity range is low as the true match is not present. Consequently, by controlling that disparity values have not saturated, one can be more confident in the matches found.

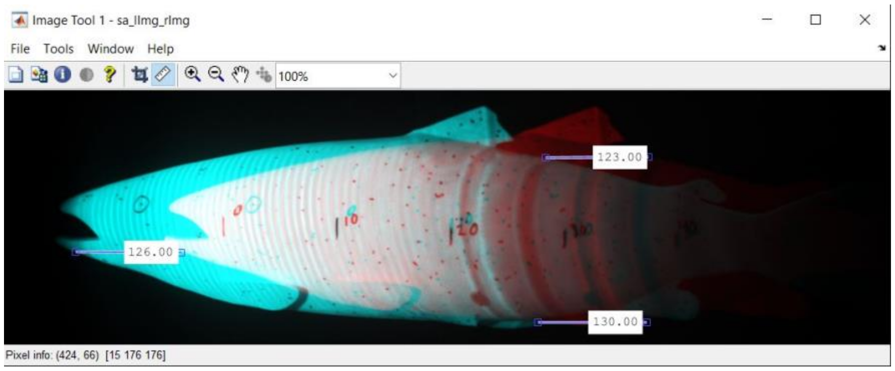


Image Tool 1 - sa_limg_rimg
File Tools Window Help
100%
123.00
126.00
130.00
Pixel info: (424, 66) [15 176 176]

Figure 35: Measuring the smallest disparity through the stereo anaglyph of the image pair.

The method has two parameters, namely d^{min} and n. These had to be tuned for each image. The parameters were tuned according to the rules found in Table 2.

Situation	**Rules**
First iteration	d^{min}: set to estimated minimal disparity. n: set to 1
Estimated disparities saturate only on d^{max}	d^{min}: increase slightly by 1 or 2 n: unchanged.
Estimated disparities saturate only on d^{min}	d^{min}: decrease slightly by 1 or 2

	n: unchanged
Estimated disparities saturate in one direction, and increasing or decreasing d^{min} by 1 leads to saturation in the other direction.	d^{min}: decrease until values only saturate on d^{max}. n: increase by 1
Estimated disparities have saturated in both directions.	d^{min}: decrease until values only saturate on d^{max}. n: increase by 1.
Estimated disparities do not saturate on the disparity range.	End of tuning.

Table 2: Tuning scheme of the two parameters n and d^{min}.

The value of n was not increased on any image pair, and as a result the largest difference between d^{min} and d^{max} was 16.

3.8.2 Removing disparity estimates that do not belong to the fish

MATLAB’s SGBM is a dense stereo matcher, and so it attempts to provide as many matches as possible within the disparity range. We are only interested in the pixels that belong to the fish, and not the background. As a result, the binary image is in this stage imposed on the estimated disparity map to extract only the disparity estimates that belong to the fish. Figure 36 shows the disparity map before the binary image is imposed, and Figure 37 after. Note that the change in color comes from the change of scale in “Disparity colors” as many disparity estimates have been removed.

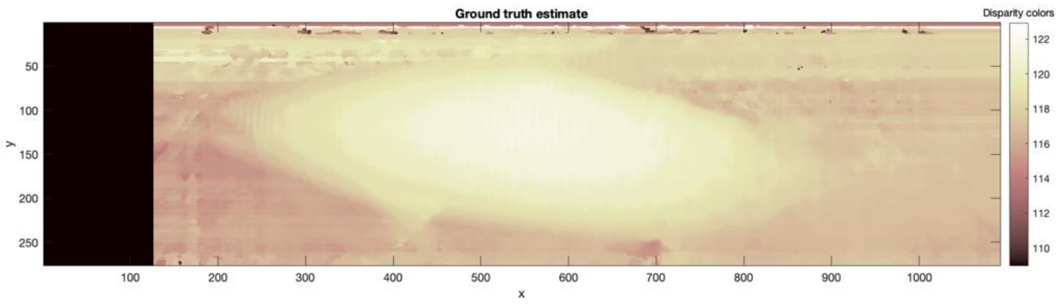


Figure 36: Disparity map before imposing the binary image.

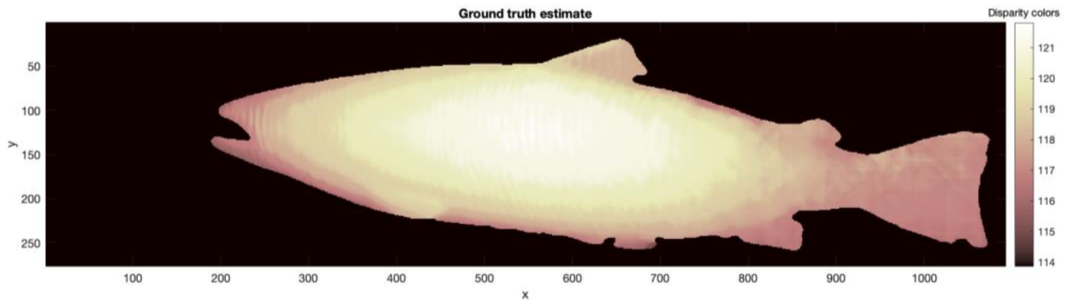


Figure 37: Disparity map after the binary image was imposed.

3.8.3 Median filtering to remove outliers and periodic patterns

Following the tuning phase, a square median filter was used to remove extremals within a local area of the fish and periodic changes in the disparity. Due to the shape of the fish, one would not expect altering values of disparity in for instance the midst of the fish body and close to its snout, or large differences in disparity in a small local area. Thus, a median filter of variable size depending on the size of the fish in the image was introduced. If the fish appeared to be large the filter size was chosen to be 13x13 and this size was gradually reduced down to 5x5 for the smallest. Figure 38 shows an example of altering disparities in areas such as the midst of the fish and a small area of disparities along the back of the fish which are correct, and Figure 39 shows the results of median filtering. We also notice that smaller areas with discontinuous disparities are kept along the borders of the fish.

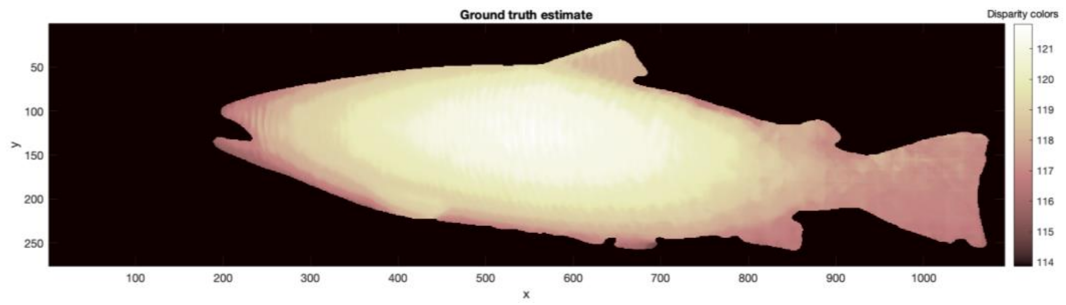


Figure 38: Disparity map before applying the median filter.

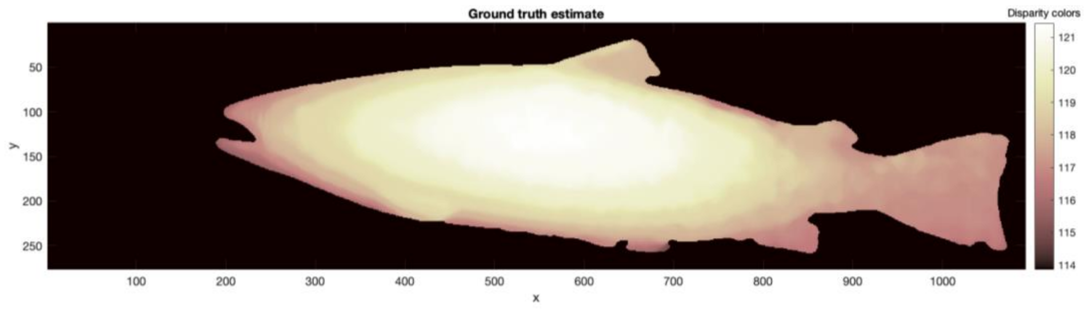


Figure 39: The result of the median filter's operations.

3.8.4 Visually controlling disparity estimates

Following the filter operation, the disparity map was plotted as seen in Figure 39 above in order to look for abnormal disparity values. If any abnormal disparities were detected and it was deemed that those could be fixed, the process of creating the disparity map was repeated. Elsewise the image pair and ground truth were discarded. Furthermore, once pleased with the outlook of the disparity map, the user can interactively mark recognizable spots on the left image, for instance the reference bars in its midst as shown in Figure 40, and then the stereo anaglyph of both images would be plotted in addition to a line representing the disparity as illustrated in Figure 41. If it was difficult to see whether the disparity matched or not as seen at the fish' tail part in the figure, the image would still be kept if the ground truth looked sound and all other points observably looked correct. If the line's start and end point coincided with the same spots in the left and right image, the disparity map and its respective image pair was saved and accepted as ground truth. It is recommended to perform this stage on a rather large monitor with high resolution, as on smaller screens with lower resolution it becomes harder to see darker areas of the fish. The script implementing the method may be found in section 10.2.7.

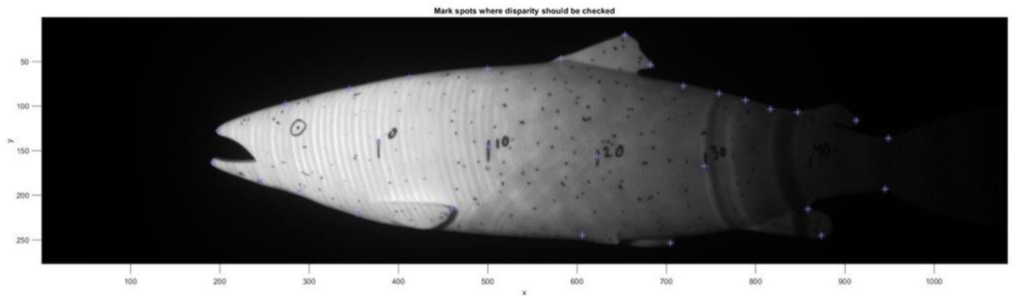


Figure 40: Interactively marking spots were the estimated disparity should be checked.

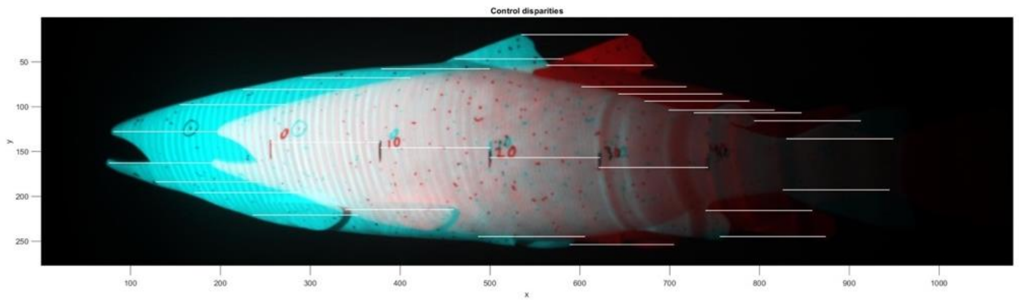


Figure 41: The image visualizing disparities between the left (red) image and the right (blue) image.

3.8.5 Deciding on error metrics for the dataset

The accuracy of the dataset has been empirically controlled and is not bounded by mathematical proof. Consequently, the precedent set by KITTI 2015 Stereo Evaluation, who also empirically controlled their disparity values (Menze & Geiger, 2015, p.5), will be followed. They chose the 3-pixel error, which is deemed as a fair metric to use given the level of accuracy the empirical controls used in this paper give.

Furthermore, it is interesting to see how close the methods predictions come to the values of the ground truth, and so a second error metric will be used. Thus, the second error metric is the 1-pixel error.

Finally, pixels have not been annotated into categories, and so the accuracies we find will be related to the “all” category. In other words, we will only consider entries in the ground truth disparity map that have a value larger than zero when calculating a prediction’s accuracy.

3.9 Deep Learning stereo-matching algorithms

3.9.1 Deciding which Deep-Learning stereo-matching algorithms to test

The code repositories of the methods found in Table 12 were downloaded and tested. In the end, MC-CNN-fst, and Content-CNN were the implementations the author managed to get working after quarrelling with Linux Ubuntu.

Due to Content-CNN’s code repository not containing the scripts used in post-processing, and the removal of post-processing from MC-CNN-fst in section 3.9.5, the results obtained in this paper will not reflect the results the full version of these methods would have shown. As a result, the methods without postprocessing will from here on be called “Content-CNN-nopost” and “MC-CNN-fst-nopost”, to not misguide the reader into thinking that the results were obtained using the full version of the methods.

3.9.2 Creating a dataset for training, validation and testing

The image pairs and their respective ground truths are now available to be trained on, however they need preprocessing. First off, the images were stored in the “png” format and the ground truth was converted from a matrix of type double to an image matrix of type uint16, as both deep learning implementations use the “png16” format for the ground truth, and subsequently saved to “png”. Furthermore, the images had to be divided into a training, validation and testing dataset. All 469 images were loaded into a struct array which was randomized in order to remove any bias present in the dataset, which there was in this case is as the images were taken from a video and so image 0 and image 1 are in close relation to each other before randomizing.

Furthermore, it was noticed that when in trial runs on Content-CNN-nopost that a batch size of 128, max epoch of 50, and 50 iteration per epoch would mean the method would be able to process $128 * 50 * 50 = 320\ 000$ pairs of image patches. Each image pair in the dataset contains about 60,000 valid patch pairs which means even with 10 images in the dataset, Content-CNN would not be able to process them all. Moreover, the network only validates on about 20,000 image pairs chosen at random from the validation set due to memory consumption. When it comes to MC-CNN-fst, it trains on every patch pair in the training set per epoch, and validates on all patch pairs in the validation set. Consequently, the division shown in Table 3 was used, were the number of valid pairs was extracted from the preprocessing script of Content-CNN-nopost when setting “psz” to 18.

Dataset	Number of images	Number of valid patch pairs
Training	10	632 697
Validation	5	325 363
Testing	454	31 841 774

Table 3: Division of samples into training, validation and testing sets

The division serves as a compromise between the data Content-CNN-nopost and MC-CNN-fst-nopost can handle. The number of valid patch pairs in the training set is about double what the Content-CNN is able to process given the hyperparameters mentioned to enable testing with larger epochs and have some diversity in the samples for MC-CNN-fst-nopost. Furthermore, the number of validation patch pairs certainly accommodates Content-CNN-nopost’s demand of only 20,000 patch pairs while still having more pairs available for MC-CNN-fst-nopost which evaluates on all of them.

Finally, the images were stored and named according to the format used by the KITTI 2015 Stereo Evaluation as both implementations already have code that supports this. In essence, the format consists of storing left images in the folder “image_2” and right images in the folder “image_3”. Finally, the ground truth disparity map is stored in the folder “disp_noc_0”. Naming wise, the first picture has a six-digit ID equal to “000000_10” and the following image gets the ID “000001_10”, where the former part of the ID has been incremented by 1.

3.9.3 Preprocessing script for Content-CNN-nopost

The original script present in the code repository of the implementation was edited to suit the new dataset. The script randomized the images before separating them into the training and validation set, and since we already have randomized the images, this feature was removed. The parameter "half_range" which defines the disparity range as $[-half\_range, half\_range]$ was set to 170 and "psz" which defines the patch size was set to 18 when the filters spatial extent was 5, and 9 when the spatial extent was 3. Additionally, the script only includes patches it deems valid. A valid patch means that the x-coordinate of the center pixel of the patch minus the "half_range" and "psz", or plus, does not exceed the bounds of the image the patch is extracted from. The script is included in the zip-file that is described in section 10.3.

3.9.4 Preprocessing script for MC-CNN-fst

The original script present in the code repository of the implementation was edited to suit the new dataset. The script contained code which cropped an image's width to its height. Our images are of varying size, and as a result this feature was removed, and the script edited to accommodate images of different sizes. Furthermore, the script converted images to grayscale. This function was removed, and the grayscale version of the images shown earlier in the method, in Figure 16, was used in its place. The script is included in the zip-file that is described in section 10.3.

3.9.5 Changes made to MC-CNN-fst

The following aspects of the original MC-CNN-fst code were removed.

1. Postprocessing steps. When using the postprocessing steps with the dataset, the disparity maps would look wrong and the shape of the fish not captured. Consequently, these steps were removed after attempting to fix them. Additionally, the parameter deciding the range of the positive sample s_+ was removed, since the theory mentioned that it was introduced to give better postprocessing results.

2. Evaluation method. After the training had finished, the network would load validation samples and output the accuracy on them, and while training it showed the loss on the training samples at each epoch. Such information does not tell us how well the network will generalize to new samples, and we have no way of knowing when the network begins to overfit on the training data. Consequently, the evaluation method was removed and replaced with another evaluation method which is described below.

The following additions were added to MC-CNN-fst's code.

1. Evaluation method. Instead of considering the network's loss, the script was edited to show the average accuracy over the training samples and the average accuracy over the validation samples at every epoch. Such a scheme makes it possible to evaluate whether the network is becoming better at predicting the ground truth, if it is overfitting on the training data and when it is optimal to stop the training. The evaluation method was implemented by moving and adapting code already present in the "main.lua" file of MC-CNN-fst.
2. Logging method. MC-CNN-fst outputted the evaluation metrics in the console and did not store them in a convenient way for documentation. Consequently, code was borrowed from the Content CNN's code repository to allow the logging and plotting of training and validation accuracy at each epoch.
3. Learning rate scheme. In the original code, the learning rate would we lowered at the fixed epoch of 12. Such a scheme leaves little leeway in adjusting the learning rate as the network trains, and so a patience parameter was introduced allowing for the network to scale its learning rate at every epoch specified in the parameter.
4. A parameter was added to choose the number of epochs the network should train for. Finally, the original code had a fixed epoch of 14. Leaving little room to investigate the effects of training over more epochs, and so a parameter was introduced that specifies the epochs the number of epochs the network should train over.

The script is included in the zip-file that is described in section 10.3.

3.9.6 Learning rate scheme, number of epochs to train for and related hyperparameters

The learning rate scheme was chosen by considering the plots of training and validation accuracy as seen in Figure 42 of the networks when using batch size 128.

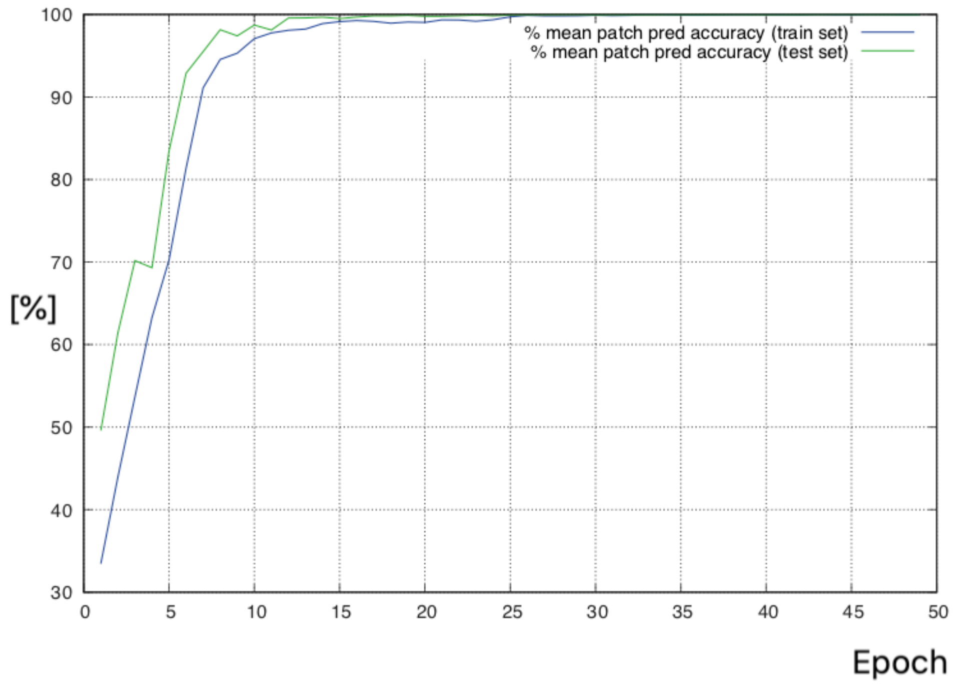


Figure 42: Mean training and validation patch accuracy per epoch plot from Content-CNN-nopost.

If the graph showed stable convergence towards a high mean accuracy, the learning rate scheme would be kept. If it showed unstable characteristics, or converged to lower mean accuracy, the learning rate scheme would be changed. If the training ended while the graphs had not flattened out, the number of epochs to train for would be increased. In the end, the max epoch was chosen to be 50 for both MC-CNN-fst-nopost and Content-CNN-nopost. The learning rate scheme used for MC-CNN-fst-nopost may be seen in Table 4, and the scheme used for Content-CNN-nopost may be seen in Table 5.

Epochs	**Learning rate**
1-14	0.002
15-29	0.001
30-44	0.0005
45-50	0.00025

Table 4: MC-CNN-fst-nopost's learning rate scheme.

Epochs	**Learning rate**
1-24	0.02
25-34	0.004
35-44	0.0008
45-50	0.00016

Table 5: Content-CNN-nopost's learning scheme.

Additionally, Content-CNN-nopost was trained with the learning rate optimizer "adam". Its parameters were chosen to be the standard parameters shipped with the implementation, and may be seen in Table 6.

Parameter	**Value**
Weight decay	5e-4
Momentum	0.9
Learning rate decay	1e-7

Table 6: Parameters used for the "adam" learning rate optimizer in Content-CNN-nopost.

4 Experiments

All experiments were conducted in Linux Ubuntu 16.04 with the GPU Nvidia GTX 1080 Ti. Furthermore, MATLAB R2018b, Torch 7, CUDA 9.0 and cudNN 7.0.5 were the software, libraries and drivers used.

4.1 Accuracy definitions

4.1.1 Optoscale's original algorithm

The "Mean test accuracy" was the mean accuracy over all valid disparity values in the ground truth disparity maps. In these results, the 3-pixel and 1-pixel error was used to determine if a disparity estimate was correct. The script used to perform the evaluation of Optoscale's algorithm may be found in section 10.2.8.

4.1.2 MC-CNN-fst-nopost

The "Mean training accuracy", "Mean validation accuracy" and "Mean test accuracy" of MC-CNN-fst-nopost was the mean accuracy over all valid disparity values in the ground truth disparity maps of the training, validation and testing dataset respectively. For the training and validation results, the 3-pixel error was used to determine if a disparity estimate was correct or not, while on the test accuracy both the 3-pixel and the 1-pixel was used. The scripts used to perform the evaluation of MC-CNN-fst-nopost may be found in the attached zip-file that is described in section 10.3.

4.1.3 Content-CNN-nopost

The "Mean training accuracy" was the mean accuracy over a subset of the training set processed at the epoch the weights were extracted, the network processed 320,000 image pairs per epoch. "Mean validation accuracy" was the mean accuracy on 20,000 patch pairs from the validation dataset chosen at random when the training was started. The accuracies on the training and validation used the 3-pixel error to determine if a disparity estimate was correct or not. Finally, "Mean test accuracy" was the mean accuracy over all ground truth pixels in the test dataset. In these results, both the 3-pixel and 1-pixel error was used. The scripts used to perform the evaluation of Content-CNN-nopost may be found in the attached zip-file that is described in section 10.3.

4.2 Implementation details

4.2.1 Optoscale's algorithm

Table 7 shows the parameters Optoscale's algorithm uses.

Parameter	**Value**
Method	SemiGlobal
Disparity range	[65,145]
Block size	11
Contrast threshold	0.5
Uniqueness threshold	15
Distance threshold	Disabled

Table 7: Parameters of MATLAB's SGBM used by Optoscale.

4.2.2 MC-CNN-fst-nopost

Table 7 shows the general aspects of the network which were not changed in any experiment.

Aspect	**Value**
Disparity range	[0, 170]
Input data	3-channeled color images
Max number of epochs when training	50
Learning rate scheme	See Table 4
Learning rate optimizer	None
Number of filters in the convolutional layers	All layers contained 64 filters
Filter stride	1
s_+ deviation	0
s_- deviation	[-4,10]
Data augmentation	None
Post-processing	None

Table 8: The general aspects of MC-CNN-nopost's that were held constant during experiments.

4.2.3 Content-CNN-nopost

Table 7 shows the general aspects of the network which were not changed in any experiment.

Aspect	**Value**
Disparity range	[0, 170]
Input data	1-channeled “grayscale” images converted according to section 3.5.3.
Max number of epochs when training	50
Learning rate scheme	See Table 5
Learning rate optimizer	“adam”, see Table 6 for its hyperparameters.
Number of filters in the convolutional layers	The first two layers contained 32 filters, the rest 64
Filter stride	1
Data augmentation	Disabled
Post-processing	None

Table 9: The general aspects of Content-CNN-nopost that were held constant during experiments.

4.3 Comparing Optoscale’s original algorithm and the optimized version

The original algorithm used by Optoscale was run on the test dataset. Its accuracy was recorded and plots illustrating the differences between the two algorithms were made. Additionally, the “uniqueness threshold” of Optoscale’s original algorithm was lowered to the value of the optimized version to see the effects this choice has.

4.4 Testing different batch sizes

4.4.1 MC-CNN-fst-nopost

In these experiments, the batch size was chosen to be 128 and 32 for MC-CNN-fst-nopost as lower batch sizes would lead to the training of the network taking too long.

4.4.2 Content-CNN-nopost

For Content-CNN-nopost, the batch size was chosen to be 128, 32, and 2. Moreover, Content-CNN-nopost only trains and validates on a subset of the data per epoch. As a result, whenever the batch size was lowered for Content-CNN-nopost, the number of iterations per epoch was increased so that the same amount of data would be processed.

4.5 Testing different network structures

4.5.1 MC-CNN-fst-nopost

For MC-CNN-fst-nopost, the following network structures were trained and tested.

1. 9 convolutional layers, and a filter size of 3x3xD yielding a patch size of 19x19.
2. 4 convolutional layers, and a filter size of 3x3xD yielding a patch size of 9x9.

4.5.2 Content-CNN-nopost

For Content-CNN-nopost, the following network structures were trained and tested.

1. 9 convolutional layers, and a filter size of 5x5xD yielding a patch size of 37x37.
2. 4 convolutional layers, and a filter size of 5x5xD yielding a patch size of 17x17.

5 Results

5.1 Methods used by popular stereo contests to create stereo image datasets with ground truth disparity maps

In order for the methods found to be considered, they have to adhere to one of the following criteria. Their papers or webpages must either have been cited by 50 or more sources, not all being the main authors, or have been peer-reviewed or lay the foundation for a popular stereo evaluation contest.

5.1.1 Methods found, criteria validation and code availability

Table 10 sums up the seven methods found which might be used to create a ground truth dataset for stereo matching. The symbol "*" in the abbreviation column symbolizes that no official abbreviation was found and so one was created to better the formatting of tables. Only methods which mentioned an estimate of its accuracy were included.

Abbreviation	**Method's papers**	**Criteria match**	***Accuracy estimate***
TSUSET*	(Martull et al., 2012)	Citations	1 pixel
MIDDLESET*	(Scharstein et al., 2014)	Peer-review, contest	$\frac{1}{5}$ pixel
KITTISET*	(Geiger et al., 2013; Menze & Geiger, 2015)	Citations, contest	3 pixels (empirical)

Table 10: Methods which may be used in the creation of a stereo image dataset with ground truth disparity maps.

5.1.2 The methods' general requirements

Table 11 lists the general requirements of each method found. Said requirements include if any special equipment is needed, whether the setup has any assumptions and so forth.

Abbreviation	**General requirements**
TSUSET	• 3D-modelling of the scene via for instance Autodesk Maya 2012. This includes digital models of objects and cameras
MIDDLESET	• A source of structured light using maximum min-stripe-width gray-codes projecting the code pattern and its inverse • A stationary stereo rig with option to move projector of structured light to different locations and two translatable cameras. • A series of images captured with different projector positions. • Static indoor scene
KITTISET	• A stereo rig with IMU/GPS, LIDAR, 2 grayscale cameras and 2 color cameras.

Table 11: The general requirements of each ground truth disparity map method.

5.2 Deep Learning Stereo Matchers

In order for the methods found to be considered, they have to adhere to the following criteria. Their papers or webpages must either have been cited by 50 or more sources, not all being the main authors, or have been peer-reviewed. In addition, the implementations must have code available.

5.2.1 Methods found, criteria validation and code availability

In Table 12 the relevant information for each implementation found has been summarized. The first column indicates the paper of the implementation and its code repository. Sub-columns of the “**Code**” column indicates whether the code repository contains code for training (Train), inferring (Infer), preprocessing (Prep), data augmentation (Aug) and post-processing/disparity refinement (Post). Note that Table 12 is the result of the author’s interpretation of the code and its corresponding paper, and as a result some details of the implementations may have been missed. Additionally, as of March 2019, the code repository of Content-CNN has been unavailable.

Abbreviation	**Method’s paper and code repository**	**Criteria match**	**Code**				
			Train	Infer	Prep	Aug	Post
Content-CNN	(Luo et al., 2016; Luo et al., n.d.)	Citations	Yes	Yes	Yes	No	No
SegStereo	(Yang et al., 2018; Yang et al., n.d.)	Peer-review	No	Yes	No	No	Yes
SCV	(Lu et al., 2018; Lu et al., n.d.)	Peer-review	Yes	Yes	Yes	Yes	No
CRL	(Pang et al., 2017; Pang et al., n.d.)	Citations	No	Yes	No	No	Yes
MC-CNN-ACRT/ MC-CNN-fst	(Žbontar & LeCun, 2016; Žbontar & LeCun, n.d.)	Peer-review	Yes	Yes	Yes	Yes	Yes

Table 12: The Deep Learning stereo-matching methods found, and the extent of their code repositories.

5.2.2 The methods' accuracy and performance results

Table 13 summarizes the accuracy results, and which GPU was used when the performance runtime was recorded. Whenever a column contains values such as "67/0.8" it represents that the implementation has two variations. Thus, the former value belongs to the first variation, and the latter to the second. The "**Error all (KITTI 2015)**" is the "D1-all" metric used by the KITTI 2015 stereo evaluation (Geiger et al., 2015b). It is the average of erroneous disparity estimates over all ground truth pixels on the KITTI 2015 stereo image dataset. In this case, an erroneous disparity estimate is within 3 pixels of the ground truth.

Abbreviation	**GPU used**	**Runtime**	**Error all (KITTI 2015)**
Content-CNN	Nvidia GTX Titan X	1 second	4.54 %
SegStereo	Nvidia GTX Titan Xp	0.6 seconds	2.25 %
SCV	Nvidia GTX 1080 Ti	0.36 seconds	2.61 %
CRL	Nvidia GTX 1080	0.47 seconds	2.67 %
MC-CNN-ACRT/ MC-CNN-FST	Nvidia GTX Titan X	67/0.8 seconds	3.89/4.62 %

Table 13: Performance details related to each Deep Learning stereo-matching method found.

Table 14 below summarizes the requirements needed to run the implementations with GPU-accelerated computation. “**OS”** is short for operating system and “**GPU**” for graphical processing unit. Whenever an entry contains “…”, the author did not find information on the topic.

Abbreviation	**OS**	**GPU, GPU memory**	**Frameworks, libraries and drivers**	
			GPU specific	Miscellaneous
Content-CNN	Linux Ubuntu 12+ 64 bit or MacOS	Nvidia GPU, …	CUDA 8+, cuDNN v5+	Torch 7
SegStereo	Linux Ubuntu 14+ 64 bit or MacOS	Nvidia GPU, …	CUDA 8, cuDNN v5.1	Caffe, FlowNet 2.0, PSPNet, OpticalFlowToolkit
SCV	Linux Ubuntu … or MacOS	Nvidia GPU, 6 GB or more memory	Cuda 7.5+, cuDNN	Python 3.6, PyTorch 0.3.0, torchvision 0.1.8,
CRL	Linux Ubuntu 12+ or MacOS	Nvidia GPU, …	Cuda 7+, cuDNN v6	MATLAB R2015a, Matcaffe, BLAS, Boost 1.55+
MC-CNN-ACRT/ MC-CNN-FAST	Linux Ubuntu 12+ or MacOS	Nvidia GPU, 6 GB or more memory/ NVIDIA GPU, …	Cuda 8+, cuDNN v5+	Torch 7, OpenCV 2.4+, png++

Table 14: Requirements needed to run the implementation of each Deep Learning stereo-matching algorithm.

5.3 Details of the stereo image dataset

5.3.1 Accepted image pairs, smallest and largest disparity

Table 15 summarizes some important aspects of the final stereo image dataset, such as how many image pair and corresponding ground truth disparity maps were accepted and discarded, and the smallest and largest disparity value in the entire dataset.

Aspect	**Value**
Accepted image pairs and disparity maps	469
Discarded image pairs and disparity maps	131
Smallest disparity	66
Largest disparity	132

Table 15: Numbers of the final stereo image dataset.

5.3.2 A subset of image pairs and ground truth disparity maps from the stereo image dataset

Figure 43-Figure 54 illustrate a small subset of the cropped RGB image pairs and their corresponding ground truth disparity maps. Furthermore, the full dataset will not be released as requested by Optoscale who own the rights to it. Black colors in the plotted disparity map denote disparity values equal to zero.

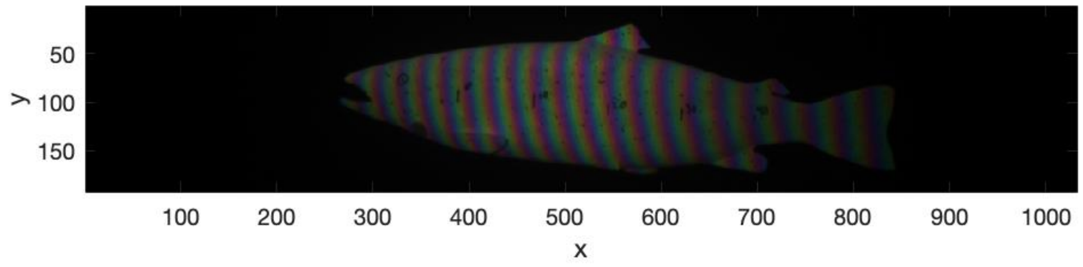


Figure 43: The first example showing an average left image in the stereo image dataset.

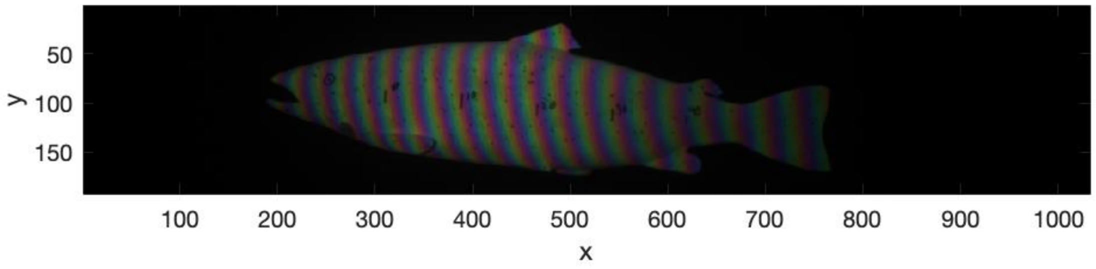


Figure 44: The first example showing an average right image in the stereo image dataset.

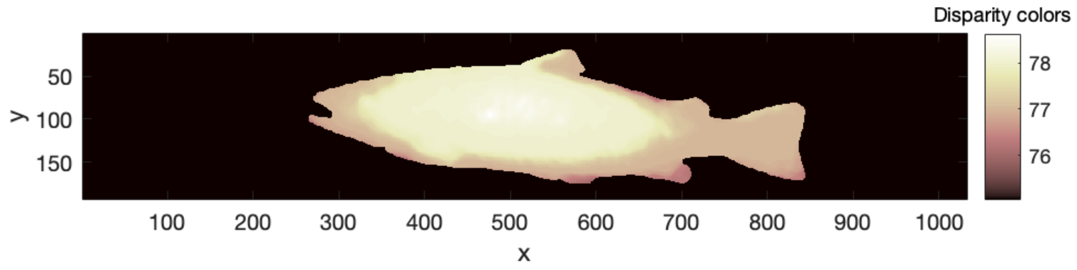


Figure 45: The first example's ground truth disparity map.

Figure 43-Figure 45 shows us that the ground truth disparity map does include small regions belonging to the background. For instance, there is a gap between the fin above the start of the tail which belongs to the background. In the disparity map, this gap has been included and given disparity estimates. Additionally, some parts of the fish' edge look rough.

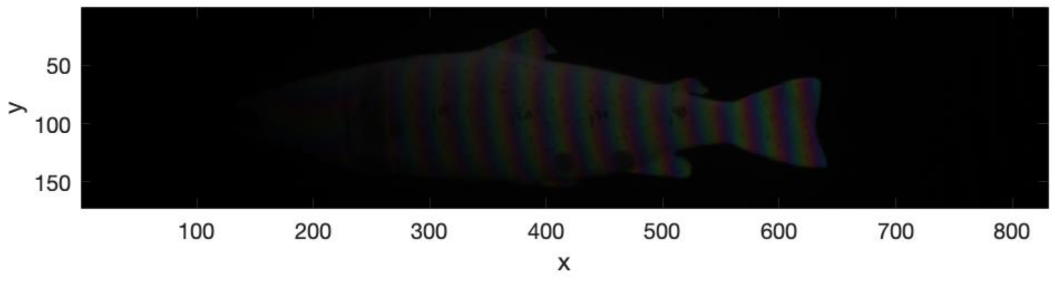


Figure 46: The second example showing a dark left image in the stereo image dataset.

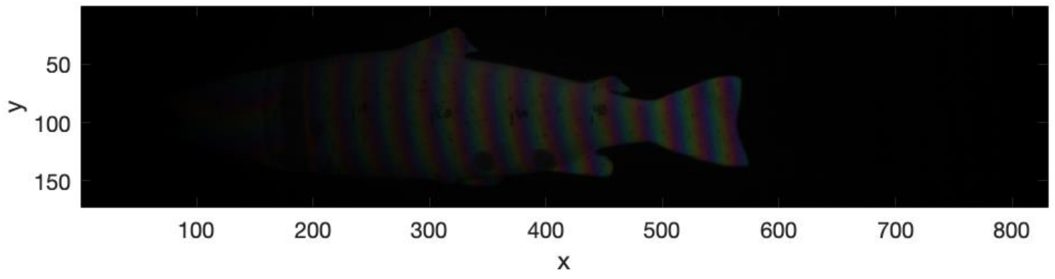


Figure 47: The second example showing a dark right image in the stereo image dataset.

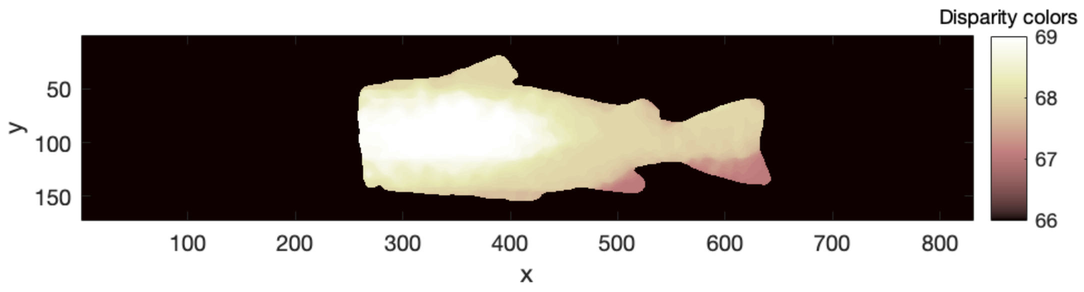


Figure 48: The second example's ground truth disparity map.

Figure 46-Figure 48 shows us that in some cases, the method used to create the stereo image dataset does not include disparity values for all pixels of the fish. Additionally, they illustrate the smaller disparity values present in the stereo image dataset.

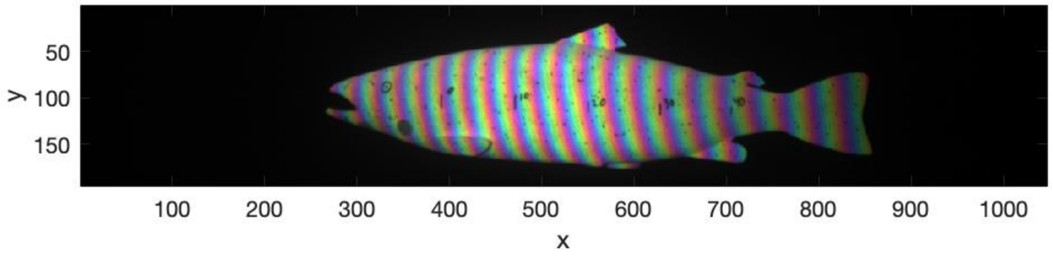


Figure 49: The third example showing a bright left image in the stereo image dataset.

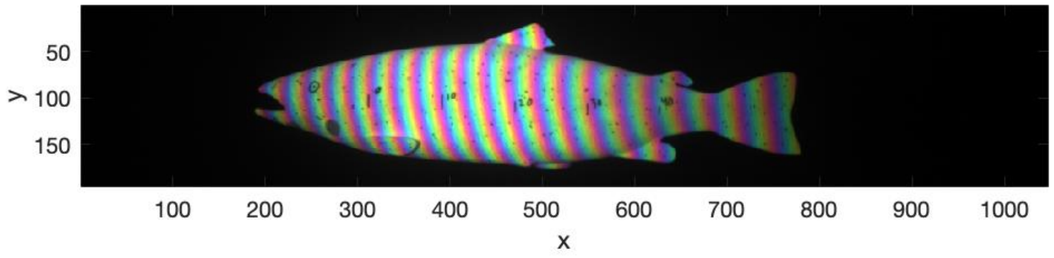


Figure 50: The third example showing a bright right image in the stereo image dataset.

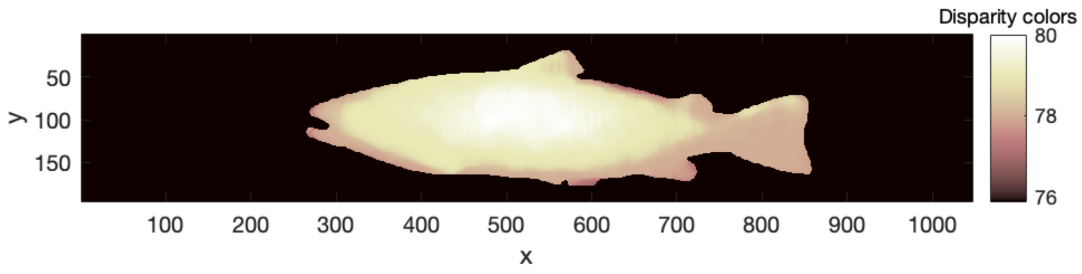


Figure 51: The third example's ground truth disparity map.

Figure 49-Figure 51 shows us that in some cases, the method used to create the stereo image dataset does manage to find disparity estimates for all parts of the fish.

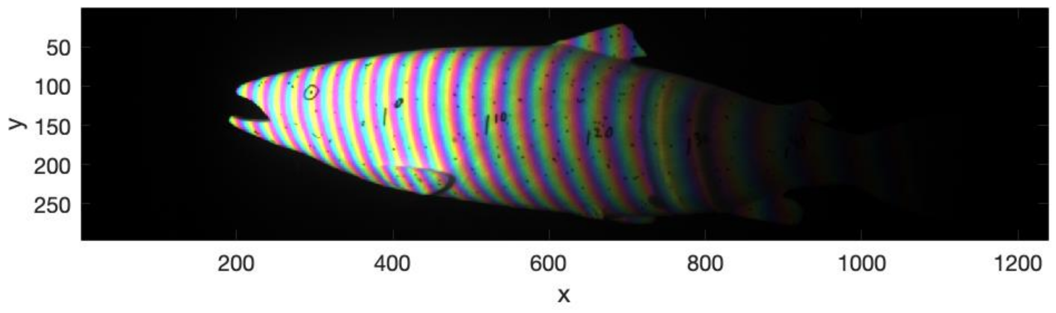


Figure 52: The fourth example showing a bright and dark left image in the stereo image dataset.

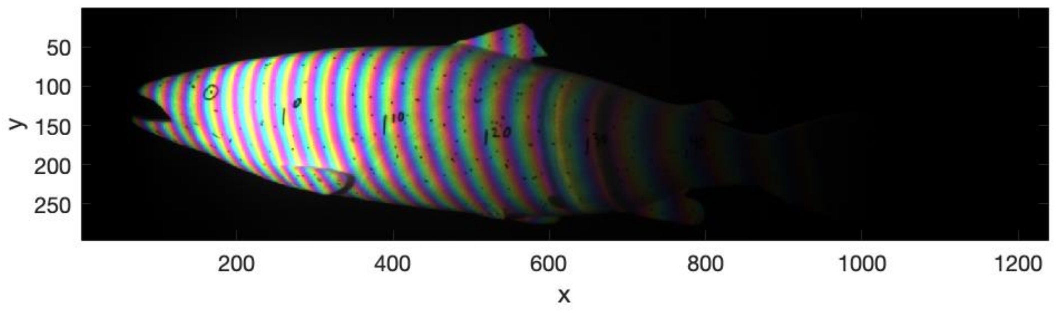


Figure 53: The fourth example showing a bright and dark right image in the stereo image dataset.

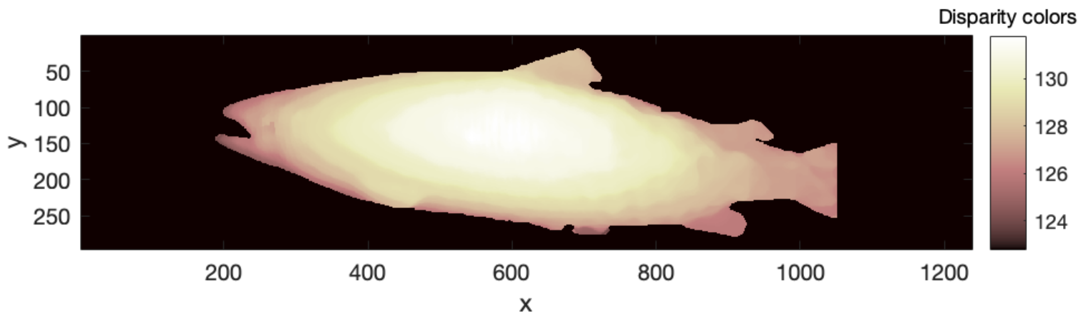


Figure 54: The fourth example's ground truth disparity map.

Figure 52-Figure 54 shows us an example of the larger disparity values present in the stereo image dataset.

5.4 Results of Optoscale's original algorithm on the test dataset

All results below are in regard to how many disparity values in the grounds truths the method managed to predict **correctly**. Furthermore, since stereo-matching algorithms do provide disparity estimates at locations not considered by the ground truth, for instance the background, the masks which define the valid disparity values in ground truths are imposed on every estimated disparity map before calculating the accuracy. The mask was found by thresholding the ground truth disparity maps at a value of 0, i.e. $mask = disparity\ map > 0$, which creates a binary image that may be imposed on the predicted disparity map to remove estimates belonging to the background.

5.4.1 Mean accuracy results over the test set

Table 16 shows the results of the experiments. The "uniqueness threshold" value of 15 belongs to Optoscale's original stereo-matching algorithm, while the value of 1 was introduced to see the effect it had.

"Uniqueness threshold" value	**Mean test accuracy [%]**	
	3-pixel	**1-pixel**
15	92.071	91.990
1	92.411	92.323

Table 16: Mean accuracy of Optoscale's original algorithm over the test set at different error and "uniqueness" thresholds.

We see that Optoscale's algorithm does not replicate the results of the optimized algorithm perfectly. In addition, the mean accuracy increases slightly when changing the value of the "uniqueness threshold" to the one used in the optimized version.

5.4.2 A plot of an erroneous prediction and the corresponding ground truth disparity map

Figure 55 and Figure 56 show the predicted disparity map from Optoscale's algorithm and the corresponding ground truth disparity map.

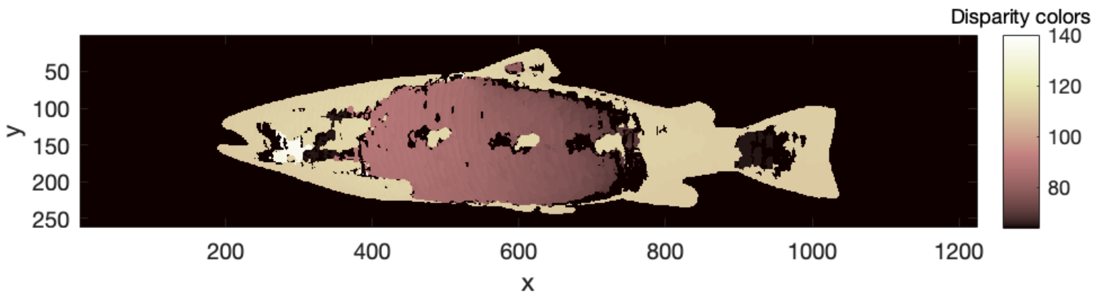


Figure 55: A wrongly predicted disparity map from Optoscale's original algorithm.

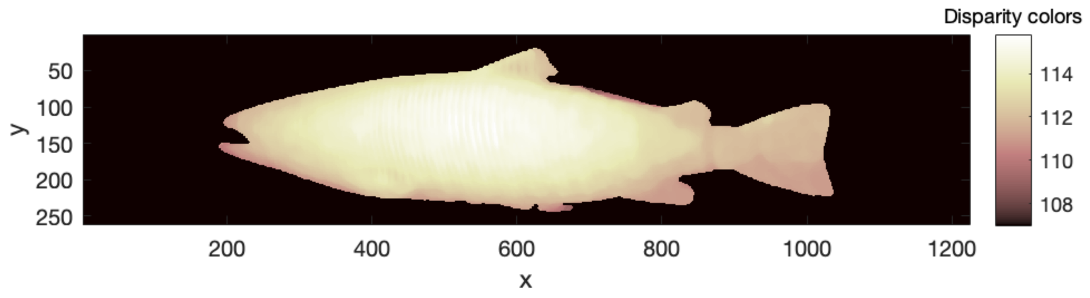


Figure 56: The corresponding ground truth disparity map.

We see that the original algorithm has failed to find all disparities, and that many disparities do not adhere to the known shape of the fish model.

5.5 Accuracy results of MC-CNN-fst-nopost and Content-CNN-nopost

All results below are in regard to how many disparity values in the grounds truths the methods managed to predict **correctly**. Furthermore, since stereo-matching algorithms do provide disparity estimates at locations not considered by the ground truth, for instance the background, the masks which define the valid disparity values in ground truths are imposed on every estimated disparity map before calculating the accuracy. The mask was found by thresholding the ground truth disparity maps at a value of 0, i.e. $mask = disparity\ map > 0$, which creates a binary image.

5.5.1 Results gathered from MC-CNN-fst-nopost

Table 17 shows the relevant information regarding all results gathered from MC-CNN-fst-nopost with 9 convolutional layers, a filter size of 3x3, and a patch size of 19x19. The networks' training and validation accuracy results used the 3-pixel error. The best results have been marked in bold.

Weights from epoch	**Batch size**	**Mean training accuracy [%]**	**Mean validation accuracy [%]**	**Mean test accuracy [%]**	
				3-pixel	**1-pixel**
48	128	99.192	99.400	98.307	90.572
49	32	**99.305**	**99.533**	**98.490**	**91.049**

Table 17: All results from MC-CNN-fst-nopost with 9 convolutional layers and a patch size of 19x19.

We see that for this particular structure of MC-cnn-fst-nopost, with 9 convolutional layers, the best accuracy result within every category was achieved when using a batch size of 32. However, the improvements to accuracy it made are marginal at best.

Furthermore, Table 18 shows the relevant information regarding all results gathered from MC-CNN-fst-nopost with 4 convolutional layers, a filter size of 3x3, and a patch size of 9x9. The networks' training and validation accuracy results used the 3-pixel error. The best results have been marked in bold.

Weights from epoch	**Batch size**	**Mean training accuracy [%]**	**Mean validation accuracy [%]**	**Mean test accuracy [%]**	
				3-pixel	**1-pixel**
47	128	**87.508**	**87.775**	**84.912**	**75.493**
50	32	85.899	86.119	83.091	73.991

Table 18: All results from MC-CNN-fst-nopost with 4 convolutional layers and a patch size of 9x9.

We see that for this particular structure of MC-CNN-fst-nopost, with 4 convolutional layers, the best accuracy result within every category was achieved when using a batch size of 128. This particular choice of batch size did seem to have a noticeable impact, as the result within each category increased by about 1.6-1.9 %.

5.5.2 Results gathered from Content-CNN-nopost

Table 19 shows the relevant information regarding results gathered from Content-CNN-nopost with 9 convolutional layers, a filter size of 5x5, and a patch size of 37x37. The network's training and validation accuracy used the 3-pixel error. The best results have been marked in bold.

Weights from epoch	**Batch size**	**Mean training accuracy [%]**	**Mean validation accuracy [%]**	**Mean test accuracy [%]**	
				3-pixel	**1-pixel**
50	128	99.734	99.780	96.391	95.614
45	32	**99.953**	**99.995**	**99.864**	**99.620**
45	2	24.328	24.410	18.745	11.160

Table 19: All results from Content-CNN-nopost with 9 convolutional layers, 5x5 filters and a patch size of 37x37.

We see that for this particular structure of Content-CNN-nopost, with 9 convolutional layers and a filter size of 5x5, the best accuracy result within every category was achieved when using a batch size of 32, showing significantly better accuracy on the test set.

Moreover, Table 20 shows the relevant information regarding results gathered from Content-CNN-nopost with 4 convolutional layers, a filter size of 5x5, and a patch size of 17x17. The network's training and validation accuracy used the 3-pixel error. The best results have been marked in bold.

Weights from epoch	**Batch size**	**Mean training accuracy [%]**	**Mean validation accuracy [%]**	**Mean test accuracy [%]**	
				3-pixel	**1-pixel**
50	128	97.141	98.630	93.019	91.441
50	32	**98.297**	**99.380**	**96.325**	**94.858**
50	2	20.469	30.235	24.326	18.062

Table 20: Results from Content-CNN-nopost with 4 convolutional layers, 5x5 filters and a patch size of 17x17.

5.5.3 The methods' best results on the test set

Table 21 shows the best results of each method. The best result of MC-CNN-fst-nopost was obtained with a batch size of 32, 9 convolutional layers, 3x3 filter size, and a patch size of 19x19. The best result of Content-CNN-nopost was obtained with a batch size of 32, 9 convolutional layers, 5x5 filter size and a patch size of 37x37. Both methods followed the learning rate schemes shown in section 3.9.6. The best result for each category has been marked in bold.

Method	**Mean test accuracy [%]**	
	3-pixel	**1-pixel**
Optoscale’s algorithm	92.071	91.990
MC-CNN-fst-nopost	98.490	91.049
Content-CNN-nopost	**99.864**	**99.620**

Table 21: Best results of Optoscale's method, MC-CNN-fs-nopost and Content-CNN-nopost.

We see that the particular structure of Content-CNN-nopost out-performed the other methods both when using the 3-pixel and 1-pixel error metric, and that its prediction accuracy is close to 100 % in both error categories. Additionally, both Optoscale’s algorithm and MC-CNN-fst-nopost had significantly lower accuracy than Content-CNN-nopost when using the 1-pixel error, and when using the 3-pixel error, MC-CNN-fst-nopost’s results improved.

6 Discussion

6.1 Methods used in the creation of ground truth disparity maps

6.1.1 Assessing the methods accuracy

The accuracy of each method is also an important aspect to consider. According to the creators of TSUSET, the accuracy is 1 pixel. As the creators are in full control of every pixel in the scene, and its location, this error could simply stem from the fact that pixel locations stored in memory only accepted whole numbers. Furthermore, the MIDDLESET method claims an accuracy of $\frac{1}{5}$ pixel, which is the best accuracy reported by the methods found. Finally, the creators of the benchmark for the KITTISET has "empirically found that for most parts our ground truth is at least 3 pixels accurate" (Menze & Geiger, 2015, p.2). Which is interesting, as there is no mathematical proof of it being that accurate, yet the contest is still highly popular.

5.1.2 Comparing the methods' general requirements

The MIDDLESET dataset relies on a rig which keeps all its parts static at the moment of capture. It also has the assumption that the scene captured is static. Consequently, the method seems to perform at best in a fully controlled environment were all movement is under the control of those capturing the set, and it would require the creation of a stereo camera rig. Furthermore, the KITTISET tackles a dynamic environment where both the rig itself, and the scene can shift by introducing measurement equipment such as a LIDAR and an Inertial Measurement Unit. The method does however rely on several sensors that need to be integrated and does not promise the most reliable accuracy. The TSUSET on the other hand, creates a synthetic dataset and thus effectively avoids the issue environment presents as the entire environment is controlled by the modeler. The method does rely on correctly modelling every object in the scene, which requires knowledge of the software used to do so.

6.2 The search for deep learning stereo-matching algorithms with code implementations

5.2.1 Assessing the implementations' performance and accuracy results

When looking at the accuracy results of each method SegStereo, CRL and SCV seem to be the strongest candidates. Runtime wise, SCV is the fastest however the implementations have been run on different GPUs, and for instance a Nvidia GTX 1080 Ti is more powerful than a Nvidia GTX 1080. Consequently, one has to take the GPU used into account when deciding which implementation has the fastest runtime. Finally, all implementations found have been tested and compared against each other on the KITTI 2015 stereo evaluation dataset. As a result, it is unknown what the results might be when testing them on a different dataset.

5.2.2 Comparing the requirements of the implementations

When it comes to compatible GPUs, all GPU-based implementations share the requirement of the GPU coming from the brand Nvidia. Additionally, two implementations required the GPU's memory to be of size 6 GB or larger. When it comes to frameworks, drivers and libraries a version of CUDA, and cuDNN is required to enable GPU-accelerated computations. All implementations seem to be compatible with CUDA 8+. If one opts to use cuDNN v5, all implementations but CRL become compatible. If one opts to use cuDNN v6, all but SegStereo become compatible. Finally, the OS of choice would be Linux Ubuntu or macOS.

6.3 Consequences of creating the stereo dataset with an optimized version of Optoscale's algorithm

6.3.1 The ground truth disparity map's accuracy can only be determined empirically

When creating the ground truth disparity map with a stereo matcher of unknown accuracy, one will not be able to mathematically prove the accuracy of the disparity maps unless a new method with proven accuracy is used, and the results compared. Consequently, one can only give an estimate of the dataset's accuracy based on empirical observations and assume that most disparity estimates lie within this accuracy. Furthermore, the decision causes implications for the generality of the results found. In essence, the results obtained may not be used in deciding if one stereo-matcher is better than another in general, only on the dataset created given that the chosen 3-pixel error captures the inaccuracies present in the disparity maps.

6.3.2 Optoscale's algorithm will have a natural bias towards the dataset

Optoscale's original algorithm will have a natural bias towards the created dataset since they both share the same stereo-matching algorithm at core. As a result, one can expect Optoscale's algorithm to yield many of the same disparity estimates as those found in the dataset. This should be taken into account when discussing which method got the best accuracy results.

6.4 The subset of images from the stereo image dataset

6.4.1 Example 1

Example 1 showed that the binary images does not fit perfectly on when the disparity is small, the reason being is most likely the size of the median filters used in refining the binary image and the ground truth disparity map. Whenever the depth to the fish increases, the gap which belonged to the background shrinks. As a result, the median filters could have essentially joined the regions belonging to the background with the region denoting the pixels related to the fish. This could have impacted the accuracy of the ground truth disparity maps. However, the areas that are in question are close to the pixels belonging to the fish and have been given disparity values that coincide with the disparity values that do belong to the fish in the same area. This is most likely due to the Semi-Global energy function, which penalizes discontinuities in disparity values. Consequently, we are quite certain that the estimates are mostly correct and note that the method used to finds the binary images does have room for improvement. For instance, by creating a function for the size of the median filter which takes into account the area of the full-sized image the fish covers, which may be estimated by using the binary image outputted after thresholding. The same proposal may be used in deciding the size of the median filter used in refining the disparity map.

6.4.2 Example 2

Example 2 showed that not all disparity maps contain disparity values for every part of the fish, which is most likely caused by the fish not being illuminated by the structured light and the thresholding method used. This tells us one of two things. Firstly, this may illustrate a weakness in the method used to find binary images as it has not captured all the pixels that belong to the fish. However, this is most likely not true, as the left region of the images become successively darker, until the point where there is no data of the fish. Secondly, it may illustrate a strength of the method used to find binary images in that when the intensity of the pixels become too small, i.e. it would be hard to separate them from the intensities of the dark background, the method makes sure the disparity estimates in the disparity map belonging to those areas are not included.

Furthermore, one needs to discuss if a disparity map only covering parts of a fish should be valid. If one for instance wanted to know the 3D-shape of the fish, such a disparity map would not work. However, in this report the focus is not on finding the entire 3D-shape of the fish, it is on evaluating if deep learning stereo-matching algorithms manage to beat Optoscale's algorithm when it comes to finding good matches and disparity estimates. Consequently, the disparity maps that do not give estimates to every pixel belonging to the fish do contain valuable data which is worth keeping in the dataset.

6.4.3 Example 3

Example 3 showed us that some of the ground truth disparity maps do contain estimates for all parts of the fish. This tells us that the results from the methods used to predict the ground truth disparity maps do also reflect the methods' ability to find matches on all parts of the fish, and that the method used in finding binary images does not always capture only certain regions of the fish.

6.4.4 Example 4

Example 4 showed us that the method used to create the binary images is more precise when the disparities between the image pairs are larger. This tells us that the median filter size used has been optimized on image pairs with larger disparities.

6.4.5 The diversity of the dataset

All examples illustrate the diversity of the dataset, from smaller and larger disparities, to darker, brighter and a mix of the two intensities. Additionally, an observation to be made is that all fishes are "swimming" from right to left and do not show a wide range of orientations. This is the result of the setup and method used to capture the images. At different orientations, the intensities' patterns on the fish may change due to its shape. Consequently, the results on the stereo image dataset do reflect the evaluated stereo-matching algorithms' ability to match images of fish within the narrow range of orientations and range of disparities present in the dataset.

Another point to be made is that the dataset does not include challenging scenes including occlusions, and difficult lighting conditions. As a result, the dataset represents a simpler environment than that in for instance fish pens, and so the results obtained in this paper may be better than the results would be on images from a fish pen.

6.5 Optoscale’s original algorithm and the optimized version

6.5.1 Assessing if the optimized version is better than the original one

Owing to the results seen in table Table 16, and Figure 55 and Figure 56 it seems that the optimized version is more accurate than the original, and provides a more dense disparity map. The disparity map created by the original method tells us that the original algorithm struggles on some of the images. The main reason why has to be the larger disparity range, as a median filter would not be able to correct the errors present in it. Furthermore, the original version did show an accuracy of 91.99 % when using the 1-pixel error. This tells us that many of the estimates found by the original algorithm are very close to the disparities in the ground truth disparity map, and that not all the disparity maps are as wrong as the one in Figure 55. Finally, due to the empirical controls used by the optimized method and the fact that the original method did not manage to get 100 % accuracy when sharing the same stereo-matching algorithm at core, we do conclude that some improvement, in terms of finding more valid disparity estimates that are “close” the true disparity value, has been made by the steps taken in the optimization.

6.5.2 Did the choice of “Uniqueness threshold” in the optimized version have an impact?

By choosing a lower “uniqueness threshold” the results slightly improved, however they did not come close to 100 % accuracy as seen in Table 16. This means that the choice of lower “uniqueness threshold” had a marginal impact in the optimization of the algorithm.

6.6 Training and validation accuracies differ from the test accuracies

In general accuracies of the deep learning stereo-matching algorithms are better and more similar on the training and validation set than the test set. Such a result suggests that the number of training samples in the training and validation set does not capture the diversity of the data in the stereo image dataset. Consequently, it would be recommended to increase the size of the training and validation set.

6.7 Effects of Different Network Structures

6.7.1 Differences in results for MC-CNN-fst-nopost

When training with 9 layers, the test, training, and validation results, are better. One explanation could be that a bigger network gets to process the data extracted through the patch more thoroughly, and in doing so captures more complex features of the pixel and its surroundings making it easier to distinguish pixels from each other. Another explanation could be the smaller patch size of 9x9 compared to 17x17, as the convolutional filters trained to activate on features are given less data to work with. The shape of the fish is smooth, and there are no other objects present than the fish model in the images. Consequently, the scene as seen from both cameras does not change much, and so capturing more data with a bigger patch size could yield a better descriptor of a particular pixel.

6.7.2 Differences in results for Content-CNN-nopost

Peculiar results were obtained when it comes the differences in results when varying the number of convolutional layers. The smaller network would show training and validation accuracies close to those of the larger networks, while the test accuracy would drop significantly. By chance, the smaller training and validation sets may contain similar data, while the bigger test set contains data with more diversity. The smaller network learns to detect simpler features, and those may have been enough to properly predict the ground truth disparity maps in the test and validation set. However, in the larger test dataset they may have come short. On the other hand, a larger network learns to detect more complex features, and so those complex features may have generalized better to the new data in the test set.

6.7.3 Differences in training and validation accuracy between MC-CNN-fst-nopost and Content-CNN-nopost

When using 4 layers with Content-CNN-nopost, the accuracy results obtained did not drop by the same amount as with the 4-layered version of MC-CNN-fst-nopost. One explanation may be that Content-CNN-nopost's filters are of a bigger size 5x5 and not 3x3, thus they are able to detect bigger, more complex features. Another reason would be that Content-CNN-nopost processes color information, while MC-CNN-fst-nopost does not have color information available through the "grayscale" image. Thus, the extra information present in the colored structured light is not available to MC-CNN-fst-nopost. The final factor which may have influenced in Content-CNN-nopost being able to generalize better is its usage of batch normalization which is known to make a network generalize better to new data.

6.8 Discussing if the batch size had an impact on the prediction results

6.8.1 MC-CNN-fst-nopost

When it comes to the network variant with 9 layers, the batch size did not seem to make a significant difference in accuracy. The results are relatively close, and the small differences in accuracy may be due to the random weight initialization at the start of the training procedure.

The batch size did have an impact when the network consisted of 4 convolutional layers. Consequently, it may be beneficial to use a larger batch size when training MC-CNN-fst-nopost with 4 convolutional layers.

6.8.2 Content-CNN-nopost

Both variants of the network showed significantly better results on the test set when using a batch size of 32. This may have been caused by the network being able to generalize better on new data when using a smaller batch size as mentioned in section 2.6.9.

Furthermore, when using a batch size of 2 the network failed to learn properly. The reason being is most likely the learning rate of the constant learning rate scheme used being too high, as the small mini-batch gives more noise in the estimate of the loss functions gradient.

6.9 Weights extracted close to the last epoch

The results show that the weights used in evaluating a network's accuracy have been extracted close to the last epoch. This indicates that even better results could have been obtained by increasing the number of epochs to train for.

6.10 Differences in disparity ranges

Due to a blunder, the disparity range for the deep learning methods was chosen to be [0,170], while the range for Optoscale's algorithm was chosen to be [65,145] which is the best disparity range possible owing to limitations in MATLAB's Semi-Global Block Matching implementation of the disparity range. Due to aspects of the deep learning methods' code implementations, the smallest disparity was always zero, however the max disparity should have been set to 145. Owing to the differences in disparity range, the task of stereo-matching was more difficult for the deep learning methods in that they had to distinguish between more potential matches and makes it more impressive that they still managed to out-perform Optocale's algorithm.

6.11 Differences in 1-pixel error results

When using the 3-pixel error, the best accuracy of Content-CNN-nopost and MC-CNN-fst-nopost was 98.490 % and 99.864 % respectively on the test set. When using the 1-pixel error, the results were 91.049 % and 99.620 % respectively. One observes that the differences in accuracies between the two networks is significantly higher when using the stricter 1-pixel error. Thus, the particular configuration of Content-CNN-nopost which gave the best result is deemed to be better at replicating the disparity values found through the optimized version of Optoscale's method.

7 Conclusion

To conclude, the field of deep learning stereo-matching algorithms has been investigated by providing theory on how convolutional neural networks can be used as stereo image matchers and giving an overview of well-established methods in literature, including details regarding their code implementations. Furthermore, an optimized version of Optoscale's algorithm which yielded better accuracy and density in its disparity maps, when being compared to the original algorithm, has been used to create a stereo image dataset of a fish model in a freshwater tank with an empirical accuracy of 3-pixels. Moreover, the stereo image dataset was used to evaluate the accuracy of variants of the deep learning stereo-matching algorithms MC-CNN-fst (Žbontar & LeCun, 2016) and Content-CNN (Luo et al., 2016) without post-processing steps, named MC-CNN-fst-nopost and Content-CNN-nopost, against Optoscale's original stereo-matching algorithm. The results when using the 3-pixel error at base showed that the deep learning methods managed to out-perform Optoscale's algorithm, where the best accuracy result of 99.864 % was obtained with a 9 convolutional layer version of Content-CNN-nopost compared to Optoscale's algorithm's 92.071 %. Additionally, when using the 1-pixel error at base, only Content-CNN-nopost managed to significantly out-perform Optoscale's algorithm's 91.990 % accuracy with a top accuracy of 99.620 % when using 9 convolutional layers.

8 Future work

1. It would be interesting to establish the true accuracy of the stereo image dataset by redoing it with a method of known mathematical accuracy and see if the results obtained in this paper uphold.
2. It would be of value to see if the networks trained in this paper do well on real images of fish, and at the same time establish the potential to use transfer learning.
3. One could capture more stereo images of the fish model at different orientations to see if it effects the accuracy results reported.

9 Bibliography

Albregtsen, F. (2013) *INF 4300 – Digital Image Analysis Digital Image Analysis; MORPHOLOGICAL IMAGE PROCESSING.* Available at: https://www.uio.no/studier/emner/matnat/ifi/INF4300/h13/undervisningsmateriale/inf4300-2013-f07-morfologi.pdf (Accessed 24 May 2019).

Apple. (n.d.) *Digital Color Meter User Guide* Available from: https://support.apple.com/guide/digital-color-meter/welcome/mac (Accessed 31 May 2019).

Banks, J. & Corke, P. (2001) *Quantitative Evaluation of Matching Methods and Validity Measures for Stereo Vision* Available from: https://journals.sagepub.com/doi/pdf/10.1177/02783640122067525.

Bengio, Y. (2012) *Practical Recommendations for Gradient-Based Training of Deep Architectures.* (2) Available at: https://arxiv.org/pdf/1206.5533.pdf.

Birchfield , S. & Carlo, T. (1999) *Depth Discontinuities by Pixel-to-Pixel Stereo.* Available at: https://users.cs.duke.edu/~tomasi/papers/tomasi/tomasiIjcv99.pdf.

Bolles, R.C., Baker, H. & MARIMONT, D.H. (1987) *Epipolar-Plane Image Analysis: An Approach to Determining Structure from Motion*.* Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.188.992&rep=rep1&type=pdf (Accessed 22 May 2019).

Brownlee, J. (2019a) *Understand the Impact of Learning Rate on Model Performance With Deep Learning Neural Networks.* Available from: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/ (Accessed 31 May 2019).

Brownlee, J. (2019b) *How to Control the Speed and Stability of Training Neural Networks With Gradient Descent Batch Size.* Available from: https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/ (Accessed 31 May 2019).

Caffe. (n.d.) *Installation* Available at: https://caffe.berkeleyvision.org/installation.html (Accessed 19 February 2019).

Žbontar, J. & LeCun, Y. (2016) *Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches.* Available from: https://arxiv.org/abs/1510.05970.

Žbontar, J. & LeCun, Y. (n.d.) *Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches.* Available from: https://github.com/jzbontar/mc-cnn (Accessed 17 February 2019).

Geiger, A., Lenz, P., Stiller, C. & Urtasun, R. (2013) *Vision meets Robotics: The KITTI Dataset.* Available at: http://www.cvlibs.net/publications/Geiger2013IJRR.pdf.

Geiger, A., Lenz, P., Stiller, C. & Urtasun, R. (2015a) *Stereo Evaluation 2015* Available from: http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo (Accessed 2 June 2019).

Geiger, A., Lenz, P., Stiller, C. & Urtasun, R. (2015b) *Stereo Evaluation 2015* Available from: http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo (Accessed 17 February 2019).

Hata, K. & Savarese, S. (n.d.) *CS231A Course Notes 1: Camera Models* Available at: https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf (Accessed 22 May 2019).
Hirschmuller, H. (2007) *Stereo Processing by Semi-Global Matching and Mutual Information.* Available at: https://core.ac.uk/download/pdf/11134866.pdf.
Huang, T., Yang, G. & Tang, G. (1979) *A fast two-dimensional median filtering algorithm.* Available from: https://ieeexplore.ieee.org/document/1163188.
Ioffe, S. & Szegedy, C. (2015) *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* Available at: https://arxiv.org/pdf/1502.03167.pdf (Accessed May 2019).
Kaehler, A. & Bradski, G. (2017) *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. Sebastopol, California, USA: O'Reilly Media, inc.
Keskar, N.S. et al. (2017) *ON LARGE-BATCH TRAINING FOR DEEP LEARNING: GENERALIZATION GAP AND SHARP MINIMA.* Available at: https://arxiv.org/pdf/1609.04836.pdf.
Klaus, A., Sormann, M. & Karner, K. (2006) *Segment-Based Stereo Matching Using Belief Propagation and a Self-Adapting Dissimilarity Measure.* Available at: https://ieeexplore.ieee.org/abstract/document/1699458.
Konolige, K. (1997) *Small Vision Systems: Hardware and Implementation*. Menlo Park: Carnegie Mellon University Artificial Intelligence Center, SRI International.
Luo, W., Schwing, A.G. & Urtasun, R. (2016) *Efficient Deep Learning for Stereo Matching* Available at: https://ieeexplore.ieee.org/document/7780983.
Luo, W., Schwing, A.G. & Urtasun, R. (n.d.) *deepLowLevelVision* Available from: http://www.cs.toronto.edu/deepLowLevelVision/ (Accessed 17 February 2019).
Lu, C. et al. (2018) *Sparse Cost Volume for Efficient Stereo Matching* Available at: https://www.mdpi.com/2072-4292/10/11/1844.
Lu, C. et al. (n.d.) *Sparse Cost Volume for Efficient Stereo Matching* Available from: https://github.com/rairyuu/SCVNet (Accessed 17 February 2019).
Martull, S., Peris, M. & Fukui, K. (2012) *Realistic CG Stereo Image Dataset with Ground Truth Disparity Maps.* Available at: http://www.cvlab.cs.tsukuba.ac.jp/~kfukui/english/epapers/trakmark2012.pdf.
Mathworks. (n.d. a) *Bwconncomp; Find connected components in binary image.* Available from: https://se.mathworks.com/help/images/ref/bwconncomp.html (Accessed 24 May 2019).
Mathworks. (n.d. b) *Stereo Camera Calibrator App.* Available from: https://se.mathworks.com/help/vision/ug/stereo-camera-calibrator-app.html (Accessed 23 May 2019).
Mathworks. (n.d. c) *Strel; Morphological structuring element* Available from: https://se.mathworks.com/help/images/ref/strel.html (Accessed 23 May 2019).
Mathworks. (n.d. d) *disparity.* Available from: https://se.mathworks.com/help/vision/ref/disparity.html#responsive_offcanvas (Accessed 27 May 2019).
Mathworks. (n.d. e) *estimateCameraParameters.* Available from: https://se.mathworks.com/help/vision/ref/estimatecameraparameters.html (Accessed 23 May 2019).
Mathworks. (n.d. f) *rectifyStereoImages* Available from: https://se.mathworks.com/help/vision/ref/rectifystereoimages.html (Accessed 23 May 2019).

Menze, M. & Geiger, A. (2015) *Object Scene Flow for Autonomous Vehicles* Available at: http://cvlibs.net/publications/Menze2015CVPR.pdf.
Monasse, P., Morel, J.-M. & Tang, Z. (2010) *Three-step image rectification.* Available at: http://imagine.enpc.fr/publications/papers/BMVC10.pdf.
Nvidia. (n.d.) *GPU-ACCELERATED CAFFE* Available from: https://www.nvidia.com/en-gb/data-center/gpu-accelerated-applications/caffe/ (Accessed 19 February 2019).
Nvidia. (n.d.) *GPU-ACCELERATED TORCH* Available from: https://www.nvidia.com/en-gb/data-center/gpu-accelerated-applications/torch/ (Accessed 19 February 2019).
Opencv StereoSGBM. (n.d.) *Class StereoSGBM.* Available from: https://docs.opencv.org/java/2.4.9/org/opencv/calib3d/StereoSGBM.html (Accessed 27 May 2019).
OpenCV tutorial. (n.d.) *Canny Edge Detection.* Available from: https://docs.opencv.org/master/da/d22/tutorial_py_canny.html (Accessed 31 May 2019).
Optoscale AS. (n.d.) *Optoscale* Available from: https://www.optoscale.no (Accessed 1 June 2019).
Pang, J. et al. (2017) *Cascade Residual Learning: A Two-Stage Convolutional Neural Network for Stereo Matching.* Available at: http://openaccess.thecvf.com/content_ICCV_2017_workshops/papers/w17/Pang_Cascade_Residual_Learning_ICCV_2017_paper.pdf.
Pang, J. et al. (n.d.) *Cascade residual learning: A two-stage convolutional neural network for stereo matching.* Available from: https://github.com/Artifineuro/crl (Accessed 17 February 2019).
Pytorch. (n.d.) *GET STARTED* Available from: https://pytorch.org/get-started/previous-versions/ (Accessed 19 February 2019).
Rao, V. (2018) *Get started with PyTorch* Available from: https://developer.ibm.com/articles/cc-get-started-pytorch/ (Accessed 19 February 2019).
Ronan, Clément, Koray & Soumith. (n.d.) *Getting started with Torch* Available from: http://torch.ch/docs/getting-started.html (Accessed 19 February 2019).
Ruder, S. (2017) *An overview of gradient descent optimization algorithms.* Available at: https://arxiv.org/pdf/1609.04747.pdf (Accessed May 2019).
Santurkar, S., Tsipras, D., Ilyas, A. & Madry, A. (2019) *How Does Batch Normalization Help Optimization?* Available from: https://arxiv.org/abs/1805.11604.
Scharstein, D. et al. (2014) *High-Resolution Stereo Datasets with Subpixel-Accurate Ground Truth.* Available at: http://www.cs.middlebury.edu/~schar/papers/datasets-gcpr2014.pdf.
Scharstein, D. & Szeliski, R. (2001) *A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms* Available at: http://vision.middlebury.edu/stereo/taxonomy-IJCV.pdf.
Shannon, C.E. (1948) *A Mathematical Theory of Communication* Available at: http://math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf.
Srivastava, N. et al. (2014) *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Available at: http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf?utm_content=buffer79b43&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer (Accessed May 2019).
Stanford University. (n.d. a) *Convolutional Neural Networks (CNNs / ConvNets)* Available from: http://cs231n.github.io/convolutional-networks/ (Accessed 31 May 2019).

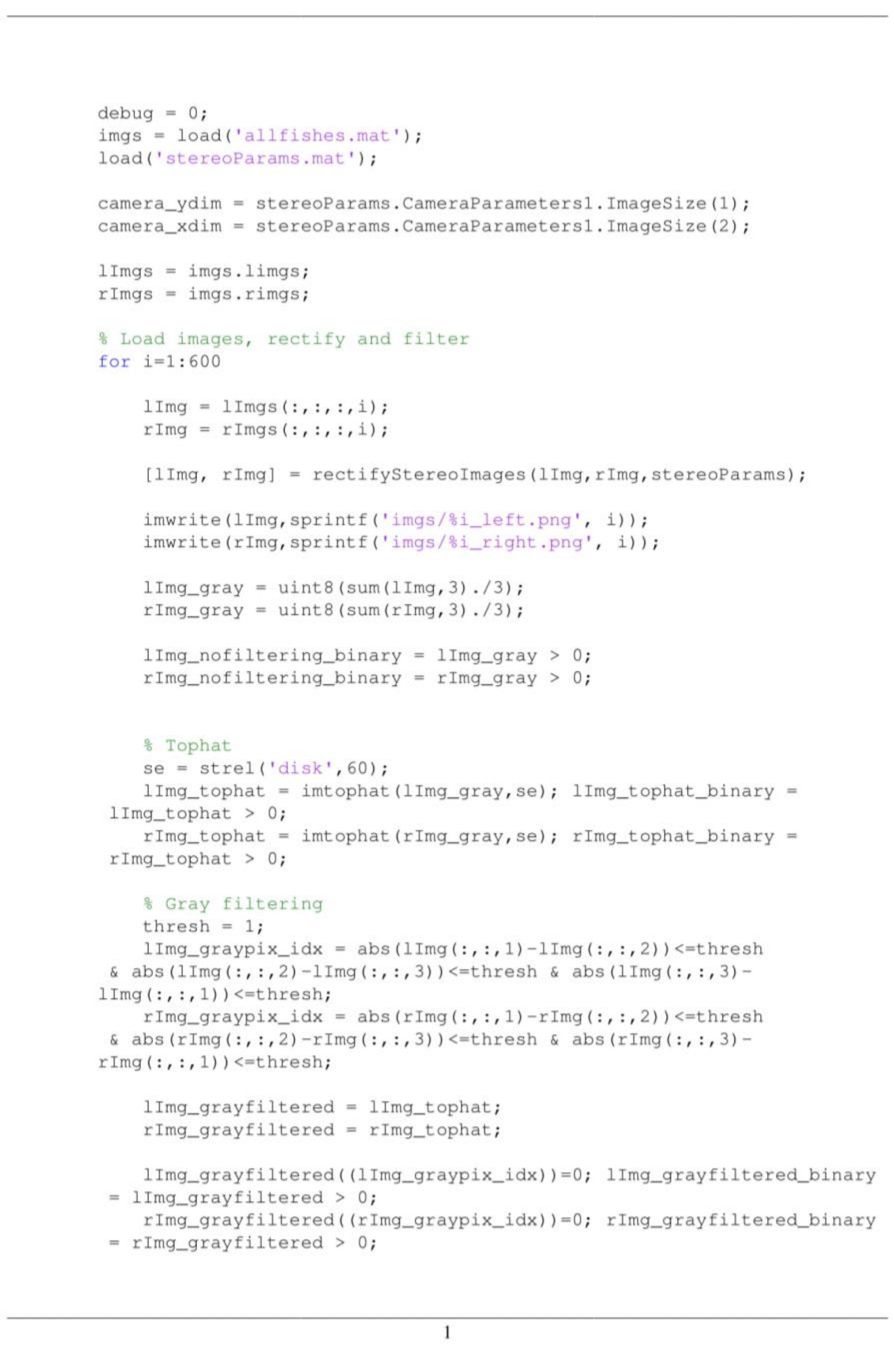
Stanford University. (n.d. b) *Linear Classification.* Available from: http://cs231n.github.io/linear-classify/ (Accessed 31 May 2019).
Stanford University. (n.d. c) *Quick intro* Available from: http://cs231n.github.io/neural-networks-1/ (Accessed 31 May 2019).
Szeliski, R. (2010) *Computer Vision: Algorithms and Applications* Available at: http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf.
Vikram, F. (2018) *Installing pytorch and tensorflow with CUDA enabled GPU* Available from: https://medium.com/datadriveninvestor/installing-pytorch-and-tensorflow-with-cuda-enabled-gpu-f747e6924779 (Accessed 19 February 2019).
Vincent, O.R. & Folorunso, O. (2009) *A Descriptive Algorithm for Sobel Image Edge Detection.* Available from: http://proceedings.informingscience.org/InSITE2009/InSITE09p097-107Vincent613.pdf (Accessed 24 May 2019).
Westlie, S. (2018) *An investigation into using state-of-the-art convolutional neural network object detectors as real-time fish detectors on underwater grayscale images*. Trondheim: Norwegian University of Science and Technology.
Yang, G. et al. (2018) *SegStereo: Exploiting Semantic Information for Disparity Estimation.* Available at: http://openaccess.thecvf.com/content_ECCV_2018/papers/Guorun_Yang_SegStereo_Exploiting_Semantic_ECCV_2018_paper.pdf.
Yang, G. et al. (n.d.) *SegStereo: Exploiting Semantic Information for Disparity Estimation* ECCV Available at: https://github.com/yangguorun/SegStereo (Accessed 17 February 2019).
Zabih, R. & Woodfill, J. (2005) *Nonparametric Local Transforms for Computing Visual Correspondence.* Available at: http://www.cs.cornell.edu/~rdz/Papers/ZW-ECCV94.pdf.
Zagoruyko, S. & Komodakis, N. (2015) *Learning to Compare Image Patches via Convolutional Neural Networks* (1) Available at: https://arxiv.org/pdf/1504.03641.pdf.

10 Appendix

10.1 Table of figures

10.2 MATLAB scripts used in the creation of ground truth disparity maps

10.2.1 Script implementing the filter pipeline which finds the binary image of the fish



```
debug = 0;
imgs = load('allfishes.mat');
load('stereoParams.mat');

camera_ydim = stereoParams.CameraParameters1.ImageSize(1);
camera_xdim = stereoParams.CameraParameters1.ImageSize(2);

lImgs = imgs.limgs;
rImgs = imgs.rimgs;

% Load images, rectify and filter
for i=1:600

    lImg = lImgs(:,:,:,i);
    rImg = rImgs(:,:,:,i);

    [lImg, rImg] = rectifyStereoImages(lImg,rImg,stereoParams);

    imwrite(lImg,sprintf('imgs/%i_left.png', i));
    imwrite(rImg,sprintf('imgs/%i_right.png', i));

    lImg_gray = uint8(sum(lImg,3)./3);
    rImg_gray = uint8(sum(rImg,3)./3);

    lImg_nofiltering_binary = lImg_gray > 0;
    rImg_nofiltering_binary = rImg_gray > 0;


    % Tophat
    se = strel('disk',60);
    lImg_tophat = imtophat(lImg_gray,se); lImg_tophat_binary =
 lImg_tophat > 0;
    rImg_tophat = imtophat(rImg_gray,se); rImg_tophat_binary =
 rImg_tophat > 0;

    % Gray filtering
    thresh = 1;
    lImg_graypix_idx = abs(lImg(:,:,1)-lImg(:,:,2))<=thresh
 & abs(lImg(:,:,2)-lImg(:,:,3))<=thresh & abs(lImg(:,:,3)-
lImg(:,:,1))<=thresh;
    rImg_graypix_idx = abs(rImg(:,:,1)-rImg(:,:,2))<=thresh
 & abs(rImg(:,:,2)-rImg(:,:,3))<=thresh & abs(rImg(:,:,3)-
rImg(:,:,1))<=thresh;

    lImg_grayfiltered = lImg_tophat;
    rImg_grayfiltered = rImg_tophat;

    lImg_grayfiltered((lImg_graypix_idx))=0; lImg_grayfiltered_binary
 = lImg_grayfiltered > 0;
    rImg_grayfiltered((rImg_graypix_idx))=0; rImg_grayfiltered_binary
 = rImg_grayfiltered > 0;
```

1

```
   %Thresholding

   lImg_meanthreshed = lImg_grayfiltered;
   rImg_meanthreshed = lImg_grayfiltered;
   intensity_threshold_left = mean(lImg_meanthreshed,1)/3.0;
   intensity_threshold_right = mean(rImg_meanthreshed,1)/3.0;
   lImg_meanthreshed_binary = lImg_meanthreshed >
intensity_threshold_left;
   rImg_meanthreshed_binary = rImg_meanthreshed >
intensity_threshold_right;
   lImg_meanthreshed(not(lImg_meanthreshed_binary))=0;
   rImg_meanthreshed(not(rImg_meanthreshed_binary))=0;

   % Column thresholding
   lImg_colmeanthreshed = lImg_meanthreshed;
   rImg_colmeanthreshed = rImg_meanthreshed;
   for j=1:size(lImg,2)
       lImg_col = lImg_colmeanthreshed(:,j);
       lImg_pix_vals_no_zero = lImg_col > 0;
       lImg_col_mean = mean(lImg_col(lImg_pix_vals_no_zero))/10.0;
       lImg_pix_above_thresh = lImg_col > lImg_col_mean;
       lImg_col(not(lImg_pix_above_thresh)) = 0;
       lImg_colmeanthreshed(:,j) = lImg_col;

       rImg_col = rImg_colmeanthreshed(:,j);
       rImg_pix_vals_no_zero = rImg_col > 0;
       rImg_col_mean = mean(rImg_col(rImg_pix_vals_no_zero))/10.0;
       rImg_pix_above_thresh = rImg_col > rImg_col_mean;
       rImg_col(not(rImg_pix_above_thresh)) = 0;
       rImg_colmeanthreshed(:,j) = rImg_col;
   end

   lImg_colmeanthreshed_binary = lImg_colmeanthreshed > 0;
   rImg_colmeanthreshed_binary = rImg_colmeanthreshed > 0;


   % Median filter
   filter_size = [5,5];
   lImg_median_binary =
medfilt2(lImg_colmeanthreshed_binary,filter_size);
   rImg_median_binary =
medfilt2(rImg_colmeanthreshed_binary,filter_size);

   % Smooth edges
   windowSize = 7;
   kernel = ones(windowSize) / windowSize ^ 2;
   lImg_blurred_binary = conv2(single(lImg_median_binary),
kernel, 'same');
   rImg_blurred_binary = conv2(single(rImg_median_binary),
kernel, 'same');

   lImg_binary_blurred_threshed = lImg_blurred_binary > 0.5; %
Rethreshold
```

```
    rImg_binary_blurred_threshed = rImg_blurred_binary > 0.5; %
 Rethreshold

    % Hole filling
    lImg_filled_binary = imfill(lImg_binary_blurred_threshed,'holes');
    rImg_filled_binary = imfill(rImg_binary_blurred_threshed,'holes');

    % Connected components
    lImg_connected_binary = bwareaopen(lImg_filled_binary, 20000);
    rImg_connected_binary = bwareaopen(rImg_filled_binary, 20000);

    if(debug == 0)
        imwrite(lImg_nofiltering_binary,sprintf('imgs_nofilter/
%i_left_binary.png', i));
        imwrite(lImg_gray,sprintf('imgs_gray/%i_left_gray.png', i));
        imwrite(lImg_tophat_binary,sprintf('imgs_tophat/
%i_left_tophat.png', i));
        imwrite(lImg_grayfiltered_binary,sprintf('imgs_grayfilter/
%i_left_grayfilter.png', i));
        imwrite(lImg_meanthreshed_binary,sprintf('imgs_meanthresh/
%i_left_meanthreshed.png', i));

 imwrite(lImg_colmeanthreshed_binary,sprintf('imgs_colmeanthresh/
%i_left_meanthreshed.png', i));
        imwrite(lImg_median_binary,sprintf('imgs_median/
%i_left_binary_median.png', i));
        imwrite(lImg_binary_blurred_threshed,sprintf('imgs_smoothed/
%i_left_binary_smoothed.png', i));
        imwrite(lImg_filled_binary,sprintf('imgs_filled/
%i_left_binary_filled.png', i));
        imwrite(lImg_connected_binary,sprintf('imgs_connected/
%i_left_binary_connected.png', i));
        imwrite(lImg_connected_binary,sprintf('imgs_binary/
%i_left_binary.png', i));

        imwrite(rImg_nofiltering_binary,sprintf('imgs_nofilter/
%i_right_binary.png', i));
        imwrite(rImg_gray,sprintf('imgs_gray/%i_right_gray.png', i));
        imwrite(rImg_tophat_binary,sprintf('imgs_tophat/
%i_right_tophat.png', i));
        imwrite(rImg_grayfiltered_binary,sprintf('imgs_grayfilter/
%i_right_grayfilter.png', i));
        imwrite(rImg_meanthreshed_binary,sprintf('imgs_meanthresh/
%i_right_meanthreshed.png', i));

 imwrite(rImg_colmeanthreshed_binary,sprintf('imgs_colmeanthresh/
%i_right_meanthreshed.png', i));
        imwrite(rImg_median_binary,sprintf('imgs_median/
%i_right_binary_median.png', i));
        imwrite(rImg_binary_blurred_threshed,sprintf('imgs_smoothed/
%i_right_binary_smoothed.png', i));
        imwrite(rImg_filled_binary,sprintf('imgs_filled/
%i_right_binary_filled.png', i));
```

```
        imwrite(rImg_connected_binary,sprintf('imgs_connected/
%i_right_binary_connected.png', i));
        imwrite(rImg_connected_binary,sprintf('imgs_binary/
%i_right_binary.png', i));
    end

    fprintf(sprintf('Image #: %i \n',i));
end
```

Published with MATLAB® R2018b

10.2.2 Script used to manually mark gaps to fill

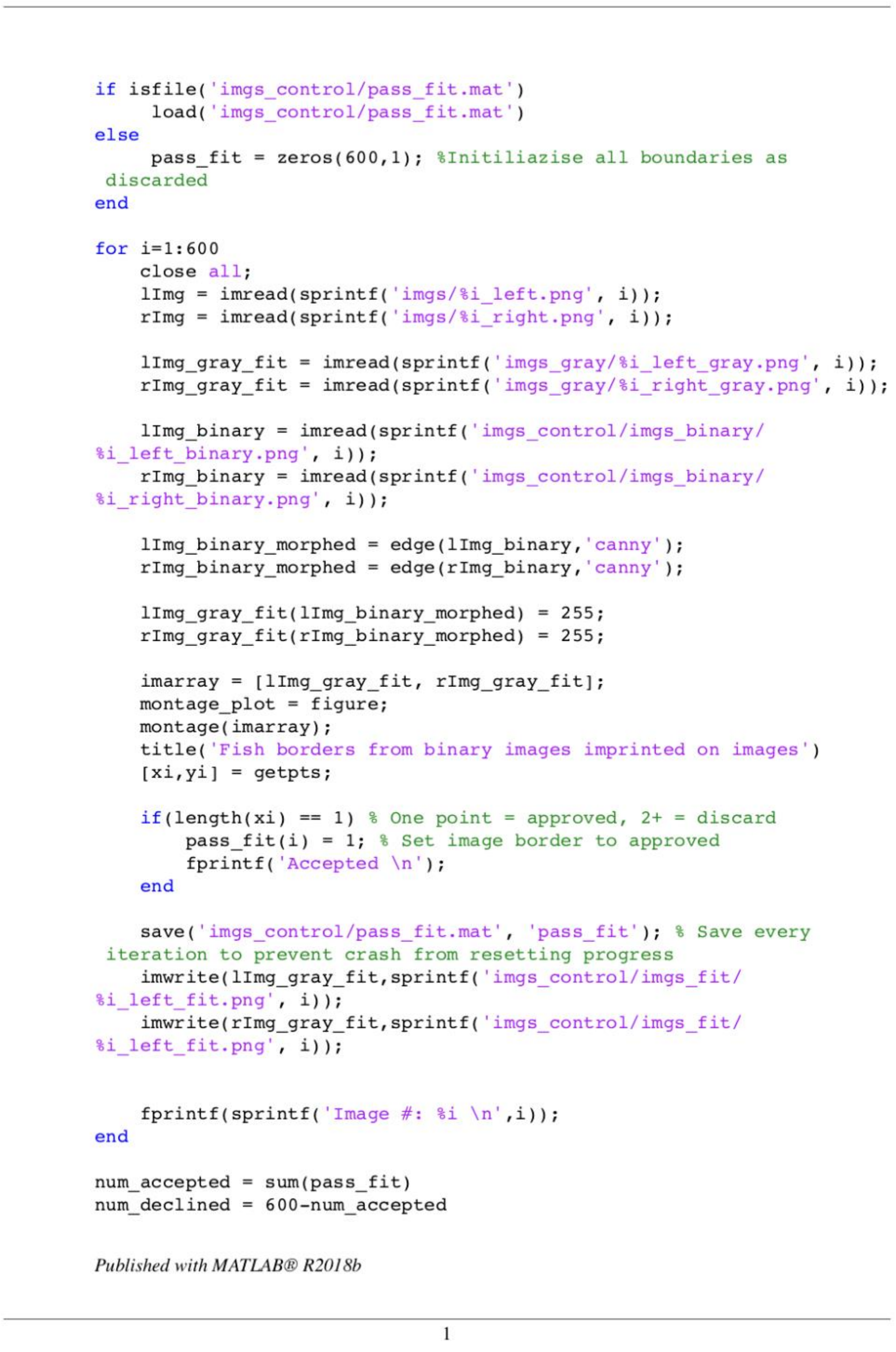
```
for i=1:600
    close all;
    % Load data
    lImg = imread(sprintf('imgs/%i_left.png', i));
    rImg = imread(sprintf('imgs/%i_right.png', i));

    lImg_binary = imread(sprintf('imgs_binary/%i_left_binary.png',
 i));
    rImg_binary = imread(sprintf('imgs_binary/%i_right_binary.png',
 i));

    % Show left and right binary image. Set x ranges for gap filling
 and cuts.

    figure
    imshow(rImg_binary)
    title('Right image binary');

    [xi,yi] = getpts;

    if(length(xi) ~= 1)
        xi = uint16(xi);
        rImg_left_region = rImg_binary(:,xi(1):xi(2));
        rImg_right_region = rImg_binary(:,xi(3):xi(4));
        SE = strel('line',19,0);
        rImg_left_region_closed = imclose(rImg_left_region,SE);
        rImg_right_region_closed = imclose(rImg_right_region,SE);
        rImg_binary(:,xi(1):xi(2)) = rImg_left_region_closed;
        rImg_binary(:,xi(3):xi(4)) = rImg_right_region_closed;

        if(length(xi) > 4)
            rImg_width = size(rImg_binary, 2);
            rImg_binary(:,1:xi(5)) = 0;
            rImg_binary(:,xi(6):rImg_width) = 0;
        end
    end
    imwrite(rImg,sprintf('imgs_control/imgs/%i_right.png', i));
    imwrite(rImg_binary,sprintf('imgs_control/imgs_binary/
%i_right_binary.png', i));

    close all;

    figure
    imshow(lImg_binary)
    title('Left image binary')
    [xi,yi] = getpts;

    if(length(xi) ~= 1)
        xi = uint16(xi);

        lImg_left_region = lImg_binary(:,xi(1):xi(2));
        lImg_right_region = lImg_binary(:,xi(3):xi(4));
```

1

```
        SE = strel('line',19,0);
        lImg_left_region_closed = imclose(lImg_left_region,SE);
        lImg_right_region_closed = imclose(lImg_right_region,SE);
        lImg_binary(:,xi(1):xi(2)) = lImg_left_region_closed;
        lImg_binary(:,xi(3):xi(4)) = lImg_right_region_closed;

        if(length(xi) > 4)
            lImg_width = size(lImg_binary, 2);
            lImg_binary(:,1:xi(5)) = 0;
            lImg_binary(:,xi(6):lImg_width) = 0;
        end
    end

    imwrite(lImg,sprintf('imgs_control/imgs/%i_left.png', i));
    imwrite(lImg_binary,sprintf('imgs_control/imgs_binary/
%i_left_binary.png', i));


    fprintf(sprintf('Image #: %i \n',i));

end
```

Published with MATLAB® R2018b

10.2.3 Script used to control how well the binary image fits the fish image



```
if isfile('imgs_control/pass_fit.mat')
     load('imgs_control/pass_fit.mat')
else
     pass_fit = zeros(600,1); %Initiliazise all boundaries as
 discarded
end

for i=1:600
    close all;
    lImg = imread(sprintf('imgs/%i_left.png', i));
    rImg = imread(sprintf('imgs/%i_right.png', i));

    lImg_gray_fit = imread(sprintf('imgs_gray/%i_left_gray.png', i));
    rImg_gray_fit = imread(sprintf('imgs_gray/%i_right_gray.png', i));

    lImg_binary = imread(sprintf('imgs_control/imgs_binary/
%i_left_binary.png', i));
    rImg_binary = imread(sprintf('imgs_control/imgs_binary/
%i_right_binary.png', i));

    lImg_binary_morphed = edge(lImg_binary,'canny');
    rImg_binary_morphed = edge(rImg_binary,'canny');

    lImg_gray_fit(lImg_binary_morphed) = 255;
    rImg_gray_fit(rImg_binary_morphed) = 255;

    imarray = [lImg_gray_fit, rImg_gray_fit];
    montage_plot = figure;
    montage(imarray);
    title('Fish borders from binary images imprinted on images')
    [xi,yi] = getpts;

    if(length(xi) == 1) % One point = approved, 2+ = discard
        pass_fit(i) = 1; % Set image border to approved
        fprintf('Accepted \n');
    end

    save('imgs_control/pass_fit.mat', 'pass_fit'); % Save every
 iteration to prevent crash from resetting progress
    imwrite(lImg_gray_fit,sprintf('imgs_control/imgs_fit/
%i_left_fit.png', i));
    imwrite(rImg_gray_fit,sprintf('imgs_control/imgs_fit/
%i_left_fit.png', i));


    fprintf(sprintf('Image #: %i \n',i));
end

num_accepted = sum(pass_fit)
num_declined = 600-num_accepted
```

Published with MATLAB® R2018b

1

10.2.4 Script used to crop images

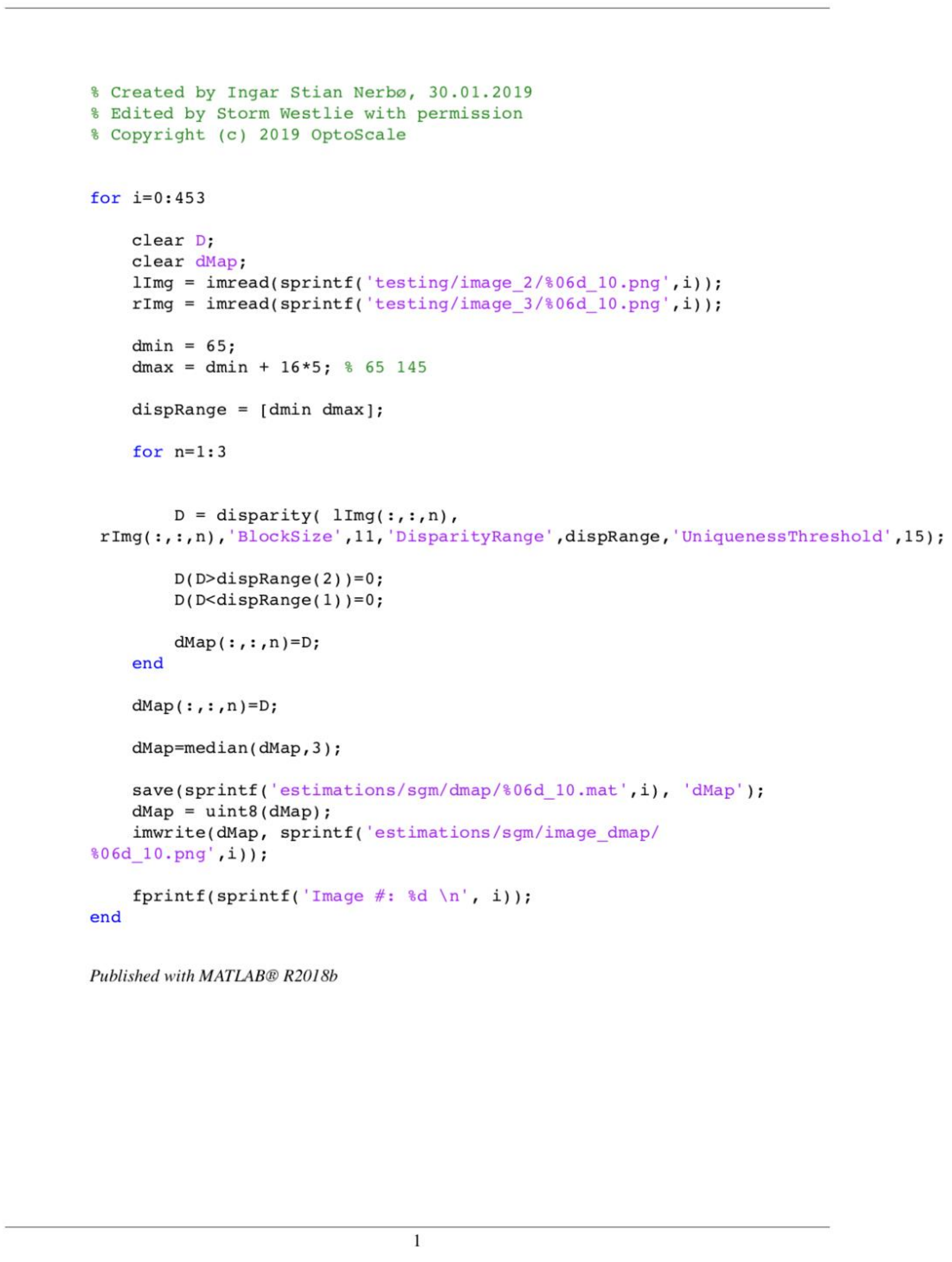
```
% Initialize data
o_crop = 36;

for i=1:600
    lImg = imread(sprintf('imgs/%i_left.png', i));
    rImg = imread(sprintf('imgs/%i_right.png', i));
    lImg_gray = imread(sprintf('imgs_gray/%i_left_gray.png',i));
    rImg_gray = imread(sprintf('imgs_gray/%i_right_gray.png',i));


    lImg_binary = imread(sprintf('imgs_control/imgs_binary/
%i_left_binary.png', i));
    rImg_binary = imread(sprintf('imgs_control/imgs_binary/
%i_right_binary.png', i));

    I_w = size(lImg_binary, 2);
    I_h = size(lImg_binary, 1);
    % Crop region height

    x_vector = ones(1,I_w); % Vector of ones with size 1xI_w

    y_min_L = -1; Y_min_L = -1;
    y_min_R = -1; Y_min_R = -1;
    y_max_L = -1; Y_max_L = -1;
    y_max_R = -1; Y_max_R = -1;
    Y_min = -1; Y_max = -1;

    % y_min_L
    for y=1:I_h
        dotProduct = dot(x_vector, lImg_binary(y,:));
        if dotProduct > 0
            y_min_L = y;
            break;
        end
    end

    % y_min_R
    for y=1:I_h
        dotProduct = dot(x_vector, rImg_binary(y,:));
        if dotProduct > 0
            y_min_R = y;
            break;
        end
    end

    % y_max_L
    for y=I_h:-1:1
        dotProduct = dot(x_vector, lImg_binary(y,:));
        if dotProduct > 0
            y_max_L = y;
            break;
        end
```

1

```
end

% y_max_R
for y=I_h:-1:1
    dotProduct = dot(x_vector, rImg_binary(y,:));
    if dotProduct > 0
        y_max_R = y;
        break;
    end
end

Y_min_L = y_min_L - o_crop/2;
Y_min_R = y_min_R - o_crop/2;
Y_max_L = y_max_L + o_crop/2;
Y_max_R = y_max_R + o_crop/2;

% Y_min
if(Y_min_L > 1 && Y_min_R > 1)
    Y_min = min(Y_min_L,Y_min_R);
elseif(Y_min_L > 1 && Y_min_R < 1)
    Y_min = Y_min_L;
elseif(Y_min_L < 1 && Y_min_R > 1)
    Y_min = Y_min_R;
else
    Y_min = 1;
end

% Y_max
if(Y_max_L < I_h && Y_max_R < I_h)
    Y_max = max(Y_max_L,Y_max_R);
elseif(Y_max_L < I_h && Y_max_R > I_h)
    Y_max = Y_max_L;
elseif(Y_max_L > I_h && Y_max_R < I_h)
    Y_max = Y_max_R;
else
    Y_max = I_h;
end

H_crop = Y_max-Y_min;

% Crop region width

d_max = 170;
y_vector = ones(I_h,1); % Vector of size I_hx1

x_min_L = -1; X_min_L = -1;
x_min_R = -1; X_min_R = -1;
x_max_L = -1; X_max_L = -1;
x_max_R = -1; X_max_R = -1;
X_min = -1; X_max = -1;

% x_min_L
for x=1:I_w
    dotProduct = dot(y_vector, lImg_binary(:,x));
    if dotProduct > 0
        x_min_L = x;
        break;
    end
end

% x_min_R
for x=1:I_w
    dotProduct = dot(y_vector, rImg_binary(:,x));
    if dotProduct > 0
        x_min_R = x;
        break;
    end
end

% x_max_L
for x=I_w:-1:1
    dotProduct = dot(y_vector, lImg_binary(:,x));
    if dotProduct > 0
        x_max_L = x;
        break;
    end
end

% x_max_R
for x=I_w:-1:1
    dotProduct = dot(y_vector, rImg_binary(:,x));
    if dotProduct > 0
        x_max_R = x;
        break;
    end
end

X_min_L = x_min_L - o_crop/2 - d_max;
X_min_R = x_min_R - o_crop/2 - d_max;
X_max_L = x_max_L + o_crop/2 + d_max;
X_max_R = x_max_R + o_crop/2 + d_max;

% X_min
if(X_min_L > 1 && X_min_R > 1)
    X_min = min(X_min_L,X_min_R);
elseif(X_min_L > 1 && X_min_R < 1)
    X_min = X_min_L;
elseif(X_min_L < 1 && X_min_R > 1)
    X_min = X_min_R;
else
    X_min = 1;
end

% X_max
if(X_max_L < I_w && X_max_R < I_w)
    X_max = max(X_max_L,X_max_R);
elseif(X_max_L < I_w && X_max_R > I_w)
    X_max = X_max_L;
    elseif(X_max_L > I_w && X_max_R < I_w)
        X_max = X_max_R;
    else
        X_max = I_w;
    end

    W_crop = X_max-X_min;

    % Crop L/R binary and RGB images

    crop_region = [X_min, Y_min, W_crop, H_crop];

    lImg_cropped = imcrop(lImg, crop_region);
    rImg_cropped = imcrop(rImg, crop_region);

    lImg_binary_cropped = imcrop(lImg_binary, crop_region);
    rImg_binary_cropped = imcrop(rImg_binary, crop_region);

    lImg_gray_cropped = imcrop(lImg_gray, crop_region);
    rImg_gray_cropped = imcrop(rImg_gray, crop_region);

    imwrite(lImg_cropped,sprintf('imgs_cropped/%i_left_cropped.png',
 i));
    imwrite(lImg_binary_cropped,sprintf('imgs_binary_cropped/
%i_left_binary_cropped.png', i));
    imwrite(lImg_gray_cropped,sprintf('imgs_gray_cropped/
%i_left_gray_cropped.png', i));

    imwrite(rImg_cropped,sprintf('imgs_cropped/%i_right_cropped.png',
 i));
    imwrite(rImg_binary_cropped,sprintf('imgs_binary_cropped/
%i_right_binary_cropped.png', i));
    imwrite(rImg_gray_cropped,sprintf('imgs_gray_cropped/
%i_right_gray_cropped.png', i));

    fprintf(sprintf('Image #: %i \n',i));

end
```

Published with MATLAB® R2018b

10.2.5 Script implementing Optoscale's stereo-matching algorithm



```
% Created by Ingar Stian Nerbø, 30.01.2019
% Edited by Storm Westlie with permission
% Copyright (c) 2019 OptoScale


for i=0:453

    clear D;
    clear dMap;
    lImg = imread(sprintf('testing/image_2/%06d_10.png',i));
    rImg = imread(sprintf('testing/image_3/%06d_10.png',i));

    dmin = 65;
    dmax = dmin + 16*5; % 65 145

    dispRange = [dmin dmax];

    for n=1:3


        D = disparity( lImg(:,:,n),
 rImg(:,:,n),'BlockSize',11,'DisparityRange',dispRange,'UniquenessThreshold',15);

        D(D>dispRange(2))=0;
        D(D<dispRange(1))=0;

        dMap(:,:,n)=D;
    end

    dMap(:,:,n)=D;

    dMap=median(dMap,3);

    save(sprintf('estimations/sgm/dmap/%06d_10.mat',i), 'dMap');
    dMap = uint8(dMap);
    imwrite(dMap, sprintf('estimations/sgm/image_dmap/
%06d_10.png',i));

    fprintf(sprintf('Image #: %d \n', i));
end
```

Published with MATLAB® R2018b

1

10.2.6 Script used to create the ground truth disparity maps

```
% Created by Ingar Stian Nerbø, 30.01.2019
% Edited by Storm Westlie with permission
% Copyright (c) 2019 OptoScale

load('pass_fit.mat');
if isfile('pass_groundtruth.mat')
     load('pass_groundtruth.mat')
else
     pass_groundtruth = zeros(600,1); %Initiliazise all ground truths
 as discarded
end

for i=1:600

    fprintf(sprintf('Image #: %i \n', i));

    % Check if valid binary mask
    if(pass_fit(i) == 0)
        pass_groundtruth(i) = 0;
        fprintf('No valid binary mask \n');
        continue;
    end

    % Load data
    lImg = imread(sprintf('imgs_cropped/%i_left_cropped.png',i));
    rImg = imread(sprintf('imgs_cropped/%i_right_cropped.png',i));

    lImg_gray = imread(sprintf('imgs_gray_cropped/
%i_left_gray_cropped.png',i));
    rImg_gray = imread(sprintf('imgs_gray_cropped/
%i_right_gray_cropped.png',i));

    % Plot stereoanaglyph and estimate smallest disparity
    sa_lImg_rImg = stereoAnaglyph(lImg_gray, rImg_gray);
    imtool(sa_lImg_rImg);

    first_stage_action = 0;
    while(first_stage_action == 0)

        % Pre clean up
        close all;
        clear D;
        clear dMap_all;
        clear dMap;

        % Calculate dMap
        B = imread(sprintf('imgs_binary_cropped/
%i_left_binary_cropped.png',i));

        d_min = str2double(input('Estimated dmin: ', 's'));
        d_max = d_min + 16;
```

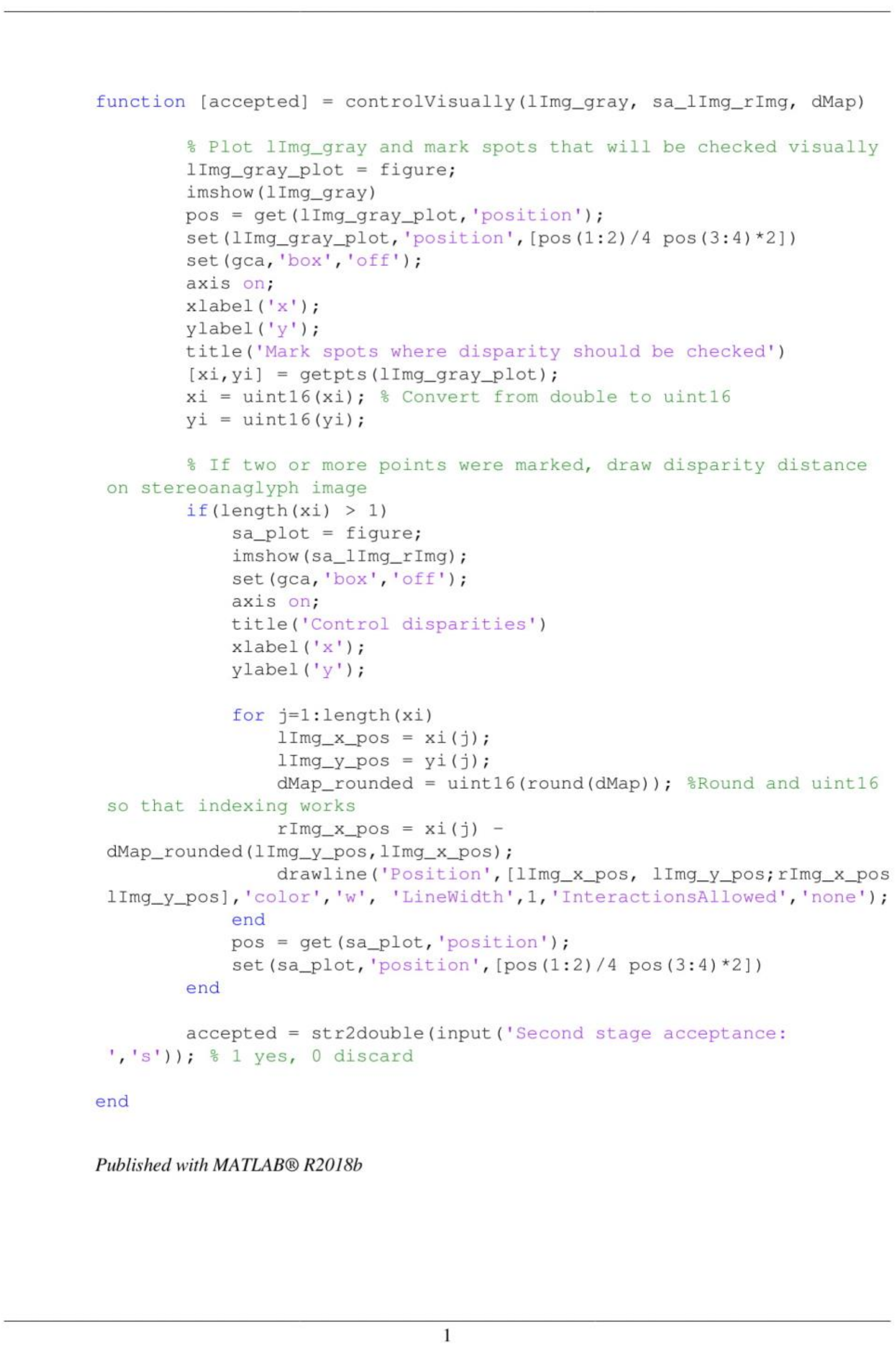
1

```
        disp_range = [d_min d_max];


        for n=1:3
            dMap_channel = disparity( lImg(:,:,n),
rImg(:,:,n),'BlockSize',11,'DisparityRange',disp_range,'UniquenessThreshold',1);
            dMap_channel(not(B))=0;
            dMap_channel(dMap_channel>d_max)=0;
            dMap_channel(dMap_channel<d_min)=0;

            dMap_all(:,:,n)=dMap_channel;

        end

        % Join disparity estimates by taking the median of channel
estimates
        dMap=median(dMap_all,3);

        % Median filter disparity map and repair border

        filter_size = [5,5];
        dMap_median = medfilt2(dMap,filter_size);

        dMap_median(not(B))=0;
        dMap_binary = dMap > 0;
        dMap_median_binary = dMap_median == 0;
        dMap_repair = dMap_binary & dMap_median_binary;
        dMap_median(dMap_repair) = dMap(dMap_repair);
        dMap = dMap_median;

        % Check for saturation on disparity range
        saturated = ismember([d_min (d_max-1)],dMap);
        dmin_saturated = saturated(1);
        dmax_saturated = saturated(2);
        if(dmin_saturated)
            fprintf('dmin saturated! \n');
            img_min_sat = dMap == d_min;
            figure
            imshow(img_min_sat);
            set(gca,'box','off');
            axis on;
            title('Pixels who saturated on disparity min')
            xlabel('x');
            ylabel('y');
            axis on;
        end
        if(dmax_saturated)
            fprintf('dmax saturated! \n');
            img_max_sat = dMap == (d_max-1);
            figure;
            imshow(img_max_sat);
            set(gca,'box','off');
            axis on;
            title('Pixels who saturated on disparity max')
            xlabel('x');
            ylabel('y');
            axis on;
        end

        % Plot disparity map with colorbar

        d_plot = figure;
        imagesc(dMap);
        axis image;
        title('Ground truth estimate')
        xlabel('x')
        ylabel('y')
        pos = get(d_plot,'position');
        set(d_plot,'position',[pos(1:2)/4 pos(3:4)*2])
        cmap = pink(256);
        colormap(cmap);
        cb = colorbar;
        dMap_min = min(dMap(dMap >0));
        dMap_max = max(dMap,[],'all');
        caxis([dMap_min-1, dMap_max])
        title(cb,'Disparity colors')



        % Decide on action after first stage
        first_stage_action = str2double(input('First stage acceptance:
 ','s')); %1 Visually inspect, 0 retry, -1 discard
        switch(first_stage_action )
            case 1
                second_stage_acceptance =
 controlVisually(lImg_gray,sa_lImg_rImg, dMap);
                if(second_stage_acceptance == 1)
                    fprintf('Accepted \n')
                    pass_groundtruth(i) = 1;
                    save('pass_groundtruth.mat', 'pass_groundtruth');
                    save(sprintf('imgs_groundtruth/
%i_groundtruth.mat',i), 'dMap');
                else
                    fprintf('Declined \n')
                    pass_groundtruth(i) = 0;
                    save('pass_groundtruth.mat', 'pass_groundtruth');
                    save(sprintf('imgs_groundtruth/
%i_groundtruth.mat',i), 'dMap')
                end
                break % End while loop and go to next image
            case -1
                fprintf('Declined \n')
                pass_groundtruth(i) = 0;
                save(sprintf('imgs_groundtruth/
%i_groundtruth.mat',i), 'dMap');
                save('pass_groundtruth.mat', 'pass_groundtruth');
                if(dmin_saturated)
```

```
                    imwrite(img_min_sat,sprintf('imgs_saturation/
%i_min.png',i));
                end
                if(dmax_saturated)
                    imwrite(img_max_sat,sprintf('imgs_saturation/
%i_max.png',i));
                end
                break % End while loop and go to next image
            case 0
                % Adjust smallest disparity and retry
        end
    end

    % Post clean up
    imtool close all;
end

num_accepted = sum(pass_groundtruth)
num_declined = sum(pass_fit)-num_accepted
```

Published with MATLAB® R2018b

10.2.7 Script used to interactively mark spots and control disparity values

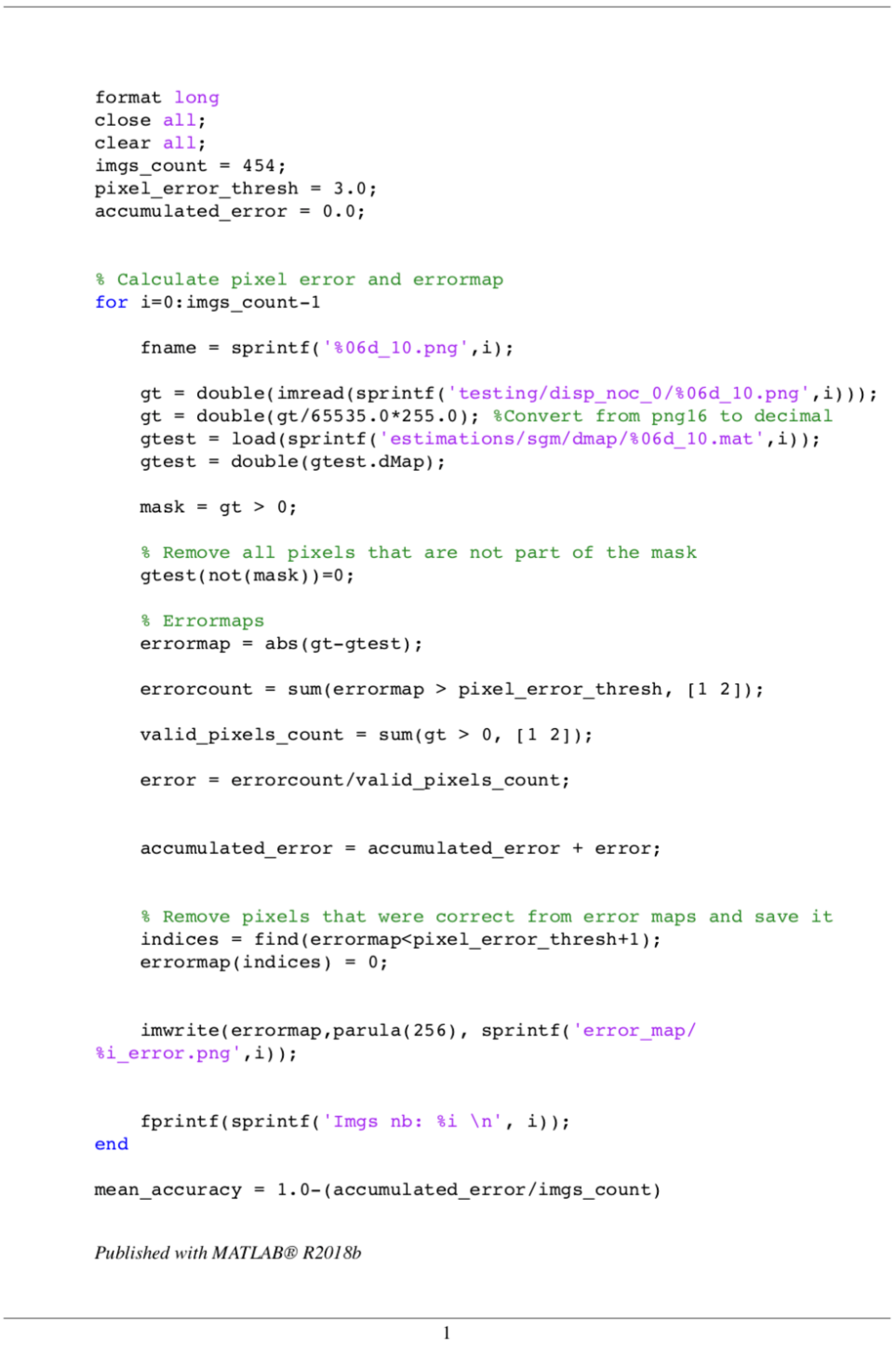


```
function [accepted] = controlVisually(lImg_gray, sa_lImg_rImg, dMap)

        % Plot lImg_gray and mark spots that will be checked visually
        lImg_gray_plot = figure;
        imshow(lImg_gray)
        pos = get(lImg_gray_plot,'position');
        set(lImg_gray_plot,'position',[pos(1:2)/4 pos(3:4)*2])
        set(gca,'box','off');
        axis on;
        xlabel('x');
        ylabel('y');
        title('Mark spots where disparity should be checked')
        [xi,yi] = getpts(lImg_gray_plot);
        xi = uint16(xi); % Convert from double to uint16
        yi = uint16(yi);

        % If two or more points were marked, draw disparity distance
 on stereoanaglyph image
        if(length(xi) > 1)
            sa_plot = figure;
            imshow(sa_lImg_rImg);
            set(gca,'box','off');
            axis on;
            title('Control disparities')
            xlabel('x');
            ylabel('y');

            for j=1:length(xi)
                lImg_x_pos = xi(j);
                lImg_y_pos = yi(j);
                dMap_rounded = uint16(round(dMap)); %Round and uint16
 so that indexing works
                rImg_x_pos = xi(j) -
 dMap_rounded(lImg_y_pos,lImg_x_pos);
                drawline('Position',[lImg_x_pos, lImg_y_pos;rImg_x_pos
 lImg_y_pos],'color','w', 'LineWidth',1,'InteractionsAllowed','none');
            end
            pos = get(sa_plot,'position');
            set(sa_plot,'position',[pos(1:2)/4 pos(3:4)*2])
        end

        accepted = str2double(input('Second stage acceptance:
 ','s')); % 1 yes, 0 discard

end
```

Published with MATLAB® R2018b

1

10.2.8 Script used to evaluate Optoscale's algorithm on the test dataset



```
format long
close all;
clear all;
imgs_count = 454;
pixel_error_thresh = 3.0;
accumulated_error = 0.0;


% Calculate pixel error and errormap
for i=0:imgs_count-1

    fname = sprintf('%06d_10.png',i);

    gt = double(imread(sprintf('testing/disp_noc_0/%06d_10.png',i)));
    gt = double(gt/65535.0*255.0); %Convert from png16 to decimal
    gtest = load(sprintf('estimations/sgm/dmap/%06d_10.mat',i));
    gtest = double(gtest.dMap);

    mask = gt > 0;

    % Remove all pixels that are not part of the mask
    gtest(not(mask))=0;

    % Errormaps
    errormap = abs(gt-gtest);

    errorcount = sum(errormap > pixel_error_thresh, [1 2]);

    valid_pixels_count = sum(gt > 0, [1 2]);

    error = errorcount/valid_pixels_count;


    accumulated_error = accumulated_error + error;


    % Remove pixels that were correct from error maps and save it
    indices = find(errormap<pixel_error_thresh+1);
    errormap(indices) = 0;


    imwrite(errormap,parula(256), sprintf('error_map/
%i_error.png',i));


    fprintf(sprintf('Imgs nb: %i \n', i));
end

mean_accuracy = 1.0-(accumulated_error/imgs_count)
```

Published with MATLAB® R2018b

1

10.3 Details of the zip-file

10.3.1 The folder “MC-CNN-fst-nopost”

“preprocess/modelfish2019_prep.m” was used to preprocess the data.

“train_match.lua” was used to train and validate the network on the training and validation set.

“test_accuracy.lua” was used to find the mean accuracy results on the test set.

10.3.2 The folder “Content-CNN-nopost”

“preprocess_modelfish” was used to preprocess the data.

“main_val.lua” was used to train and validate the network on the training and validation set.

“main_test.lua” was used to find the mean accuracy results on the test set.

10.3.3 The file “westlie_prosjektrapport.pdf”

Contains the project report written by the author, Storm Westlie.